Presentación

En esta práctica implementaremos un criptosistema de clave pública basado en curvas elípticas con el lenguaje de programación Python e implementaremos un intercambio de claves de Diffie-Hellman utilizando estas curvas.

Objetivos

Los objetivos de esta práctica son:

1. Aplicar los conocimientos de la aritmética modular a las cifras de clave pública.
2. Implementar un criptosistema de clave pública basado en curvas elípticas.
3. Implementar un sistema de intercambio de claves utilizando curvas elípticas.

Descripción de la Práctica a realizar

El protocolo de intercambio de claves de Diffie-Hellman que habéis trabajado en el módulo, se puede implementar en distintos tipos de grupos, no sólo en los grupos multiplicativos módulo un número primo. En particular se puede implementar utilizando como grupo los puntos de una curva elíptica. La idea fue propuesta en 1985 por Neal Koblitz y Victor Miller, de forma independiente, pero no fue hasta el 2000 cuando los algoritmos criptográficos basados en curvas elípticas empezaron a estandarizarse e implementarse.

Entre las ventajas de la criptografía de curva elíptica es que proporciona claves muchos más pequeñas que en cifrados como RSA o ElGamal. Así, en lugar de parámetros de 2.048 bits recomendados para Diffie-Hellman con la criptografía *tradicional*, la criptografía de curva elíptica permite parámetros de 256 bits.

Veamos a continuación cómo funciona.

1. Puntos de una curva elíptica

Comencemos definiendo formalmente una curva elíptica.

Definition 1 *Sea $p > 3$ un primo. Una curva elíptica E definida sobre $\mathbb{Z}_p$ es una ecuación*

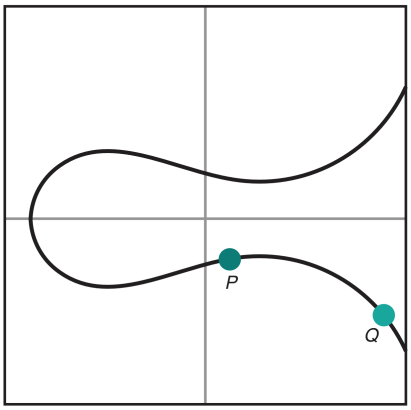
$$y^2 = x^3 + ax + b, \tag{1}$$

donde $a, b \in \mathbb{Z}_p$ satisfacen $4a^3 + 27b^2 \neq 0$. Escribimos $E/\mathbb{Z}_p$ para indicar que E está definido sobre $\mathbb{Z}_p$. Para simplificar la notación, escribiremos simplemente E.

La condición técnica $4a^3 + 27b^2 \neq 0$ asegura que la ecuación $x^3 + ax + b = 0$ no tiene raíz doble.

Geométricamente, en $\mathbb{R}$, una curva elíptica tiene una representación similar a la Figura 1.

Figura 1: Curva elíptica



Sea $E/\mathbb{Z}_p$ una curva elíptica. Decimos que un punto (x_1, y_1), donde $x_1, y_1 \in \mathbb{Z}_{p^e}$, es un punto de la curva E si (x_1, y_1) satisface la ecuación de la curva 1.

La curva incluye un punto *especial* adicional $\mathcal{O}$ llamado el *punto del infinito*. Este punto está definido en el código con la variable `PINFINITY`. Su propósito quedará claro más adelante en la práctica. Escribimos $E(\mathbb{Z}_p)$ para denotar el conjunto de todos los puntos de la curva E que están definidos sobre $\mathbb{Z}_p$, incluyendo el punto $\mathcal{O}$. Por ejemplo, consideremos la curva $E : y^2 = x^3 + 1$ definida sobre $\mathbb{Z}_{11}$. Así:

$$E(\mathbb{Z}_{11}) = \{\mathcal{O}, \quad (-1, 0), \quad (0, \pm 1), \quad (2, \pm 3), \quad (5, \pm 4), \quad (7, \pm 5), \quad (9, \pm 2)\}.$$

Esta curva tiene 12 puntos en $\mathbb{Z}_{11}$ y escribimos $|E(\mathbb{Z}_{11})| = 12$.

De hecho es conocido que el número de puntos en $E(\mathbb{Z}_p)$ es $p^e + 1$, en concreto

$$p + 1 - 2\sqrt{p} \leq |E(\mathbb{Z}_p)| \leq p + 1 + 2\sqrt{p}. \tag{2}$$

En el ejemplo anterior tiene exactamente $p + 1$ puntos.

Encontrar los puntos de una curva elíptica no es en general una tarea trivial si el parámetro p es relativamente grande, y existen algorismos específicos que optimizan esta tarea. En nuestro caso, nos centraremos en valores relativamente pequeños por lo que será suficiente una búsqueda exhaustiva de los diferentes valores. Así:

1. (1 punto) Implementar una función `ComputePoints` que dados los parámetros p, a y b, devuelva todos los puntos de la curva elíptica. Puesto que nuestro propósito no es utilizar parámetros que se utilizan en protocolos reales, podéis utilizar fuerza bruta para calcular raíces modulares, aunque en la realidad existen otros protocolos mucho más eficientes.

   Concretamente, la función que encontraréis en el esqueleto proporcionado, recibe los siguientes parámetros:

   - La tupla `curve` (a, b, p): corresponde a los parámetros a y b de la ecuación de la curva $y^2 = x^3 + ax + b$ y al número primo sobre el que se calculará la aritmética de la curva.
   - La función retornará el número de puntos (x, y) de la curva

2. (1 punto) Implementar una función `VerifyNumPoints` que compruebe que el número de puntos satisface la inecuación 2. Así, la función que encontraréis en el esqueleto proporcionado, recibe los siguientes parámetros:

   - La tupla `curve` (a, b, p): corresponde a los parámetros a y b de la ecuación de la curva $y^2 = x^3 + ax + b$ y al número primo sobre el que se calculará la aritmética de la curva.
   - El número de puntos `n` a verificar.
   - La función retornará $True$ si el número de puntos satisface la inecuación 2 o $False$ en caso contrario.

2. Suma de puntos de una curva elíptica

Existe una ley de grupo natural definida en los puntos de una curva elíptica. La operación de grupo se escribe aditivamente y utilizaremos el símbolo "$\boxplus$ "para denotar la suma de puntos.

Ahora, vamos a $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ ser dos puntos en $E(\mathbb{Z}_p)$. La suma $P \boxplus Q = (x_3, y_3)$ se define utilizando una de las tres reglas siguientes (dependiendo de los puntos a sumar):

- si $x_1 \neq x_2$, sea $s_c = \frac{y_1 - y_2}{x_1 - x_2}$ la pendiente de la cuerda que pasa por los puntos P y Q. Así, definimos
$$x_3 = s_c^2 - x_1 - x_2, \quad y_3 = s_c(x_1 - x_3) - y_1. \tag{3}$$
- si $x_1 = x_2$ y $y_1 = y_2$ (es decir, $P = Q$), pero $y_1 \neq 0$, utilizamos el método de la tangente. Sea $s_t = \frac{3x_1^2 + a}{2y_1}$ la pendiente de la tangente en P. Así, definimos
$$x_3 = s_t^2 - 2x_1, \quad y_3 = s_t(x_1 - x_3) - y_1.$$
- si $x_1 = x_2$ y $y_1 = -y_2$ entonces definimos $P \boxplus Q = \mathcal{O}$.

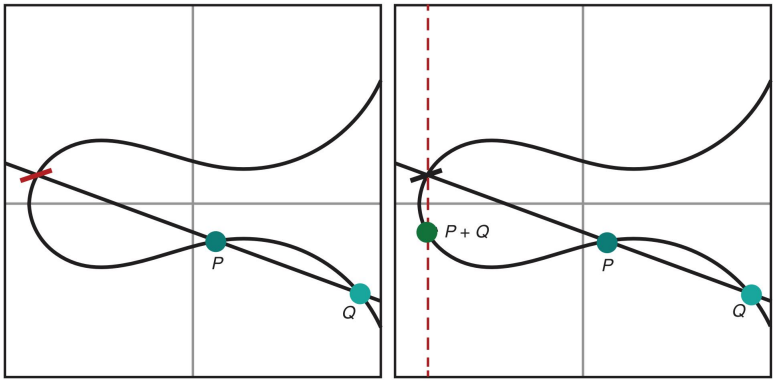
NOTA: Tened en cuenta que $P \boxplus \mathcal{O} = P$.

1. (1 punto) Implementa una función `AddPoints` que dados dos puntos de una curva elíptica, y los parámetros p, a y b, calcule la suma de los dos puntos siguiendo las reglas anteriores. Concretamente, la función que encontraréis en el esqueleto proporcionado, recibe los siguientes parámetros:
   - La tupla `curve` (a, b, p): corresponde a los parámetros a y b de la ecuación de la curva $y^2 = x^3 + ax + b$ y al número primo sobre el que se calculará la aritmética de la curva.
   - La variable `P` formada por la tupla (x_1, y_1) correspondiente al primer punto a sumar.
   - La variable `Q` formada por la tupla (x_2, y_2) correspondiente al segundo punto a sumar.
   - La función retornará una tupla `suma` que corresponde a la suma de los dos puntos de la curva elíptica que se proporcionan como parámetro.

Por completitud, interpretaremos su significado geométrico en $\mathbb{R}$, que nos sirve para ver la intuición. Esta parte es puramente informativa, no hace falta reproducirla, pero altamente aconsejable entenderla. Dados dos puntos P y Q de la curva elíptica:

1. Dibuja una recta que pase por dos puntos que quieras sumar. La recta interseca a la curva en otro punto.
2. Dibuja una recta vertical desde este nuevo punto. La línea vertical interseca a la curva en otro punto.
3. Este punto es el resultado de sumar los dos puntos originales.

Figura 2: Suma de puntos de una curva elíptica



La Figura 2 representa la suma de los puntos P y Q en una curva elíptica.

Notar, sin embargo, que hay dos casos especiales en los que esta regla no funciona. En efecto:

- ¿Cómo se suma un punto a sí mismo? La respuesta es trazar la tangente a ese punto (en lugar de trazar una recta entre los dos puntos).
- ¿Y si la recta que trazamos en el paso 1 (o en el paso 2) no toca la curva en ningún otro punto? Es el caso, como hemos vismo anteriormente, en que la suma es el punto del infinito $\mathcal{O}$.

3. Estructura de grupo

La suma definida en la sección anterior convierte al conjunto $E(\mathbb{Z}_p)$ en un grupo. El elemento identidad es el punto en el infinito. Cada punto $\mathcal{O} \neq P = (x_1, y_1) \in E(\mathbb{Z}_p)$ tiene un inverso aditivo, a saber $-P = (x_1, -y_1)$. Recordad que el punto en el infinito $\mathcal{O}$ está definido en el código con la variable `PINFINITY`.

Para un punto $P \in E(\mathbb{Z}_p)$ escribimos $2P = P \boxplus P$, $3P = P \boxplus P \boxplus P$, y más generalmente, $nP = (n-1)P \boxplus P$ para cualquier entero positivo n.

1. (1 punto) Implementa una función `SelfProductPoint` que dado un punto de una curva elíptica, los parámetros p, a y b, y un entero n, calcule nP. En particular, la función que encontraréis en el esqueleto proporcionado, recibe los siguientes parámetros:
   - La tupla `curve` (a, b, p): corresponde a los parámetros a y b de la ecuación de la curva $y^2 = x^3 + ax + b$ y al número primo sobre el que se calculará la aritmética de la curva.
   - La variable `P` formada por la tupla (x_1, y_1) correspondiente a un punto de la curva elíptica.
   - La variable n, un valor entero.
   - La función retornará una tupla `product` que corresponde al cálculo de nP.
2. (1 punto) Utiliza las funciones anteriores para implementar una función `IsGroup` que dada una curva elíptica definida por los parámetros p, a y b, compruebe que los puntos de una curva elíptica tienen estructura de grupo, esto es que dados cualquier par de puntos $P \in E(\mathbb{Z}_p)$ y $Q \in E(\mathbb{Z}_p)$, $P \boxplus Q \in E(\mathbb{Z}_p)$.
   - La tupla `curve` (a, b, p): corresponde a los parámetros a y b de la ecuación de la curva $y^2 = x^3 + ax + b$ y al número primo sobre el que se calculará la aritmética de la curva.
   - La función retornará `True` si los puntos de la curva verifican una estructura de grupo, y `False` en caso contrario.
3. (1 punto) Implementa una función `OrderPoint` que calcule el orden de cada uno de los puntos de la curva elíptica, esto es, el mínimo k tal que $kP = \mathcal{O}$. Observa que hay puntos que generan la curva. En concreto, la función que encontraréis en el esqueleto proporcionado, recibe los siguientes parámetros:
   - La tupla `curve` (a, b, p): corresponde a los parámetros a y b de la ecuación de la curva $y^2 = x^3 + ax + b$ y al número primo sobre el que se calculará la aritmética de la curva.
   - La función retornará una lista `PointsOrders` que contendrá tuplas (`P`,ord(`P`)), donde ord denota el orden del punto `P`.

4. Diffie-Hellman sobre una curva elíptica

Ahora que hemos construido un grupo sobre curvas elípticas, podemos instanciar el mismo algoritmo de intercambio de claves Diffie-Hellman sobre este grupo. Así, Alicia y Bernardo quieren encontrar una clave k en común, para ello proceden de la siguiente manera.

Generación de las claves:

Alícia y Bernardo usarán ciertos parámetros públicos: un primo p, un par de valores (a, b) que definen la ecuación de la curva elíptica como hemos explicado anteriormente, y finalmente un generador P del grupo de puntos de la correspondiente curva elíptica. A continuación, seguirán el siguiente procedimiento:

- Cada participante genera un número aleatorio entre 1 y p, que se convierte en su clave privada. Así, Alicia tendrá α como clave privada y Bernardo tendrá β como clave privada.
- Tanto Alicia como Bernardo calculan su clave pública como sigue, $A_{pub} = \alpha P$ y $B_{pub} = \beta P$, para Alicia y Bernardo respectivamente.

Cálculo de la clave compartida:

Alicia y Bernardo podrán intercambiar una clave compartida siguiendo el siguiente procedimiento:

- Alícia calculará la clave compartida usando su clave privada y la clave pública de Bernardo mediante: αPub_B que dará como resultado $\alpha\beta P$.
- Bernardo calculará la clave compartida usando su clave privada y la clave pública de Alicia mediante: βPub_A que dará como resultado $\beta\alpha P$.
- De esta manera, la clave compartida obtenida por Alicia será la misma que la clave compartida obtenida por Bernardo.

1. (2 puntos) Implementa una función `GenKey` que genere una clave pública y una clave privada para un usuario: La función recibirá como parámetros:
   - La tupla `curve` (a, b, p) que determinará la ecuación de la curva $y^2 = x^3 + ax + b$ tal que $4a^3 + 27b^2 \neq 0$
   - Un generador P de la curva.

   La función retornará una tupla (pub, priv) con la clave pública y la privada generada.
2. (2 puntos)Implementa una función `SharedKey` que calcule la clave compartida: La función recibirá como parámetros:
   - La tupla `curve` (a, b, p) que determinará la ecuación de la curva $y^2 = x^3 + ax + b$ tal que $4a^3 + 27b^2 \neq 0$
   - La clave privada del usuario que quiere generar la clave compartida.
   - La clave pública del usuario con el que quiere compartir la clave.

NOTA: Tened en cuenta que los tests asociados a este ejercicio no verifican individualmente las funciones `GenKey` y `SharedKey`. En su lugar, los tests simulan el intercambio de claves y verifican que tanto Alicia como Bernardo obtienen la misma clave compartida. Os recomendamos consultar los tests asociados en el archivo de test.

Criterios de valoración

Formato y fecha de entrega

La puntuación de cada ejercicio se encuentra detallada en el enunciado.

En ningún caso no se puede usar ninguna librería de Python que no esté incluida con un `importe` en el esqueleto de la práctica que se os proporciona.

Por otro lado, hay que remarcar que el código que se libre de la práctica tiene que contener los comentarios necesarios para poderlo seguir y entender. En caso de que el código no incluya comentarios, la corrección de la práctica se realizará únicamente de forma automática y no se proporcionará una corrección detallada. La no inclusión de comentarios también puede ser motivo de reducción de la nota.

La fecha máxima de entrega de la práctica es el **18/06/2023** (a las 24 horas).

Junto con el enunciado de la práctica encontraréis el esqueleto de la misma (fichero con extensión .py). Este fichero contiene las cabeceras de las funciones que hace falta que implementáis para resolver la práctica. Este mismo fichero es el que tenéis que enviar una vez codificáis todas las funciones. No se tiene que enviar el fichero comprimido. No se aceptarán ficheros en otros formatos que no sean .py.

Adicionalmente, también os proporcionaremos un fichero con tests unitarios para cada una de las funciones que hace falta que implementéis. Podéis usar estos testos para comprobar que vuestra implementación gestiona correctamente los casos principales, así como para obtener más ejemplos concretos de lo que se espera que devuelvan las funciones (más allá de los que ya se proporcionen en este enunciado). Notar, sin embargo, que los tests no son exhaustivos (no se prueban todas las entradas posibles de las funciones). Recordar que no se puede modificar ninguna parte del fichero de tests de la práctica.

La entrega de la práctica constará de un fichero Python (extensión .py) donde hayáis incluido vuestra implementación.