UNIVERSIDAD
REY JUAN CARLOS

DOBLE GRADO INGENIERÍA DEL SOFTWARE Y MATEMÁTICAS

Curso Académico 2018/2019

Trabajo Fin de Grado

Criptografía sobre curvas elípticas. Cifrado basado en identidades.

Autor : José Francisco Rodríguez Peña

Tutor : Ángel Luis Pérez del Pozo

Objetivo

El objetivo de este trabajo es tener un primer contacto con el estudio de la criptografía, haciendo énfasis en la criptografía de clave pública y la teoría necesaria para definir la criptografía sobre curva elíptica. Finalmente se estudiará un esquema de cifrado basado en identidades sobre curvas elípticas.

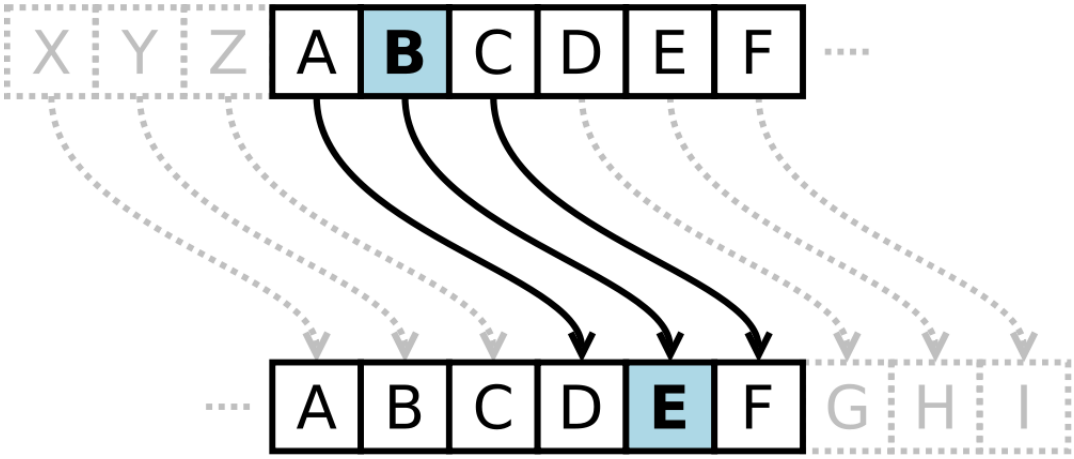
Índice general

Capítulo 1

Introducción

La criptografía es la parte de la ciencia encargada de tratar la información de manera segura en presencia de adversarios, una de sus ramas es el cifrado de datos, que se encarga de poder intercambiar mensajes manteniendo la *confidencialidad*. Para esto se hace uso de alguna transformación del mensaje de forma que no pueda ser recuperada ninguna información útil y que solo los receptores legítimos del mensaje puedan volver a transformarlo en el mensaje original, estas transformaciones son llamadas *cifrado* y *descifrado* respectivamente.

Las primeras técnicas para cifrar mensajes consistían en sustituir todas las letras del mensaje mediante alguna tabla o relación, en concreto en el cifrado César se alternaban por su correspondiente en el alfabeto desplazada un número de posiciones, este número era la clave del cifrado y para descifrar bastaba con realizar el mismo desplazamiento en sentido inverso, dada la poca cantidad de claves posibles para este cifrado es muy fácil de romper por fuerza bruta, sin embargo una de las formas más ingeniosas para romperlo consiste en analizar la distribución de letras en el idioma del mensaje y, si este es suficientemente largo, se puede detectar el desplazamiento en esa distribución.



En el siglo XV aparece el cifrado Vigènere que consistía en una palabra clave que se usaba para desplazar las letras del mensaje, por ejemplo si la clave es 'ABC' los desplazamientos serían 1, 2 y 3 posiciones respectivamente, y se repetía esta clave hasta acabar por completo el mensaje.
La criptografía estuvo ligada fuertemente al uso militar y durante los siglos XIX y XX tuvo una importancia fundamental con la maquina Enigma, en este periodo de tiempo se invirtió un gran esfuerzo en desarrollar este área dando lugar a grandes avances.
Posteriormente a la Segunda Guerra Mundial se realizaron muchos estudios teóricos y en los años 70 se crearon algunos estándares de algoritmos de cifrado, con la evolución de la informática desde entonces existe una actualización constante de algoritmos cada vez mas seguros ya que la capacidad de cómputo es mayor.
Es habitual agrupar los algoritmos según la cantidad de claves que son necesarias al cifrar y descifrar:

- Criptografía simétrica: Estos métodos utilizan la misma clave para cifrar y descifrar mensajes, las distintas partes que intercambian información deben conocer esta clave de antemano, existen distintos algoritmos que ponen esto en practica, uno de los mas conocidos entre los métodos recientes es DES (Data Encryption Standard), diseñado en los años 70 y reemplazado a día de hoy por AES (Advanced Encryption Standard).

- Ciptografía asimétrica: También conocida como criptografía de clave pública, se basa en un par de claves, una pública que se debe dejar a disposición de cualquiera y una privada que se debe mantener en secreto. Este tipo de métodos son utilizados para el cifrado basado en identidad y el cifrado basado en certificados.

Capítulo 2

Herramientas matemáticas

2.1. Grupo cíclico

Sea G un grupo, se dice que G es un *grupo cíclico* si existe un elemento del grupo $g \in G$ de forma que $G = \langle g \rangle$, es decir, cualquier elemento del grupo puede expresarse como g^n (o ng si es un grupo aditivo) para algún $n \in \mathbb{N}$, llamamos entonces a g *generador* de G. Un ejemplo habitual de grupo cíclico es el grupo aditivo de los números enteros módulo n, denotado habitualmente como $\mathbb{Z}/n\mathbb{Z}$, tomando como generador el 1.
Lema: Cada grupo cíclico finito de orden n es isomorfo al grupo $\mathbb{Z}/n\mathbb{Z}$.

2.2. Logaritmo discreto

Para algunos algoritmos de critografía necesitamos una función que sea fácil de aplicar pero difícil de invertir a nivel de tiempo de cómputo, por ello vamos a ver el caso del logaritmo discreto.
Dado un grupo cíclico finito G con base $\langle g \rangle$, tenemos que:

$$y = g^x \iff x = \log_g(y)$$

Por un lado tenemos la *exponenciación discreta*, es decir, g^x cuando x es conocido, esta operación se puede calcular con el algoritmo recursivo de exponenciación binaria, con complejidad $\mathcal{O}(\log n)$.

No obstante el problema de calcular $x = \log_g(y)$ es más difícil, el algoritmo Baby-step/Giant-step tiene complejidad $\mathcal{O}(\sqrt{p})$, asumiendo que el grupo tiene orden primo p.

2.3. Exponenciación binaria

Para calcular g^x, tenemos que hacer la operación recursiva F definida de la siguiente forma:

$$F(x^n) = \begin{cases} x \text{ si } n = 1 \\ \left(F(x^{\frac{n}{2}})\right)^2 \text{ si } n \text{ es par} \\ F(x^{(n-1)}) \cdot x \text{ si } n \text{ es impar} \end{cases}$$

Puesto que en cada operación reducimos el exponente n a la mitad, tenemos que realizar $\lceil \log_2(n) \rceil$ operaciones, por tanto este problema se puede resolver en $\mathcal{O}(\log n)$.

2.4. Algoritmo Baby-step/Giant-step

Queremos conocer x en la ecuación $h = g^x$, sea p el orden del grupo G.

Descompondremos nuestra solución en:

$$x = x_0 + x_1 \lceil \sqrt{p} \rceil$$

Puesto que $x < p$, entonces $0 \leq x_0, x_1 < \lceil \sqrt{p} \rceil$.

En primer lugar calcularemos los *baby-step*, esto es:

$$g_i = g^i \; t.q. \; 0 \leq i < \lceil \sqrt{p} \rceil$$

Tras ello calcularemos los *giant-step*, esto es:

$$h_j = hg^{-j\lceil \sqrt{p} \rceil} \; t.q. \; 0 < j < \lceil \sqrt{p} \rceil$$

Una forma eficiente de almacenar estos resultados es con una tabla hash almacenando los pares (g_i, i) y (h_j, j) respectivamente.

Ahora intentamos encontrar una coincidencia tal que $g_i = h_j$, si esto ocurre tenemos que:

$$x_0 = i, x_1 = j$$

Ya que

$$g^i = hg^{-j\lceil\sqrt{p}\rceil} \Rightarrow g^{i+j\lceil\sqrt{p}\rceil} = h$$

Dado que para calcular los *baby-step* y los *giant-step* son necesarias como mucho $2\lceil\sqrt{p}\rceil$ operaciones, y para buscar una coincidencia solo es necesario tener almacenados los pares (clave,valor) de forma (i, g_i) y (h_j, j) para ver si existe la clave g_i en el segundo hash map, por tanto encontrando una coincidencia, puesto que comprobar una clave es $\mathcal{O}(1)$ y se recorren únicamente los g_i tenemos de nuevo $\lceil\sqrt{p}\rceil$ operaciones, por lo que este algoritmo es de complejidad $\mathcal{O}(\sqrt{p})$.

Capítulo 3

Curvas elípticas

3.1. Definiciones

3.1.1. Plano proyectivo

Sea K un cuerpo, *el plano proyectivo* $\mathbb{P}^2(K)$ definido sobre K son la tripletas:

$$(X, Y, Z)\ t.q.\ X, Y, Z \in K$$

Donde X, Y, Z no son simultáneamente 0 y definimos la relación de equivalencia:

$$(X, Y, Z) \equiv (X', Y'.Z') \iff \exists \lambda \in K\ t.q.\ X = \lambda X',\ Y = \lambda Y',\ Z = \lambda Z'$$

3.1.2. Curva elíptica

Una *curva elíptica* se define sobre el conjunto de soluciones de la ecuación Weierstrass homogénea de la forma:

$$E:\ Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{3.1}$$

donde $a_1, a_2, a_3, a_4, a_6 \in K$, esta notación es conocida como forma larga Weierstrass. Dicha curva debe ser no singular, es decir, si la escribimos de la forma $F(X, Y, Z) = 0$, las derivadas parciales no deben ser cero simultáneamente en ningún punto de la curva. El conjunto de los *puntos K-racionales* en E son la solución de la ecuación 3.1 y lo denotaremos $E(K)$. Destacar que la curva tiene exactamente un punto racional con coordenada Z igual a cero, el $(0, 1, 0)$, este es el punto en el infinito y lo denotaremos como $\mathcal{O}$.

3.2. Paso al plano afín

Si tomamos la ecuación 3.1 y definimos el cambio de forma que:

- El punto $\mathcal{O}$ se toma como el infinito en cualquier dirección en K^2.
- Un punto en el plano proyectivo con $Z \neq 0$ como $\left(\frac{X}{Z}, \frac{Y}{Z}\right)$ en K^2.
- Para volver al plano proyectivo se toma cualquier $Z \in K$ y se calcula $(X \cdot Z, Y \cdot Z, Z)$.

Con este cambio la curva queda definida por:

$$E:\ Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \tag{3.2}$$

con $a_1, a_2, a_3, a_4, a_6 \in K$

Es posible demostrar (se demuestra en [3, pag. 25-26]) que dada cualquier curva de la forma 3.2, existe un isomorfismo a la curva que llamaremos *forma corta Weierstrass*:

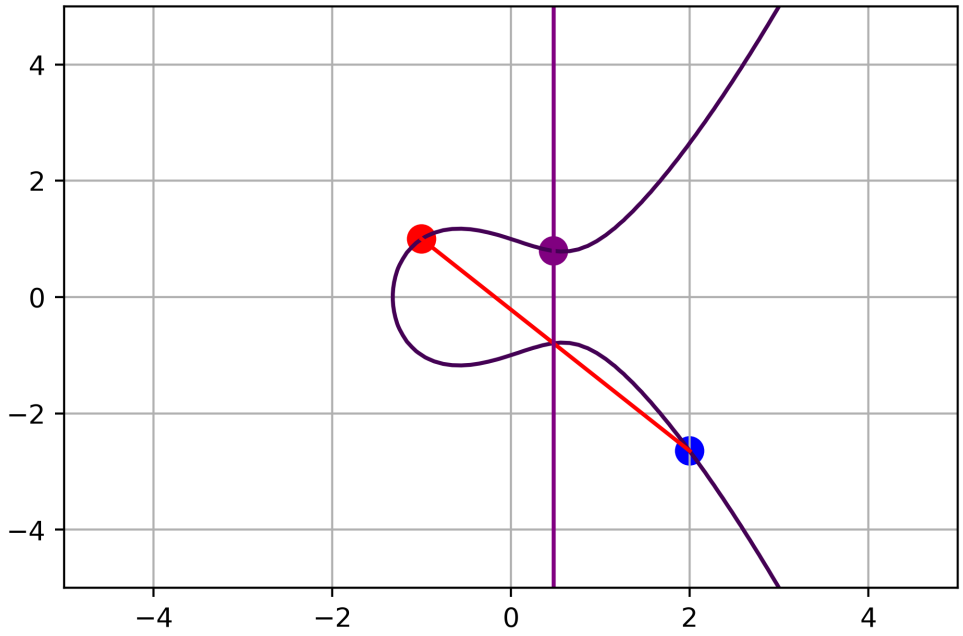
$$E:\ Y^2 = X^3 + aX + b \tag{3.3}$$

donde $a, b \in K$

3.3. Ley de grupo

Definimos la *suma de puntos*, sea $P, Q \in E$, entonces:

- Se toma la recta que une P y Q.
- Se calcula el punto de corte de la recta y la curva E al que llamaremos R.
- $P + Q$ se define como el simétrico de R respecto al eje x.



Observación:

[2] $P = P + P$ se calcula usando la recta tangente a la curva en el punto P, por lo que si la tangente es una recta vertical se considera que $P + P = \mathcal{O}$.

De manera algebraica definimos la *suma de puntos* como:

Sea la curva de la forma

$$E: \ Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$

y los puntos de la curva $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$. Entonces $-P_1 = (x_1, -y_1 - a_1x_1 - a_3)$.

Tomamos λ,μ :

$$\begin{cases} x_1 \neq x_2 & \begin{cases} \lambda = \dfrac{y_2 - y_1}{x_2 - x_1} \\ \mu = \dfrac{y_1x_2 - y_2x_1}{x_2 - x_1} \end{cases} \\ x_1 = x_2, P_2 \neq P_1 & \begin{cases} \lambda = \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} \\ \mu = \dfrac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x_1 + a_3} \end{cases} \end{cases}$$

Entonces si $P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O}$ se puede calcular x_3 e y_3 como:

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2,$$
$$y_3 = -(\lambda + a_1)x_3 - \mu - a_3$$

Nota:

Para un entero positivo m tenemos que $[m]$ denota la aplicación de la *multiplicación-por-m* de la curva en si misma. Esta aplicación lleva un punto P a $\underbrace{P + P + ... + P}_{m\ veces}$.

Esta aplicación será la base de la criptografía sobre curvas elípticas ya que se considera fácil de computar pero difícil de invertir como hemos visto en el capítulo 2.2.

3.4. Curvas elípticas sobre cuerpos finitos

Sobre un cuerpo finito $\mathbb{F}_q$ el número de puntos racionales de la curva es finito y queda determinado por:

$$\#E(\mathbb{F}_q) = q + 1 - t$$

Donde t se conoce como la traza de Frobenius.

Teorema de Hasse: t cumple que $|t| < 2\sqrt{q}$.

La aplicación q^{th}-potencia de Frobenius de la curva E definida sobre $\mathbb{F}_q$ es:

$$\varphi \begin{cases} E(\overline{\mathbb{F}_q}) \to E(\overline{\mathbb{F}_q}) \\ (x, y) \to (x^q, y^q) \\ \mathcal{O} \to \mathcal{O} \end{cases}$$

Esta aplicación es un endomorfismo que respeta la operación de la ley de grupo $\varphi(P + Q) = \varphi(P) + \varphi(Q)$

Capítulo 4

Intercambio de claves

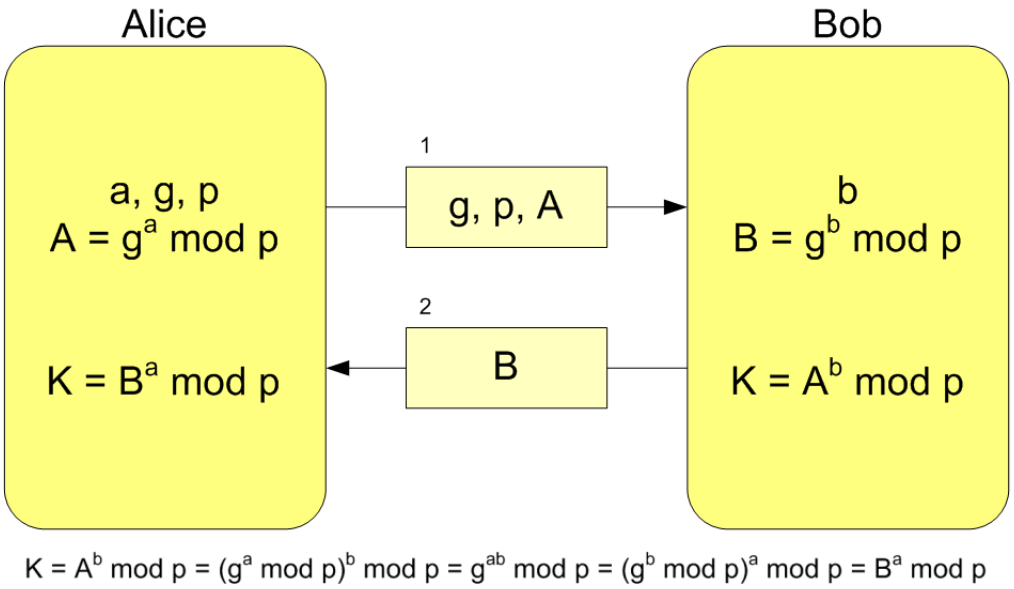
En algunos algoritmos es necesario que partes no autentificadas lleguen a intercambiar claves por un canal no seguro, esto supuso un problema que solventaron Whitfield Diffie y Martin Hellman con su protocolo, se utiliza para acordar claves simétricas en una sesión y esto posteriormente permite los protocolos de comunicación autentificada. Su seguridad tiene origen en el problema del logaritmo discreto ya que, como hemos visto antes en el capítulo2.2, es fácil de computar y difícil de invertir.
Supongamos que tenemos dos usuarios finales que quieren comunicarse entre sí, sean estos Alice y Bob, el protocolo Difie-Hellman en una versión simple sigue los siguientes pasos:

- Fijar un primo p y un generador del grupo $g \in \mathbb{Z}_p^*$. Estos valores son conocidos para cualquiera que tenga acceso al canal donde se intercambian.

- Alice elige un $a \in \mathbb{Z}_{p-1}$, computa $A = g^a \ mod \ p$ y lo envía a Bob.

- Bob elige un $b \in \mathbb{Z}_{p-1}$, computa $B = g^b \ mod \ p$ y lo envía a Alice.

Con esta información ambos extremos son capaces de computar $K = (g)^{ab} \ mod \ p$, ya que Alice puede calcular $K = B^a \ mod \ p$ y Bob $K = A^b \ mod \ p$. Se utilizará K como la clave compartida.



No obstante cualquier atacante que quiera conocer la clave K en el canal solo tiene acceso a g, p, A y B, por lo que para computar la clave tiene que calcular $a = log_p(A)$ y $b = log_p(B)$ siendo este el logaritmo discreto.

Existen ataques a este protocolo que no están basados en resolver el logaritmo discreto sino en aprovechar que las partes no están autentificadas y por tanto hacer creer a ambos que él es el otro extremo, esto es conocido como *Man-in-the-middle*.

De la misma forma que se aplica este protocolo en su versión original es posible aplicarla sobre el grupo aditivo de las curvas elípticas, este protocolo es conocido como EC-DH y una de las ventajas que nos proporciona es que utilizando la compresión de punto se puede reducir la cantidad de información necesaria para realizar el protocolo con la misma seguridad, reduciendo el ancho de banda.

Capítulo 5

Esquemas de clave pública

Los esquemas de clave pública se basan en una pareja de claves (clave pública, clave privada), estas claves a diferencia de la criptografía simétrica requieren de su pareja para realizar la operación inversa.
Existen dos usos fundamentales de este tipo de esquemas, el primero donde dos extremos quieren intercambiar información, el emisor usará la clave pública del receptor para cifrar el mensaje y enviarlo, el receptor al ser el único conocedor de su clave privada puede recuperar el mensaje original, el segundo caso donde, para demostrar la autenticidad de un emisor, éste cifra el mensaje con su clave privada y cualquier receptor comprueba la autenticidad del emisor cuando lo descifra con su clave pública, este caso es el que se conoce como *firma digital*.

5.1. Funciones Hash

Hasta este punto hemos visto como se puede mantener la confidencialidad de un mensaje para que cualquier atacante no pueda obtener la información aún teniendo acceso al canal por el que se transmite, no obstante debemos tener en cuenta el escenario donde el atacante manipula el mensaje, por tanto debemos preocuparnos de la integridad, para comprobar que un mensaje no ha sido modificado es habitual hacer uso de *funciones hash*.
Una *funcion hash* es una aplicación que lleva una cadena de bits de cualquier longitud a una cadena de bits de longitud fija. Estas funciones tienen una serie de propiedades deseables, si tenemos la función hash $h(x) = y$,sus propiedades deberían ser:

- Resistencia a la preimagen: debe ser difícil a efectos prácticos que conociendo y se pueda

calcular x.

- Resistencia a colisiones: debe ser difícil encontrar dos valores x y x' tal que $h(x) = h(x')$ pero $x \neq x'$.
- Resistencia a la segunda preimagen: dado un x debe ser difícil encontrar otro x' tal que $h(x) = h(x')$ pero $x \neq x'$.

Más adelante veremos como estas funciones son necesarias a la hora de comprobar la integridad de la información, ya que cualquier pequeña modificación en el mensaje se reflejaría al calcular el hash del mismo.

5.2. RSA

Supongamos dos usuarios finales Alice y Bob, Bob quiere enviar un mensaje a Alice y para ello se va a usar el algoritmo RSA, este consiste en:

- Alice elige dos números primos grandes p y q y calcula $N = p \cdot q$.
- Alice elige un exponente e de forma que $mcd(e, (p-1) \cdot (q-1)) = 1$.
- En el calculo del exponente para descifrar, Alice usa el algoritmo de Euclides extendido para conocer d en la ecuación:

$$e \cdot d \equiv 1 \; mod \; (p-1)(q-1) \tag{5.1}$$

En este punto la clave pública de Alice es el par (N, e) y su clave privada es la tripleta (d, p, q). Por tanto para intercambiar el mensaje se realizan los siguientes pasos:

- Bob quiere mandar su mensaje m, por lo que cifra el mensaje para obtener c calculando $c = m^e \; mod \; N$.
- Alice recibe c por lo que para poder leer el mensaje tiene que calcular m de la forma $m = c^d \; mod \; N$.

Demostración: $m = c^d \; mod \; N$

El orden del grupo $(\mathbb{Z}/N\mathbb{Z})^*$ viene dado por $\phi(N) = (q-1)(p-1)$, por tanto:

$$\forall x \in (\mathbb{Z}/N\mathbb{Z})^* : \; x^{(q-1)(p-1)} \equiv 1 \; mod \; N \tag{5.2}$$

Ahora bien, para algún entero s se tiene que $ed - s(q-1)(p-1) = 1$ puesto que se cumple la ecuación 5.1. Por tanto:

$$\begin{aligned} c^d &= m^{ed} \; mod \; N \\ &= m^{1+s(q-1)(p-1)} \; mod \; N \\ &= m \cdot m^{s(q-1)(p-1)} \; mod \; N \\ &= m \; mod \; N \end{aligned}$$

La seguridad de este algoritmo reside en el problema de factorización, ya que si logramos factorizar N como $p \cdot q$ entonces podremos calcular d, no obstante este problema se considera complejo a nivel de cómputo.

5.3. Firma digital

La firma digital es un mecanismo que se usa para verificar el origen de un mensaje, es decir, asegura la autenticación del emisor y el no repudio. Existen varias implementaciones diferentes de este mecanismo, pero en este caso nos vamos a centrar en la firma con RSA.
Supongamos dos usuarios finales Alice y Bob, Alice quiere enviar un mensaje a Bob, pero Bob tiene que poder asegurarse que el mensaje procede de Alice y nadie lo ha modificado, para ello van a usar la firma digital, Alice firmaría el mensaje:

- Alice aplica una función hash h al mensaje m, quedandose con $r = h(m)$.
- Alice usa su clave privada (d, p, q) para cifrar r, esta será la firma del mensaje $S = m^d \; mod \; p \cdot q$.
- Alice envía el mensaje, que no tiene porque estar cifrado, junto con la firma a Bob.

Una vez firmado el mensaje Bob podrá verificar la firma de la siguiente forma:

- Bob recibe el mensaje y la firma de Alice y usa la firma para calcular $r_s = S^e \; mod \; N$ con la clave pública de Alice (e, N).
- Bob aplica la misma función hash que Alice al mensaje $r_m = h(m)$.
- Si $r_s = r_m$ entonces el mensaje queda verificado.

Dado que Alice tuvo que usar su clave privada para cifrar la firma, no es posible modificar la firma S sin que invalide la verificación, de la misma forma que ocurre en el mensaje ya que no podrías cifrar el nuevo hash.

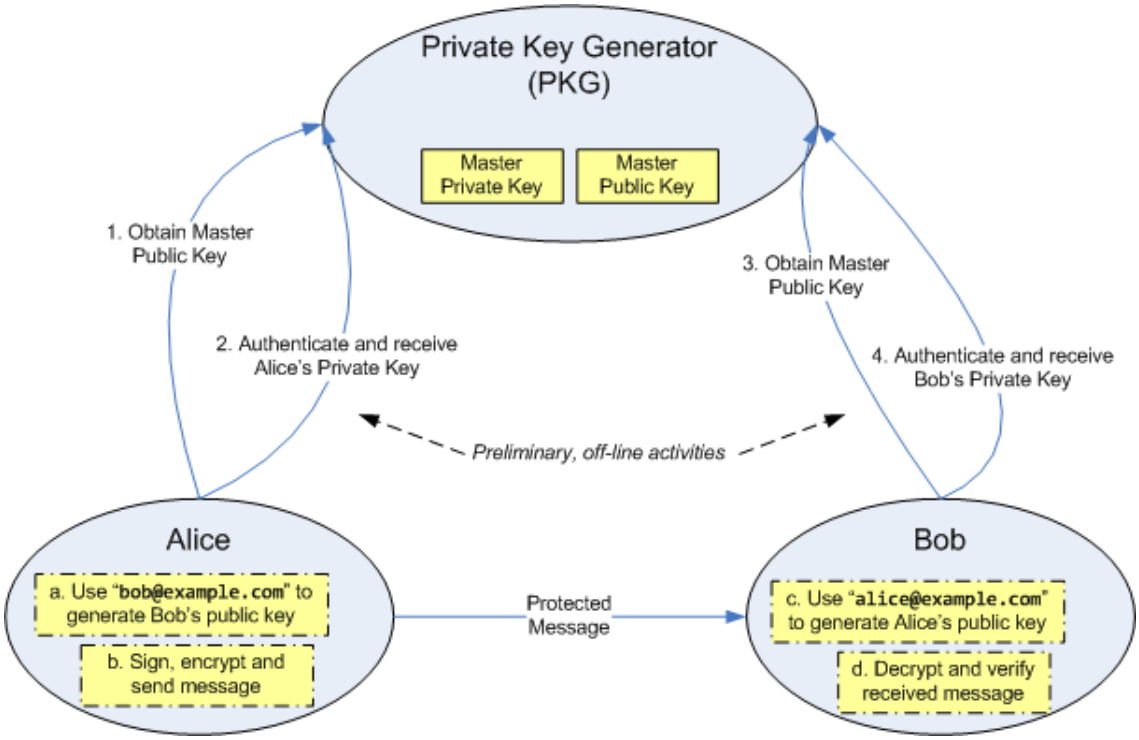
Capítulo 6

Cifrado basado en identidad

El cifrado basado en identidad es un esquema de clave pública donde la clave de un usuario se obtiene a partir de una información que lo identifique, ya sea su email, número de teléfono, etc...

Este esquema necesita de un generador de claves privadas (PKG), este provee a los usuarios de su clave privada y les da la forma de conocer la clave pública de los demás usuarios a partir de su identidad, el proceso que realiza un usuario final para conocer su clave privada y la clave pública de los demás solo se necesita aplicar una vez.

Si dos usuarios finales quieren intercambiar un mensaje, por ejemplo Alice quiere enviar un mensaje a Bob, entonces Alice usa la clave pública maestra con la identidad de Bob para cifrar el mensaje y lo envía, luego Bob usará su clave privada para descifrar el mensaje.



6.1. Emparejamiento

Dados dos grupos cíclicos aditivos G_1, G_2 de orden primo p y un grupo cíclico G_T escrito multiplicativamente, un *emparejamiento* es una aplicación $e : G_1 \times G_2 \rightarrow G_T$ que cumple:

- Bilinearidad:

$$\forall a, b \in F_p^*, \forall P \in G_1, \forall Q \in G_2 : e(aP, bQ) = e(P, Q)^{ab}$$

- No degenerado:

$$\forall P \in G_1 \; P \neq 0, \exists Q \in G_2 \; t.q. \; e(P, Q) \neq 1$$

$$\forall Q \in G_2 \; Q \neq 0, \exists P \in G_1 \; t.q. \; e(P, Q) \neq 1$$

- Computable: existe un algoritmo eficiente para computar e.

Capítulo 7

Conclusiones

Bibliografía

[1] D. Boneh and M. K. Franklin. *Identity-Based Encryption from the Weil Pairing Advances in Cryptology*. Springer-Verlag, 2001.

[2] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag, 1991.

[3] N. Smart. *Cryptography: An Introduction (3rd Edition)*. McGraw-Hill College, 2003.

[4] L. C. Washington. *Elliptic Curves: Number Theory And Cryptography (2n Edition)*. Chapman and Hall/CRC, 2008.

Capítulo 8

Anexo

```
import numpy as np
import math
import matplotlib.pyplot as plt
# Siempre en la forma corta de Weierstrass
class curva:
    a = -1
    b = 1
    #lista de diccionarios {nombre,x,signo,[color]}
    puntos = []
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def dibuja(self):
        y, x = np.ogrid[-5:5:100j, -5:5:100j]
        plt.contour(x.ravel(), y.ravel(), pow(y, 2) - pow(x, 3) - x * sel
        plt.grid()

    def figura(self):
        plt.show()

    def limpiar_p(self):
```

```
        self.puntos=[]
    def add_p(self,punto):
        self.puntos.append(punto)
    def dibuja_p(self):
        X=[punto['x'] for punto in self.puntos]
        Y=[self.calcula_y(punto) for punto in self.puntos]
        colors=[punto['color'] if 'color' in punto.keys() else 'purple' f
        plt.scatter(X,Y,c=colors,s=100)

    def calcula_y(self,punto):
        return punto['signo']*math.sqrt(pow(punto['x'], 3) + punto['x'] *

    def suma_p(self,nombre1,nombre2):
        A=next(item for item in self.puntos if item["nombre"] == nombre1)
        B=next(item for item in self.puntos if item["nombre"] == nombre2)
        A_x=A['x']
        A_y=self.calcula_y(A)
        B_x=B['x']
        B_y=self.calcula_y(B)
        if A_x != B_x:
            lamda=(B_y-A_y)/(B_x-A_x)
            mu=(A_y*B_x-B_y*A_x)/(B_x-A_x)
        elif A_x == B_x and A_y != B_y:
            lamda=(3*pow(A_x,2)+self.a)/(2*A_y)
            mu=(-pow(A_x,3)+self.a*A_x+2*self.b)/(2*A_y)
        else:
            return {'nombre':'inf'}
        P_x=pow(lamda,2)-A_x-B_x
        P_y=-lamda*P_x-mu
        punto = {'nombre':'%s + %s'%(nombre1,nombre2),'x':P_x, 'signo':1
        return punto
```

```
#El primer punto debe tener menor x
def dibuja_suma_p(self,nombre1,nombre2):
    A=next(item for item in self.puntos if item["nombre"] == nombre1)
    B=next(item for item in self.puntos if item["nombre"] == nombre2)
    suma=self.suma_p(nombre1,nombre2)
    y_init=self.calcula_y(A) if min(A['x'],suma['x'])==A['x'] else -s
    y_fin=self.calcula_y(B) if max(B['x'],suma['x'])==B['x'] else -se
    plt.plot([min(A['x'],suma['x']),max(B['x'],suma['x'])], [y_init,y
    plt.axvline(x=suma['x'],c='purple')

def guarda_plot(directorio="",nombre="Curva",extension="png"):
    plt.savefig("%s%s.%s"%(directorio+"/"if directorio!="" else "",no
```