Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Criptografía de Curva Elíptica: Abriendo la caja de Pandora

Elliptic Curve Cryptography: Opening Pandora's box

Luis Marcelo Chinea Rangel

La Laguna, 11 de julio de 2024

Dña. **Pino Caballero Gil**, Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

D. **Óscar Cigala Álvarez**, investigador asociado al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Criptografía elíptica: abriendo la caja de Pandora*”

ha sido realizada bajo su dirección por D. **Luis Marcelo Chinea Rangel**.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 11 de julio de 2024

Agradecimientos

Me gustaría agradecer a mi tutora, Pino Caballero Gil, y a mi cotutor, Óscar Cigala Álvarez, por concederme su orientación, tiempo y ayuda durante todo el desarrollo de este proyecto.

Y lo más importante a mi familia, que han supuesto mi pilar y motivación a lo largo de mi trayecto universitario, además de a todos los amigos que me han acompañado durante este viaje.

Licencia

Resumen

La Criptografía de Curva Elíptica es una forma de criptografía de clave pública basada en las propiedades de las curvas elípticas en un plano bidimensional, formadas por los puntos en un cuerpo finito que cumplen la ecuación algebraica $y^2 = x^3 + ax + b$. Dichos criptosistemas se utilizan actualmente en una gran diversidad de sectores como telecomunicaciones, finanzas, tecnologías blockchain, autenticación y firma electrónica, por lo que suponen un elemento esencial de la seguridad de muchas tecnologías actuales.

El objetivo de este Trabajo de Fin de Grado es el desarrollo de un entorno web donde los usuarios, principalmente estudiantes o investigadores que quieran iniciarse en el campo de la criptografía de clave pública, dispongan de herramientas eficaces para comprender conceptos de la criptografía de curva elíptica y realizar los cálculos necesarios de manera rápida. Adicionalmente, se ha realizado la implementación de una serie de ataques con el objetivo de hacer la herramienta más interactiva y detectar vulnerabilidades en curvas.

Palabras clave: Criptografía, Ciberseguridad, ECC, ECDH, ECEG, SETUP, Pohlig-Hellman, Baby-Step Giant-Step, Ataques Criptográficos, Cleptografía

Abstract

Elliptic Curve Cryptography is a form of public-key cryptography based on the properties of elliptic curves in a two-dimensional plane, defined by the points over a finite field that satisfy the algebraic equation $y^2 = x^3 + ax + b$. These cryptosystems are currently used in a wide range of sectors, such as telecommunications, finance, blockchain technologies, authentication, and electronic signatures, making them an essential element of the security of many current technologies.

The objective of this Bachelor's Thesis is the development of a web environment where users, mainly students or researchers who want to get started in the field of public-key cryptography, have effective tools to understand concepts of elliptic curve cryptography and perform necessary calculations quickly. Additionally, a series of attacks have been implemented in order to make the tool more interactive and detect vulnerabilities in curves.

Keywords: Cryptography, Cybersecurity, ECC, ECDH, ECEG, SETUP, Pohlig-Hellman, Baby Step Giant Step, Cryptographic Attacks, Kleptography

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción y antecedentes

Este capítulo recoge una introducción al trabajo, ya que incluye un repaso histórico de la criptografía, así como los objetivos, justificación y estructura de este trabajo.

1.1. Criptografía

1.1.1. Historia

La criptografía nació como una necesidad innata de la humanidad de asegurar la transmisión de información de manera confidencial entre distintas partes involucradas. Desde tiempos antiguos las personas se han estado enviando mensajes secretos ya sea de carácter personal, militar, político o comercial.

En la cultura popular se suele pensar que esta disciplina surgió tras la aparición y auge de los sistemas informáticos. Sin embargo, la criptografía existe desde mucho antes que la informática. Por ejemplo, si nos remontamos al 27 a.C., el imperio romano ya utilizaba sistemas criptográficos sencillos para la transmisión de órdenes del emperador Cayo Julio César entre las distintas regiones del imperio. De hecho, el primer ordenador en la historia, al menos en sentido práctico, la máquina *Bombe*, desarrollada a partir del trabajo del equipo del matemático británico Alan Turing en 1939 durante la Segunda Guerra Mundial [1], surgió como solución a un problema criptográfico: romper el cifrado de los mensajes transmitidos entre el ejército nazi. El resultado permitió a los aliados ganar la guerra.

Durante los dos últimos siglos, la criptografía ha sufrido un crecimiento exponencial, en parte gracias al simultáneo crecimiento de los sistemas informáticos. Hoy en día, la criptografía es un componente esencial en la seguridad de la información, ya que permite proteger datos sensibles en multitud de aplicaciones, desde transacciones bancarias en línea hasta

comunicaciones de carácter personal y corporativo.

Los avances en la criptografía moderna han llevado a la creación de numerosos algoritmos complejos. Sin ir más lejos, el cifrado estándar del gobierno de los Estados Unidos, *Rijndael*, también conocido como AES (*Advanced Encryption Standard*), fue escogido en 2001 [2]. Entre esos avances destaca el nacimiento en 1976 de la criptografía de clave pública [3], de la cual es máximo exponente el algoritmo *RSA* [4].

La criptografía de clave pública permite la comunicación entre entidades sin necesidad de contacto previo, ya que se basa en la utilización de claves públicas para el cifrado. Concretamente, cada participante involucrado en la comunicación cuenta con una clave privada que usa para descifrar, y una clave pública que usan los demás para cifrar mensajes dirigidos a ese participante. Nótese que la distribución públicas de claves de cifrado no pone en peligro la confidencialidad de la información cifrada puesto que mientras la clave pública se utiliza para el cifrado de mensajes, la correspondiente clave privada es la única que permite descifrar esos mensajes.

1.1.2. Criptografía de curva elíptica

El tema central de este Trabajo de Fin de Grado, los sistemas criptográficos basados en la Criptografía de Curva Elíptica (ECC, *Elliptic Curve Cryptography*) [5], son cifrados de clave pública. Dichos sistemas, cuyas propiedades serán estudiadas más adelante, se basan en la utilización de curvas elípticas sobre cuerpos finitos para aplicar una versión del algoritmo de intercambio de claves de Diffie-Hellman.

La ECC ha ganado mucha popularidad en los últimos años debido a su capacidad de proporcionar niveles de seguridad comparables o hasta más eficientes que otros algoritmos contemporáneos, como el mencionado RSA, haciendo uso de claves más pequeñas. Esta característica tiene un impacto notorio en la eficiencia y rendimiento de dichos sistemas, incrementando la viabilidad de implementación en dispositivos con recursos limitados.

1.1.3. Aplicaciones modernas

La ECC está presente en la inmensa mayoría de tecnologías usadas actualmente, siendo las siguientes algunas de las aplicaciones más destacables:

- Comunicaciones seguras: Protocolo HTTPS para la navegación segura por internet [7] .
- Firmas digitales: Autenticación de documentos y transacciones [8].
- *Blockchain* y criptomonedas: Asegurar la integridad y privacidad de las transacciones [9].
- Protección de datos personales: Cifrado de datos en dispositivos móviles y servicios en la nube.
- Seguridad nacional: Protección de comunicaciones y datos sensibles en entornos militares y gubernamentales.

Cabe recalcar, dadas sus aplicaciones, que su aplicación no radica únicamente en la protección de la confidencialidad de la información, sino que además tiene otras muchas funciones como la protección de la autenticidad e integridad de los datos, por lo que supone un pilar fundamental en el entorno digital tal y como lo conocemos hoy en día.

1.2. Objetivos

Se ha planteado la siguiente serie de objetivos a realizar en este trabajo en relación a la ECC:

- Realizar la implementación del algoritmo de Diffie-Hellman de Curva Elíptica (ECDH, Elliptic Curve Diffie-Hellman) [10] y el Cifrado de ElGamal Elíptico (ECEG, Elliptic Curve ElGamal) [11], siendo esta la base del resto de objetivos.
- Realizar el diseño y desarrollo de una herramienta web enfocada en el estudio de la ECC en entornos seguros, poniendo especial atención en los ámbitos de investigación y didáctica. El objetivo es que dicha herramienta sea clara, intuitiva y lo más accesible posible.
- Implementar en la herramienta tres ataques a sistemas criptográficos basados en ECC, con el objetivo de analizar su seguridad.

- Realizar un estudio de las vulnerabilidades encontradas y hacer una valoración de las curvas seleccionadas, extrayendo algunas conclusiones.

1.3. Justificación y relevancia en la actualidad

El desarrollo emergente de la computación cuántica presenta un considerable desafío para los sistemas criptográficos actuales, destacando en relación a este trabajo la ECC, debido a la inevitable resolución con ordenadores cuánticos de los problemas matemáticos en los cuales se basan estos sistemas.

Por ese motivo, la criptografía post-cuántica, basada en computación actual y con el objeto de que sea resistente a la computación cuántica, se ha convertido en un campo de investigación muy activo. Dicha criptografía está respaldada por el Instituto Nacional de Estándares y Tecnología (NIST, National Institute of Standars and Technology), con el objetivo de fomentar el desarrollo de algoritmos resistentes a ataques con ordenadores cuánticos, garantizando así la seguridad en los sistemas informáticos en la nueva era cuántica. De esa manera, el descubrimiento de vulnerabilidades cuánticas en la ECC provoca inevitablemente el desarrollo de nuevos métodos criptográficos y la necesidad de migración o complementación de los sistemas criptográficos actuales.

1.4. Estructura del documento

Este trabajo se estructura de la siguiente manera:

- Capítulo 2. Introducción a la ECC.
- Capítulo 3. Introducción a algunos ataques a ECC.
- Capítulo 4. Descripción de la metodología de trabajo aplicada en este trabajo y de las herramientas y tecnologías utilizadas.
- Capítulo 5. Descripción de la implementación llevada a cabo, distinguiendo las distintas fases etapas del desarrollo de la herramienta web, destacando algunas de las principales dificultades encontradas.
- Capítulo 7. Presentación de conclusiones y discusión de resultados obtenidos y de posibles mejoras de la herramienta web implementada.

Capítulo 2

Preliminares

Antes de entrar en detalle en la descripción del desarrollo, es conveniente explicar los sistemas criptográficos en los que se basa este Trabajo de Fin de Grado, así como los principios fundamentales de la aritmética de puntos, intercambio de claves, codificación y cifrado de mensajes.

2.1. Criptografía de Curva Elíptica

La ECC es un tipo de criptografía asimétrica que se basa en las propiedades matemáticas de las curvas elípticas definidas sobre cuerpos finitos. A diferencia de la criptografía simétrica, que utiliza la misma clave para cifrar y descifrar los mensajes, la criptografía asimétrica emplea un par de claves: una pública y una privada.

Una curva elíptica es una curva algebraica definida sobre un cuerpo finito F_p, cuyos puntos satisfacen la ecuación de Weierstrass: $y^2 = x^3 + ax + b$, donde a y b son coeficientes pertenecientes al cuerpo F_p. La ecuación de la curva elíptica no debe tener singularidades, lo que se garantiza siempre que los coeficientes a y b satisfagan la condición: $4a^3+27b^2 \neq 0$ en F_p. En este trabajo se supone que p es un número primo.

La curva elíptica no solo es el conjunto de los puntos (x, y) que satisfacen la ecuación de Weierstrass, sino que también incluye el "punto en el infinito", denotado usualmente como O, cumpliendo la función de identidad del grupo que forma la curva con la operación suma que se define a continuación.

2.2. Aritmética de puntos

Los sistemas criptográficos basados en ECC funcionan realizando operaciones aritméticas, principalmente la adición, duplicación, negación y multiplicación de los puntos de la curva. Dichas operaciones se rigen por las siguientes reglas:

- Dado dos puntos P y Q de la curva E, la suma de los dos puntos implica la generación de otro punto R perteneciente a la misma curva.

$$R = P + Q \quad \text{donde} \quad R, P, Q \in E$$

- La duplicación de un punto P implica la suma de ese punto consigo mismo:

$$2P = P + P$$

- La negación de un punto $P = (x, y)$, implica la negación de su coordenada y:

$$-P = (x, -y)$$

- La multiplicación de un punto P por un escalar k implica sumar el punto P consigo mismo repetidamente k veces:

$$kP = P + P + P + ... + P \ (k \ veces)$$

Para la operación de adición se pueden distinguir dos casos dependiendo de si los puntos son iguales o distintos, dichos casos conducen a fórmulas distintas:

- Si P es igual a Q (duplicación de puntos):
  - $\lambda = \frac{3x_P^2 + a}{2y_P}$
  - $x_R = \lambda^2 - 2x_P$
  - $y_R = \lambda(x_P - x_R) - y_P$
- Dados dos puntos P y Q distintos entre sí:
  - $\lambda = \frac{y_Q - y_P}{x_Q - x_P}$
  - $x_R = \lambda^2 - x_P - x_Q$
  - $y_R = \lambda(x_P - x_R) - y_P$

Todas las mencionadas operaciones de puntos se realizan con aritmética modular en base al cuerpo finito F_p.

2.3. Diffie-Hellman elíptico

El algoritmo Diffie-Hellman de Curva Elíptica (ECDH, Elliptic Curve Diffie-Hellman) es un protocolo de intercambio de claves que permite a dos partes generar una clave secreta compartida utilizando sus claves privadas y públicas de curvas elípticas. El protocolo consta de los siguientes pasos:

1. Cada parte A y B genera su par de claves pública y privada usando una curva elíptica y un punto base G de esa curva como elementos comunes y públicos.
2. Ambas partes intercambian sus claves públicas.
3. Cada parte combina su clave privada con la clave pública de la otra parte para calcular la clave secreta compartida (ver Figura 2.1).

Esta clave compartida puede usarse luego para cifrar y descifrar mensajes, garantizando una comunicación segura.

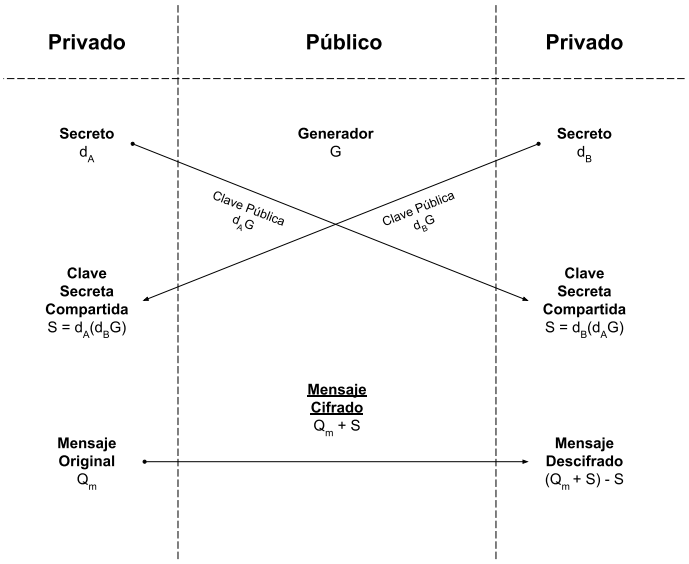

Figura 2.1: Intercambio de claves ECDH y cifrado ECEG

2.4. Cifrado de ElGamal elíptico

El cifrado de ElGamal Elíptico (ECEG, Elliptic-Curve ElGamal) es un esquema de cifrado que extiende al cifrado ElGamal tradicional de la forma siguiente:

1. Clave pública y privada: El receptor B genera su par de claves pública y privada usando ECDH.

2. La emisora A utiliza la clave pública del receptor B para cifrar el mensaje original m codificado como punto Q_m, generando un punto como mensaje cifrado (ver Figura 2.1):

$$Q_m + d_A(d_B G)$$

3. Descifrado: El receptor B utiliza su clave privada d_B para descifrar el mensaje cifrado, recuperando el mensaje original:

$$Q_m = ((Q_m + d_A(d_B G)) - d_B(d_A G)$$

Las operaciones como el intercambio de claves ECDH y la firma de mensajes mediante ECDSA (*Elliptic Curve Digital Signature Algorithm*) deben su seguridad al problema del logaritmo discreto elíptico, un problema matemático que se puede formular de la siguiente forma: *"Dados dos puntos P y Q pertenecientes a una curva elíptica E, hallar el entero k tal que $kP = Q$"*. Se cree que es más difícil resolver el problema del logaritmo discreto elíptico que resolver la versión básica del problema del logaritmo discreto en cuerpos finitos.

Capítulo 3

Ataques implementados

En este capítulo se detallan el funcionamiento y propiedades de los algoritmos de ataques implementados en la aplicación web desarrollada. Para este trabajo se han escogido tres ataques criptográficos: SETUP [12], Pohlig-Hellman [13] y Baby-Step Giant-Step [14]. Así, este capítulo servirá como un recurso valioso para entender la teoría detrás de los ataques criptográficos, lo que permitirá su aplicación práctica en un contexto real mediante una herramienta accesible y funcional.

3.1. SETUP

El ataque SETUP, llamado así por las siglas de Secretly Embedded Trapdoor with Universal Protection, implica la implementación de un mecanismo oculto dentro de un sistema criptográfico de caja negra. Está diseñado para aprovechar las propiedades matemáticas de las curvas elípticas y sus operaciones criptográficas, especialmente el intercambio de claves, para filtrar información confidencial de manera encubierta.

A simple vista el sistema realiza las operaciones criptográficas de manera convencional, utilizando el punto base G de la curva, para generar las claves públicas de los usuarios, dG, a partir de las claves privadas d de los usuarios. Sin embargo, el sistema incluye una serie de operaciones adicionales para generar un valor adicional Z utilizando la clave pública del atacante VG, con el objetivo de calcular mediante una función *hash*, la clave privada del receptor, $c2$. Una vez generados esos valores, lo único que devuelve, y que debe ser interceptado por el atacante, son los valores de las claves públicas de los usuarios, $M1$ y $M2$, que el atacante podrá utilizar para recuperar la clave privada $c2$.

El procedimiento completo está compuesto de tres algoritmos [15]:

- Algoritmo 1:
  - Durante la primera ejecución, el dispositivo genera de manera aleatoria el secreto $c1$ y lo almacena en una parte de la memoria no volátil. Dicho secreto se debe de generar acorde al orden de la curva.

$$2 \leq c_1 \leq n - 1$$

  - El dispositivo devuelve como salida la clave pública $M_1 = c_1 G$.
- Algoritmo 2:

- Después del primer uso, el dispositivo cambia su modo de operación y genera los enteros fijos y aleatorios a, b, h, e, además de j, u, siendo estos aleatoriamente 0 o 1:

$$a, b, h, e <= n - 1$$

$$j, u \in 0, 1$$

- El dispositivo calcula el punto Z usando los valores previamente generados, el generador de la curva G y la clave pública del atacante V mediante la fórmula:

$$Z = a \cdot c_1 \cdot G + b \cdot c_1 \cdot V + h \cdot j \cdot G + e \cdot u \cdot V$$

- Se genera la clave privada c_2, mediante una función *hash* criptográficamente segura. Realizar un *hashing* de un punto de una curva elíptica implica utilizar la función *hash* sobre la coordenada x del punto:

$$c_2 = H(Z)$$

- El dispositivo devuelve como salida la clave pública $M_2 = c_2 G$.

- Algoritmo 3:

  - El atacante habiendo tenido acceso a M_1 y M_2, recupera el secreto c_2 realizando las siguientes operaciones:

$$Z_1 = a \cdot M_1 + b \cdot v \cdot M_1 = a \cdot M_1 + b \cdot v \cdot c_1 \cdot G = a \cdot c_1 \cdot G + b \cdot c_1 \cdot V$$

  - Para cada valor posible de j, u:

$$Z_2 = Z_1 + h \cdot j + \cdot G + e \cdot u \cdot V$$

$$c_2 = H(Z_2)$$

  - Si $c_2 G = M_2$, se devuelve como salida c_2.

Así, el atacante puede recuperar mediante un canal oculto en un dispositivo presuntamente confiable para el resto de usuarios las claves privadas generadas, sin levantar ningún tipo de sospecha.

3.2. Pohlig-Hellman

El algoritmo de Pohlig-Hellman es un algoritmo recursivo eficiente que resuelve el problema del logaritmo discreto elíptico dividiéndolo en subproblemas, que se resuelven calculando los logaritmos discretos en los subgrupos de orden primo de un punto P correspondientes a cada subproblema.

A continuación se detallan las tres fases de un ataque usando Pohlig-Hellman:

1. Se factoriza el orden de grupo n en sus factores primos:

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_r^{e_r}$$

2. Una vez que se tiene la factorización del orden del grupo, se procede a descomponer el problema del logaritmo discreto en subproblemas. Para cada factor primo $p_i^{e_i}$, se calcula $l_i = lmodp_i^{e_i}$. Esto implica resolver el logaritmo discreto en el subgrupo de orden $p_i^{e_i}$. La descomposición facilita la solución del problema, ya que los subproblemas resultantes son más sencillos de manejar que el problema original.

3. Una vez resueltos los subproblemas y dado que dichos problemas fueron formados usando los factores primos del orden n, se puede aplicar el Teorema Chino del Resto (CRT, *Chinese Remainder Theorem*) [16] a los subgrupos: $gcd(pi, pj) = 1$ para todo i, j. Así, se halla una solución única a la serie de congruencias obtenidas:

$$l \equiv l_1(modp_1^{e_1})$$

$$l \equiv l_2(modp_2^{e_2})$$

$$\dots$$

$$l \equiv l_r(modp_r^{e_r})$$

Es importante destacar que la eficiencia del algoritmo de Pohlig-Hellman depende en gran medida del tamaño de los factores primos en la factorización del orden del grupo. Si los factores primos son grandes, el tiempo de cómputo requerido para resolver los subproblemas aumenta significativamente, haciendo que el algoritmo sea menos viable en la práctica. Por lo tanto, el algoritmo es más adecuado para situaciones donde los factores primos son relativamente pequeños.

Además, la factorización del orden del grupo en sí puede ser un problema complejo, especialmente para grupos con un orden grande. En tales casos, encontrar la factorización puede requerir una cantidad considerable de tiempo y recursos computacionales.

3.3. Baby-Step Giant-Step

El algoritmo *Baby-Step Giant-Step* busca resolver el problema del algoritmo discreto dividendo el problema en dos fases, con el objetivo de ahorrar tiempo de cómputo con los resultados intermedios, en comparación a una búsqueda por fuerza bruta. La idea es que dado el generador de una curva elíptica, G y un punto A, hallar el escalar k tal que $A = k \cdot G$ mediante la colisión de valores entre las dos fases realizadas.

La primera fase, conocida como *Baby-Steps* o pasos pequeños, crea un conjunto de puntos, resultantes de multiplicar el generador G por incrementos sucesivos determinados por la variable m, un escalar entero aproximado al valor de la raíz cuadrada del orden n.

$$m \approx \sqrt{n}$$

$$i = 1, 2, \dots m - 1 \rightarrow BabySteps_i = i \cdot G$$

La segunda fase, conocida como *Giant-Steps* o pasos grandes, crea otro conjunto de puntos, pero esta vez resultados de restar al punto A a jmG de manera iterativa.

$$j = 1, 2, \dots m - 1 \rightarrow GiantSteps_j = A - j \cdot m \cdot G$$

Una vez generados los conjuntos de *Baby-Steps* y *Giant-Steps*, el algoritmo busca una colisión entre ellos, es decir, encuentra un índice i y j tal que:

$$i \cdot G = A - j \cdot m \cdot G$$

Si se encuentra una colisión, el escalar k puede ser determinado como:

$$k = i + j \cdot m$$

Este método aprovecha el hecho de que la búsqueda de colisiones entre los conjuntos de *Baby-Steps* y *Giant-Steps* es más eficiente que explorar todos los posibles valores de k de forma secuencial.

Similar al algoritmo de Pohlig-Hellman, el éxito y la eficiencia del algoritmo *Baby-Step Giant-Step* dependen del tamaño del orden n de la curva elíptica. A medida que el orden aumenta, el tamaño de los conjuntos *baby-steps* y *giant-steps* crece, lo que incrementa el tiempo necesario para generarlos. Por lo tanto, este algoritmo es particularmente efectivo en curvas elípticas con órdenes no excesivamente grandes, donde la raíz cuadrada de n, m, sigue siendo manejable en términos de tiempo de computación.

Capítulo 4

Metodología de trabajo

En este capítulo se presentarán las distintas tecnologías utilizadas en la metodología de trabajo empleada durante el desarrollo de la herramienta web, a la que se ha apodado *Pandora* en referencia al mito de la caja de Pandora [17], donde todos los males y miserias del mundo se encuentran encerrados en una caja, según la mitología griega. La metodología de trabajo durante el desarrollo de Pandora ha seguido el modelo SCRUM debido a la eficiencia y adaptabilidad a la hora de gestionar el trabajo, pues SCRUM, como marco de trabajo ágil, permite dividir el desarrollo de proyectos en *sprints* o ciclos cortos de trabajo, lo que promueve el desarrollo iterativo y margen rápidos de adaptación a los cambios y necesidades emergentes en el proyecto. Al comienzo de cada *sprint*, en este caso con una duración aproximada de 1-2 semanas, se ha realizado una sesión de planificación para definir los objetivos y tareas a lograr. Las revisiones diarias de los avances y problemas encontrados fomentaron una rápida implementación y su respectiva depuración en las etapas finales del sprint. Los usuarios pueden acceder a la aplicación desde en el enlace [18].

4.1. Control de versiones

Durante la realización del trabajo se ha hecho uso de Git como sistema de control de versiones para una correcta organización y registro de las distintas fases de desarrollo del cliente y servidor de Pandora. Todos los cambios y documentación relacionadas con la aplicación se encuentran disponibles en el repositorio de GitHub, desde el cual está disponible la descarga de la herramienta [19].

4.2. Control de calidad del código

Debido a la escalabilidad del código se optó por hacer uso de SonarCloud [20], una herramienta basada en la nube que permite obtener un informe de calidad de código y su evolución de manera continua con cada *commit* realizado. Gracias a ello, la identificación temprana de errores, vulnerabilidades de seguridad y *bugs* han estado presente en todo momento, junto a buenas prácticas de programación recomendadas por la misma herramienta (ver Figura 4.1).

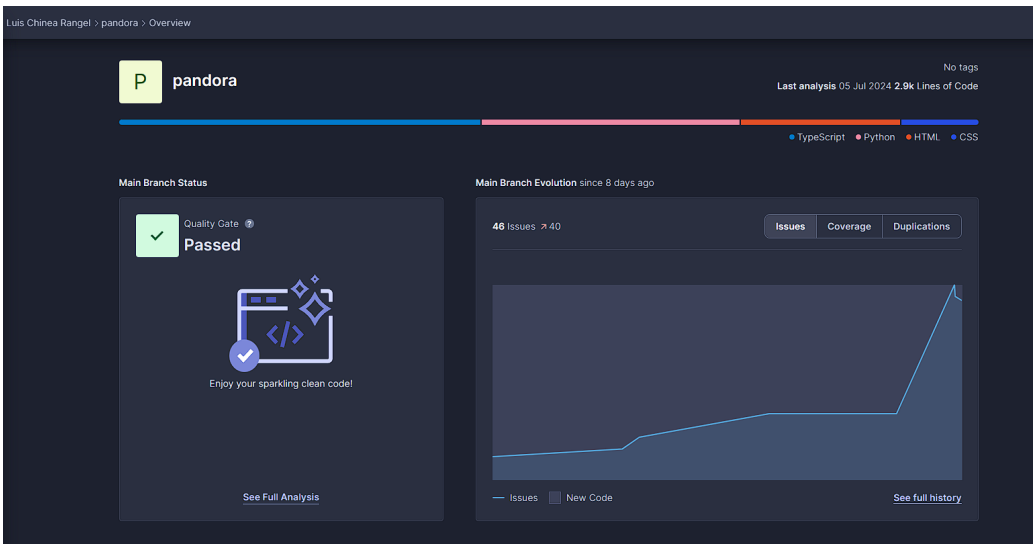

Figura 4.1: *Dashboard* de la herramienta Sonarcloud

4.3. Despliegue continuo

En primera instancia se concibió a Pandora como aplicación web de uso local y exclusivamente personal a cualquier entidad que tuviese acceso al código fuente desde el repositorio de GitHub. Sin embargo, dada la naturaleza didáctica y académica del proyecto, durante las últimas fases de desarrollo se priorizó la búsqueda de opciones de despliegue continuo para que cualquier persona pudiese acceder a la aplicación desde cualquier parte y sin necesidad de descargar el código fuente.

Esta búsqueda influyó de manera positiva en el desarrollo no sin tener que hacer severas revisiones e inclusiones de *frameworks* enfocados en el desarrollo web como Angular [21] con el objetivo de conseguir una arquitectura de doble capa cliente-servidor.

Las herramientas seleccionadas para cumplir esta tarea fueron :

- Render: plataforma basada en la nube con multitud de servicios como alojamiento web, bases de datos PostgreSQL, despliegue de aplicaciones web con dominio gratuito incluido, *background workers* y configuración de variables de entorno [22] (ver Figura 4.2).
- Netlify: plataforma enfocada en el despliegue de la capa de presentación o *Frontend* de aplicaciones web, con despliegues continuos con cada cada *commit* y *pull request* del repositorio, dominio gratuito y herramientas de soporte para *frameworks* como Angular [23].

4.4. Lenguajes de programación

4.4.1. Python

Python [24] es un lenguaje de programación de alto nivel, de código abierto, con una amplia colección de librerías y paquetes disponibles al público, lo que permite su

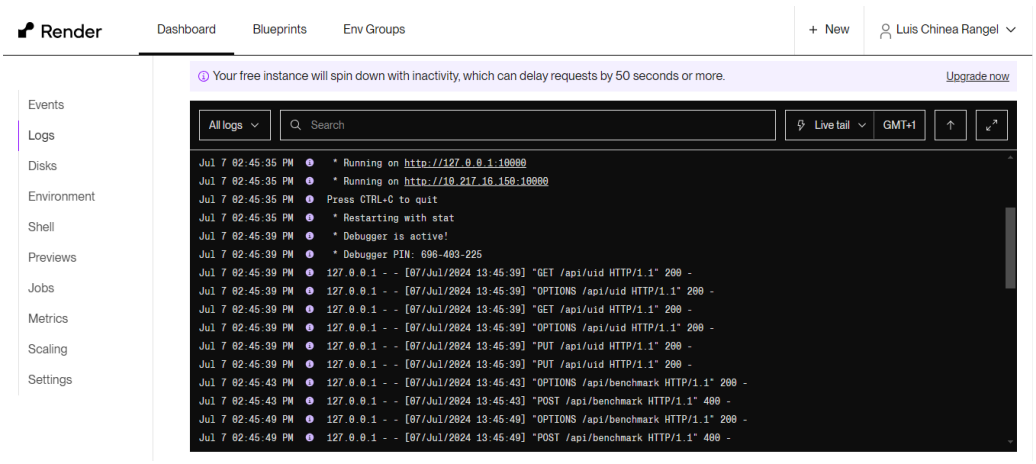

Figura 4.2: Historial de Pandora en Render

presencia en el desarrollo de aplicaciones de cualquier área. Además, Python es un lenguaje interpretado, haciéndolo dinámico, flexible e independiente de la plataforma donde se ejecute el código.

Algunas de las áreas son el desarrollo de aplicaciones web, el *Data Science* con el uso de librerías como Numpy[25] y Seaborn[26], además de la inteligencia artificial, como es el caso de Tensorflow [27] y Pytorch [28]. Python también se utiliza extensamente para automatización y *scripting*, permitiendo la escritura de scripts para tareas repetitivas como la manipulación de archivos, el procesamiento de datos y la administración de sistemas.

4.4.2. Typescript

Typescript [29] es un lenguaje de programación orientado a objetos, desarrollado por Microsoft y basado en JavaScript [30] que añade características inexistentes a su base como:

- Tipado fuerte, es decir la prevención de violaciones de tipos de datos.
- Detección de errores en la fase de desarrollo en vez de en tiempo de ejecución como en JavaScript.
- Soporte con multitud de *frameworks* para el desarrollo web.

En fases tempranas del proyecto se optó por utilizar Javascript, pero dadas a las limitaciones del lenguaje y a los crecientes requisitos de la aplicación se tomó la decisión de cambiar el lenguaje a Typescript, entre otros aspectos debido a su amplia compatibilidad con *frameworks* y herramientas, siendo el más destacado Angular.

Esta toma de decisiones permitió una mejora notable en la interfaz de usuario de manera rápida, además de reducir la carga de trabajo del servidor Flask [31], que hasta ahora se encargaba de servir la interfaz de usuario. Otra de las ventajas de este cambio fue la detección temprana de errores y mejora en la calidad del código, no obstante la migración de un lenguaje a otro llevo un período de tiempo y trabajo no previsto en las fases tempranas del proyecto.

4.5. Tecnologías Empleadas

4.5.1. Flask

En el desarrollo de aplicaciones web, Python es ampliamente utilizado gracias a *frameworks* como Django [32] y Flask. En el caso de este proyecto profundizaremos en Flask, que es un *micro-framework* ligero y flexible ideal para el desarrollo de aplicaciones web y APIs.

Este *framework* se encarga de desplegar el servidor de Pandora, de gestionar, procesar y validar las peticiones HTTP realizadas por los usuarios y lo más importante, de realizar las operaciones criptográficas como generación de claves, cifrado y descifrado de mensajes y ataques a curvas elípticas.

4.5.2. Node.js

Node.js [33] es un entorno de ejecución de JavaScript construido sobre el motor V8 de Google Chrome y diseñado para la creación de sitios web dinámicos y la ejecución de código en el lado del servidor. Para el desarrollo de Pandora se ha hecho uso de este entorno como base para la instalación de los *frameworks* relativos al Frontend.

La elección de Node.js como entorno de ejecución se debe a varias razones clave: su velocidad y eficiencia; su extenso conjunto de paquetes y módulos disponibles, que facilita el desarrollo y la integración de diversas funcionalidades; y por su arquitectura asíncrona, basada en eventos y no bloqueante, que permite manejar múltiples conexiones concurrentes con alto rendimiento.

Angular

Angular es un *framework* de código abierto, desarrollado en Typescript y soportado por Google que consiste en un amplio conjunto de librerías y herramientas dedicadas al *Frontend* de aplicaciones web. Su principal atractivo es la creación rápida de vistas de interfaz de usuario a través de la creación de componentes independientes encargados de funciones específicas en la herramienta web.

Angular Material

Angular Material [34] es un módulo de Angular enfocado en la implementación de componentes con diseños basados en Material Design [35], el lenguaje de diseño oficial de Google. La incorporación de este módulo ha sido clave para mejorar el diseño y usabilidad de Pandora, añadiendo diseños minimalistas y limpios.

Las ventajas de utilizar Angular Material incluyen la consistencia en el diseño, ya que al seguir las directrices de Material Design garantiza una apariencia y comportamiento consistentes en toda la herramienta; los componentes predefinidos, que proporcionan una amplia gama de elementos como botones, tarjetas, barras de herramientas y formularios, que pueden ser fácilmente personalizados y extendidos; la accesibilidad, debido a que los componentes están diseñados teniendo en cuenta las mejores prácticas de accesibilidad, facilitando la creación de aplicaciones inclusivas; y el *responsive design*, que asegura una correcta visualización y funcionamiento en diferentes dispositivos y tamaños de pantalla.

Chart.js

Para la realización de elementos interactivos, como la visualización de una curva elíptica o los puntos sobre el cuerpo finito primo, se requería de hacer uso de gráficos avanzados y que se generasen de manera dinámica en función a los cambios realizados por los usuarios finales en los parámetros. Para cumplir con este objetivo, se optó por utilizar Chart.js [36].

Chart.js es una biblioteca JavaScript de código abierto diseñada para facilitar la creación de gráficos interactivos y visualizaciones de datos en aplicaciones web. Esta herramienta es especialmente útil debido a su flexibilidad, facilidad de uso y amplia variedad de tipos de gráficos a elegir.

Capítulo 5

Implementación

Este capítulo detallará la implementación de los ataques introducidos en capítulos anteriores dentro de la herramienta web creada. Así, este trabajo no solo explora la teoría detrás de los algoritmos discutidos, sino que también proporciona una visión profunda de cómo se llevan a cabo en la práctica pues se proporcionan a continuación varios ejemplos concretos y resultados de pruebas utilizando la herramienta desarrollada, demostrando su capacidad para resolver problemas de seguridad criptográfica en entornos prácticos.

5.1. Simulador de Criptografía de Curva Elíptica

Desarrollar un servidor básico que pudiese crear y gestionar instancias de curvas elípticas, sus respectivos puntos, los principios y esquemas fundamentales para el cifrado y descifrado de mensajes en sistemas criptográficos elípticos ha sido el primer objetivo de implementación de Pandora.

Para ello se crearon las clases $Point$ y $Curve$ (ver Apéndices A.1 y A.2 respectivamente), que definen la aritmética de puntos así como la creación y generación de los puntos pertenecientes de una curva elíptica sobre su cuerpo finito.

Realizadas las pruebas desde el servidor Flask, se requería de una interfaz de usuario que gestionase la introducción de datos. Se optó por crear un componente utilizando el *framework* Angular y sus módulos para la validación de formularios de manera reactiva; como añadido extra se instaló la biblioteca para Javascript Chart.js, enfocada en la generación de gráficos, para la visualización de las curvas elípticas. Este último detalle enriquece el trasfondo didáctico de Pandora al poder mostrar de manera gráfica a los usuarios las propiedades de las curvas configuradas (ver Figura 5.1). En caso de cualquier error de validación, como que se seleccione un número no primo como cuerpo finito, o que no se proporcionen todos los campos obligatorios, se mostrará un mensaje de error al usuario indicando como debe proceder.

Todo usuario que acceda al componente de simulación a través de la herramienta web desplegada será provisto de un identificador de usuario (UID, *User ID*) interno con el objetivo de alcanzar una correcta gestión de las instancias creadas por el servidor. Dicha característica fue uno de los cambios más importantes a lo largo del desarrollo debido a que expandió la usabilidad de la herramienta al no estar limitada a un único usuario.

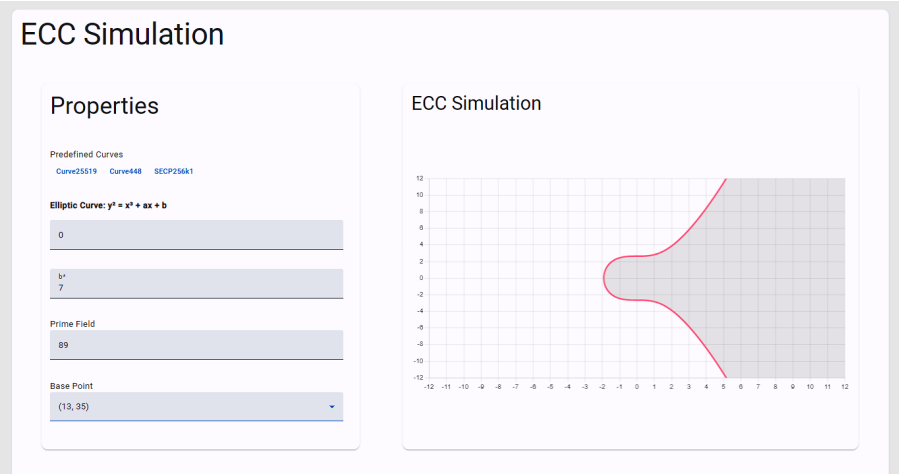

Figura 5.1: Simulación de la curva $y^2 = x^3 + 7$

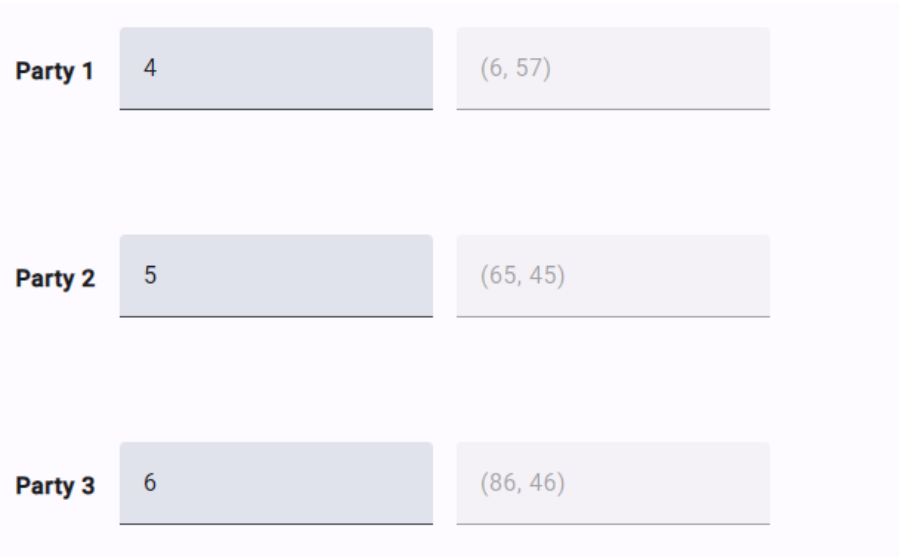

Figura 5.2: Generación de las claves públicas de las entidades

5.2. Intercambio de claves

Una vez lograda la generación de los puntos de la curva, el siguiente objetivo consiste en implementar el algoritmo de intercambio de claves ECDH (ver Figura 5.2). Para lograr este objetivo, se crearon las rutas en el servidor para aceptar peticiones HTTP de los usuarios, recibiendo el UID correspondiente y la clave secreta. Una vez recibido, el servidor responde devolviendo la clave pública, previamente habiendo validado los datos. En el caso de la clave secreta compartida, el servidor recibe el UID, la clave secreta de una de las entidades y la clave pública de otra.

En el Frontend se añadió al componente de simulación un formulario para elegir la cantidad de entidades involucradas en el intercambio de mensajes, con dicho valor se piden las claves privadas de cada uno para realizar la generación de claves públicas (ver Figura 5.3). Finalizada la generación de los pares de claves para cada entidad, y siempre y cuando cuyos valores sean válidos, se realiza de manera automática una serie de peticiones al servidor para generar la clave compartida de los integrantes.

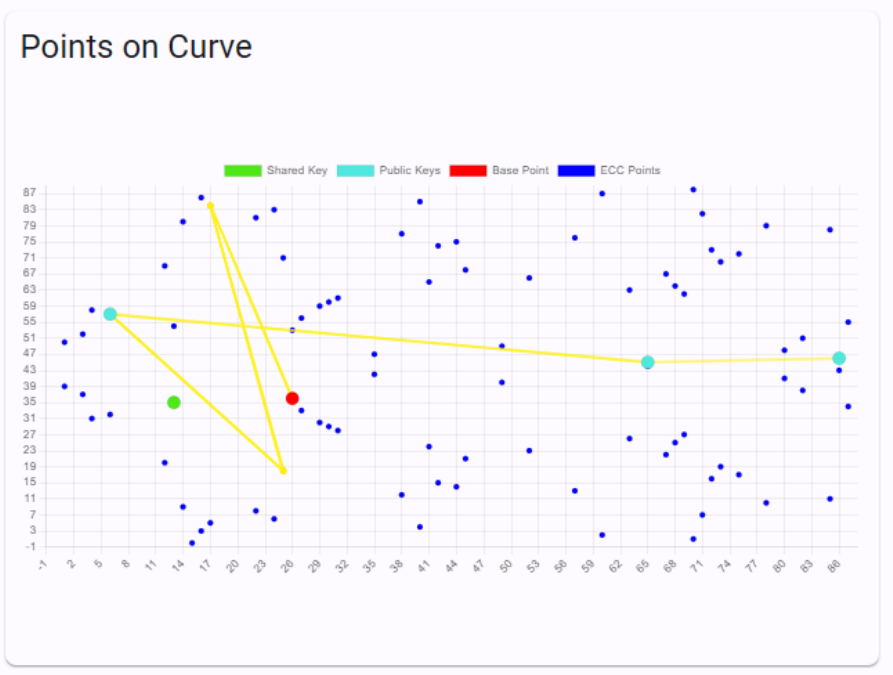

Figura 5.3: Cálculo de las claves públicas mediante ECDH

5.3. Cifrado y descifrado de mensajes

Una vez obtenidas las claves públicas el usuario puede proceder con el cifrado de mensajes. El primer paso para esta tarea es configurar el alfabeto a usar, cumpliendo la siguiente serie de requisitos:

- Cualquier carácter es válido en el alfabeto: letras, números, símbolos especiales...
- Se debe tener en cuenta el uso de mayúsculas y minúsculas en los alfabetos. Si se configura un alfabeto de exclusivamente caracteres en minúscula, cualquiera de sus miembros en mayúscula no pertenecen al alfabeto y viceversa.
- El cuerpo finito F_p de la curva debe ser mayor al doble a M, siendo este el tamaño total del alfabeto.

Para facilitar al usuario la tarea de introducir manualmente alfabetos con tendencias a ser comúnmente usados, la herramienta cuenta con una amplia colección de opciones predeterminadas, entre las que se incluyen la selección de los lenguajes más conocidos, sistemas numéricos como el hexadecimal, binario y octal, formatos de codificación como UTF-8 y opciones de transformado rápido de caracteres (ver Figura 5.4).

5.3.1. Cifrado

El proceso de cifrado es muy sencillo, el usuario selecciona el emisor desde la lista de opciones disponibles y el receptor del mensaje, introduce el texto a cifrar. En el caso de que el proceso de cifrado no se pueda realizar se le notifica al usuario en el mismo formulario (ver Figura 5.5).

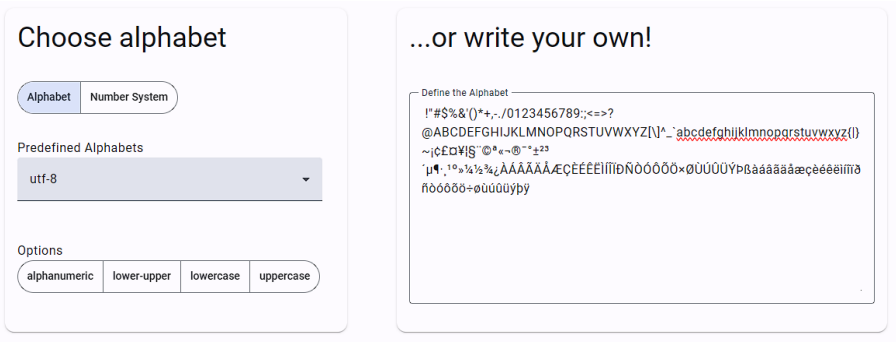

Figura 5.4: Configuración de un alfabeto para el cifrado de los mensajes

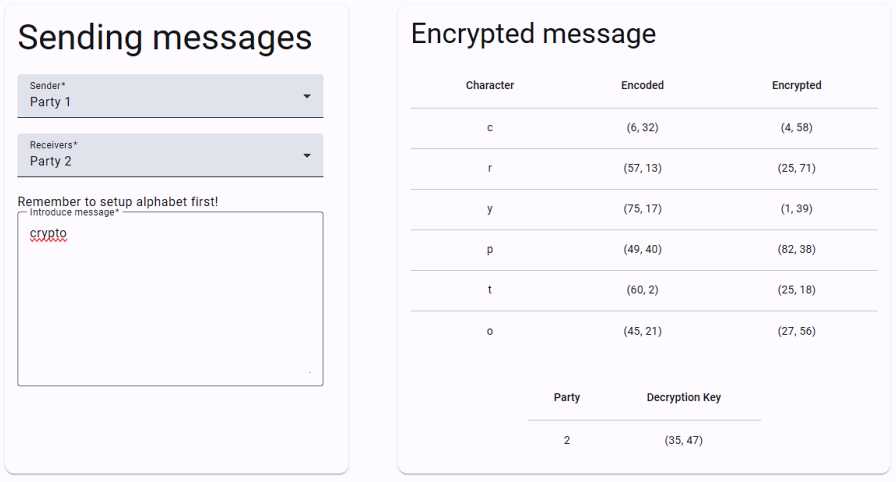

Figura 5.5: Cifrado de mensaje

En caso contrario, la tabla de resultados se actualiza dinámicamente para incluir los resultados del cifrado, Q_m, además de indicar en una tabla adicional la clave de descifrado d_AG, es decir la clave pública del emisor. De esta manera se puede enviar el mensaje y la clave pública por un canal no seguro al receptor sin riesgos de filtrado de información.

5.3.2. Descifrado

Para descifrar el mensaje, el receptor introduce su clave privada d_B, la clave pública del emisor d_AG, además de los puntos que componen el mensaje cifrado $Q_m + d_A(d_BG)$. El servidor primero resta al mensaje cifrado la clave secreta compartida entre emisor-receptor $d_B(d_AG)$, dando como resultado el mensaje original codificado como puntos de la curva Q_m, al que se descodifica y recupera el texto (ver Figura 5.6).

5.4. Ataques a ECC

Para medir la seguridad de un sistema criptográfico basado en ECC, se ha implementado una sección de banco de pruebas de ataques. En dicha sección, el usuario puede poner a prueba las curvas elípticas de su elección, además de curvas ya predefinidas en la herramienta como SECP256k1, frente a los ataques criptográficos implementados en Pandora: SETUP, Pohlig-Hellman y Baby-Step Giant-Step (ver Figura 5.7). La implementa-

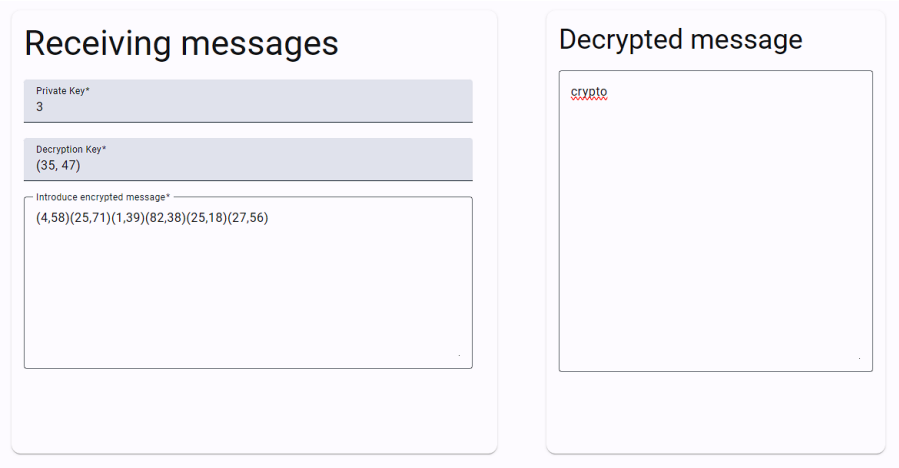

Figura 5.6: Descifrado de mensaje cifrado

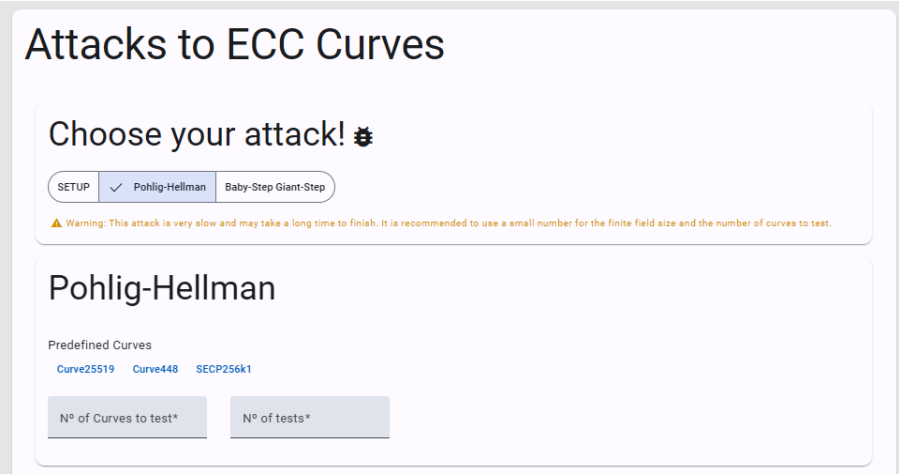

Figura 5.7: Selección de ataque criptográfico

ción de los ataques realizados en el Backend de Pandora se encuentran disponibles en el Apéndice B de este TFG.

Para el estudio se analizaron las siguientes curvas recomendadas por el NIST: SECP256k1, una curva elíptica especialmente utilizada en aplicaciones relacionadas con criptomonedas como Bitcoin y Ethereum; Curve448 y Curve25519, ambas curvas elípticas empleadas en protocolos de intercambio de claves y firma digital.

5.4.1. Pruebas realizadas con SETUP

Para el ataque SETUP se realizaron 10 pruebas a las curvas elípticas seleccionadas debido a su rapidez independientemente de los parámetros de la curva elíptica. Estos experimentos arrojaron resultados mixtos: se logró un éxito del 100 % en las curvas SECP256k1 y Curve25519, lo cual indica una vulnerabilidad significativa en la seguridad de estas curvas frente al ataque SETUP.

Sin embargo, la curva Curve448 no mostró ninguna vulnerabilidad durante el ataque, permaneciendo completamente segura (ver Tabla 5.1). Este contraste en los resultados resalta la importancia de seleccionar curvas elípticas con parámetros que no comprometan

la seguridad frente a métodos de ataque avanzados como el utilizado en el experimento SETUP.

Curve	a	b	Field	Successes	Rate (%)	Time (ms)
SECP256k1	0	7	$1{,}157920892373162 \times 10^{77}$	10	100	16.1263
Curve448	156326	1	$7{,}268387242956069 \times 10^{134}$	0	0	20.0589
Curve25519	486662	1	$5{,}78960446186581 \times 10^{76}$	10	100	14.6184

Tabla 5.1: Resultados de ataque SETUP

Dados los resultados de las pruebas realizadas, es evidente que hay motivos de preocupación ante la alta tasa de éxito del algoritmo SETUP, especialmente por su eficacia y el relativamente bajo tiempo de cómputo necesario para recuperar claves privadas. Es por este motivo y debido a la premisa subyacente en este algoritmo, que apunta a la vulnerabilidad de los usuarios que confían en sistemas de caja negra presumiblemente seguros, que es crucial ejercer precaución tanto con los sistemas ya implementados como en la elección de curvas elípticas recomendadas.

5.4.2. Pruebas realizadas con Pohlig-Hellman

La velocidad de cómputo del algoritmo de Pohlig-Hellman está directamente relacionado con el tamaño del cuerpo finito utilizado, por lo que realizar ataques a curvas elípticas con cuerpos finitos F_p elevados como SECP256k1, Curve25519 y Curve448 demanda una cantidad considerable de tiempo (llegando a ser inviable incluso en un único test a una sola curva) para su cálculo.

Es por ello que Pandora no ha logrado obtener resultados en tiempos razonables de los tests (ver Figura 5.8), no obstante cabe la posibilidad de éxito si se pudiesen acortar los tiempos de cómputo. Estos cuerpos finitos son escogidos precisamente por su robustez ante métodos de ataque como el algoritmo de Pohlig-Hellman, que se vuelve prohibitivamente lento conforme aumenta el tamaño del cuerpo finito y la complejidad del cálculo requerido.

No obstante, se puede probar el algoritmo con parámetros más pequeños para evaluar su desempeño en condiciones menos exigentes (ver Tabla 5.2).

a	b	Field	Successes	Rate (%)	Time (ms)
0	7	89	10	100	0.002997159957885742
2	2	17	10	100	0.0020017623901367188
1	1	23	10	100	0.003998517990112305

Tabla 5.2: Resultados de ataque Pohlig-Hellman

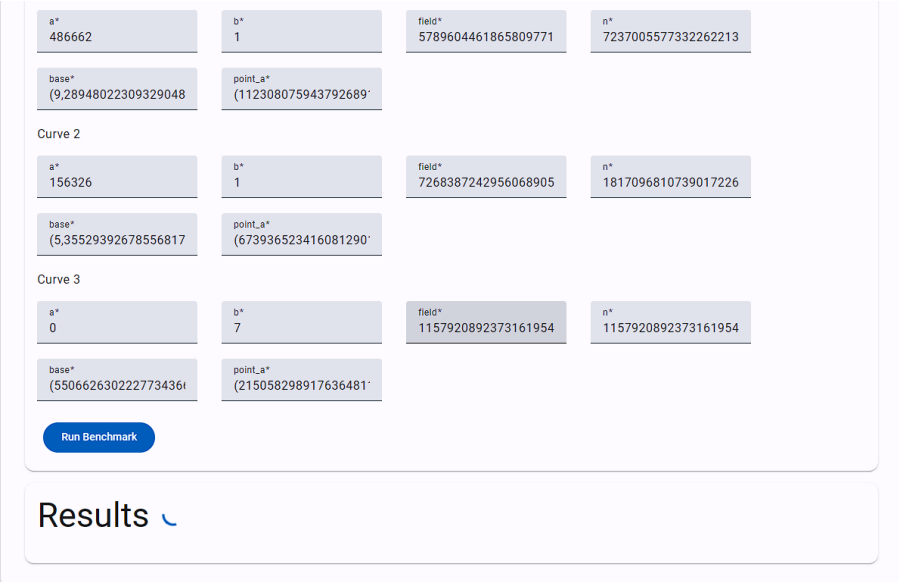

Figura 5.8: Tiempo de espera en Pohlig-Hellman

5.4.3. Pruebas realizadas con *Baby-Step Giant-Step*

En el caso del *Baby-Step Giant-Step* las pruebas dieron resultados positivos en pos a la seguridad de las curvas seleccionadas, debido a que la creación de conjuntos con una longitud m siendo esta $\sqrt{n}$ no es soportada directamente por su excesivo tamaño en memoria, por lo que los ataques fracasaron en estas curvas en concreto.

No obstante, se pueden realizar pruebas con curvas elípticas más pequeñas para probar el rendimiento del algoritmo (ver Tabla 5.3). Podemos observar que el tiempo de cómputo es mucho mayor comparado con las pruebas realizadas en curvas de similares parámetros en Pohlig-Hellman.

a	b	Field	Successes	Rate (%)	Time (ms)
0	7	89	10	100	0.06399917602539062
2	2	17	10	100	0.008007049560546875
1	1	23	10	100	0.025075435638427734

Tabla 5.3: Resultados de ataque Baby-Step Giant-Step

Capítulo 6

Conclusiones y líneas futuras

El objetivo principal de este trabajo ha sido la investigación de los algoritmos criptográficos que conforman la base de la ECC, además de los *frameworks* y herramientas necesarias para desarrollar una herramienta web de acceso público y libre, diseñada con un objetivo didáctico, siguiendo los principios de usabilidad y accesibilidad.

A lo largo de la fase de desarrollo se han encontrado varias dificultades relacionadas con la implementación y enlace de cliente-servidor de la herramienta, ya que como se ha mencionado, el planteamiento inicial era realizar una herramienta más bien de uso personal y local para cualquier usuario que accediese al repositorio. Sin embargo, debido a las limitaciones prácticas que este enfoque implicaba, se optó por realizar un despliegue completo, teniendo mejoras significativas en el diseño y flujo de trabajo, además de un incremento adicional del rendimiento del servidor al estar la herramienta desplegada en dos capas, cada una independiente y con sus propios recursos de *hardware*.

Merece destacar los resultados obtenidos en las pruebas de los ataques realizados, que si bien demuestran que la criptografía elíptica por el momento es notablemente práctica y segura, indican la existencia de diversas curvas criptográficas susceptibles a ataques, siendo un caso muy notable la de la curva SECP256K1, debido al impacto mundial al ser la base de infinidad de tecnologías *blockchain* y operaciones con criptomonedas.

Se ha tratado de optimizar el código lo máximo posible, pero es cierto que, especialmente para los ataques de Pohlig-Hellman y Baby-Step Giant-Step, pueden requerir un tiempo de cómputo muy alto en relación a los valores introducidos. Estos algoritmos, por su naturaleza, demandan una gran cantidad de recursos computacionales, lo que puede resultar en tiempos de ejecución prolongados. Para abordar este problema, es indispensable mejorar el rendimiento de los algoritmos, lo que incluye optimizar las operaciones matemáticas internas, reestructurar código y gestionar de manera más eficiente los recursos en memoria. Además, mejorar el rendimiento de Pandora puede requerir considerar alternativas más potentes de alojamiento web. Esto podría implicar el uso de servidores con mayor capacidad de procesamiento, almacenamiento más rápido y mayores capacidades de memoria. Otra posible futura mejora de Pandora sería revisar la posibilidad de implementar nuevas funcionalidades que amplíen su utilidad y capacidades, como podrían ser la firma de mensajes, el cifrado de mensajes para múltiples receptores, o la inclusión de más tipos de ataques y la mejora de la interfaz de usuario.

Capítulo 7

Conclusions and future lines

The main objective of this work has been the research of the cryptographic algorithms that form the basis of ECC, as well as the frameworks and tools necessary to develop a publicly accessible and free web tool, designed with an educational purpose, following the principles of usability and accessibility.

Throughout the development phase, several challenges were encountered, particularly related to the implementation and client-server linkage of the web tool. As previously mentioned, the initial plan was to create a local and personal-use tool for any user accessing the repository. However, due to the practical limitations of this approach, a full deployment was undertaken. This shift led to significant improvements in design and workflow, as well as a noticeable increase in server performance. The tool was deployed across two layers, each independent and equipped with its own hardware resources.

The results obtained from the attack tests are noteworthy. While demonstrating that elliptic curve cryptography is currently practical and secure, the tests also revealed that various cryptographic curves are vulnerable to attacks. A particularly notable case is the SECP256K1 curve, which is widely used as the foundation for numerous blockchain technologies and cryptocurrency operations, making its vulnerabilities globally impactful.

Efforts have been made to optimize the code as much as possible, but it is evident that, especially for the Pohlig-Hellman and Baby-Step Giant-Step attacks, the computational time required can be very high relative to the input values. These algorithms, by their nature, demand a large amount of computational resources, which can result in prolonged execution times. To address these issues, it is essential to improve the performance of the algorithms. This includes optimizing internal mathematical operations, restructuring code, and managing memory resources more efficiently. Additionally, improving the performance of Pandora may require considering more powerful web hosting alternatives. This could involve using servers with greater processing capacity, faster storage, and increased memory capabilities.Another potential future enhancement for Pandora would be to explore the possibility of implementing new functionalities that expand its utility and capabilities, such as message signing, message encryption for multiple recipients, or the inclusion of additional types of attacks and improvements to the user interface.

Capítulo 8

Presupuesto

En este capítulo se mostrará un desglose del presupuesto total del trabajo realizado. En total se han dedicado 226 horas de trabajo, con un coste unitario de 16€ la hora lo cuál suponen un coste en total de 3.606€. Además, se incluye el coste del dispositivo empleado para el desarrollo y ejecución de Pandora, que tiene un coste de 459,99€, por lo que el presupuesto total consta de 4.065,99€.

Descripción	**Cantidad (Horas)**	**Precio / Hora (€)**	**Total (€)**
Planificación del proyecto	20	16	320
Implementación del servidor	62	16	992
Implementación del cliente	84	16	1.344
Documentación	15	16	240
Análisis y recopilación de resultados	10	16	150
Redacción de la memoria	35	16	560
Ordenador HP 15-da0130ns	-	-	459,99
Total	226 horas	-	4.065,99

Tabla 8.1: Presupuesto total del proyecto

Apéndice A

Clases desarrolladas

A.1. Class Point

```
/**************************************************************************************
 *
 * Fichero point_service.py
 *
 **************************************************************************************
 *
 * Luis Marcelo Chinea Rangel
 *
 * Implementación de los puntos de una curva elíptica utilizada en ECC.
 *
 *
 **************************************************************************************/
import json
from typing import Any
from app.utils import simplify_fraction, euclid_extended


class Point:
    """Class representing a point on an elliptic curve.

    Attributes:
        a: Coefficient.
        field: Finite field of point's elliptic curve.
        x: x-coordinate of the point.
        y: y-coordinate of the point.
    """

    def __init__(self, params, x=0, y=0):
        self.a, self.field, self.x, self.y = params.a, params.field, x, y

    def __setattr__(self, __name: str, __value: Any) -> None:
        if __name in ["a", "field", "x", "y"] and not isinstance(__value, int):
            raise ValueError(f"Invalid value! {__name} must be an integer.")
        super().__setattr__(__name, __value)

    def at_infinity(self) -> bool:
        return (self.x, self.y) == (-1, -1)

    def point_zero(self) -> bool:
        return (self.x, self.y) == (0, 0)
```

```
def __eq__(self, other: "Point") -> bool:
    return (self.x, self.y) == (other.x, other.y)

def __neg__(self) -> "Point":
    return Point(self, self.x, -self.y)

def __add__(self, other: "Point") -> "Point":
    # Check if either point is at infinity or the point at zero
    if self.at_infinity() or self.point_zero():
        return other
    if other.at_infinity() or other.point_zero():
        return self

    # Check if points are inverses of each other
    if self == -other:
        return Point(self, -1, -1)

    # Check if points are the same
    if self == other:
        fraction = (3 * self.x**2 + self.a), (2 * self.y)
    else:
        fraction = (other.y - self.y), (other.x - self.x)

    # Handle case where simplified fraction is infinity
    if simplify_fraction(fraction) == (-1, -1):
        return Point(self, -1, -1)

    num, den = simplify_fraction(fraction)
    lamb = (num * euclid_extended(abs(den), self.field)[1]) % self.field
    x3 = (lamb**2 - self.x - other.x) % self.field
    y3 = (lamb * (self.x - x3) - self.y) % self.field

    return Point(self, x3, y3)

def __sub__(self, other: "Point") -> "Point":
    return self + (-other)

def __mul__(self, scalar: int) -> "Point":
    if scalar == 0 or self.at_infinity():
        return Point(self, -1, -1)

    result = Point(self)
    addend = Point(self, self.x, self.y)
    while scalar:
        if scalar & 1:
            if result == Point(self):
                result = Point(self, addend.x, addend.y)
            else:
                result += addend
        addend += addend
        scalar >>= 1
    return result

def __str__(self) -> str:
    if self.at_infinity():
        return "(O)"
    return f"({self.x}, {self.y})"
```

```
    def to_json(self) -> str:
        if self.at_infinity():
            return json.dumps({"x": -1, "y": -1}, indent=4)
        return json.dumps({"x": self.x, "y": self.y}, indent=4)
```

A.2. Class Curve

```
/*****************************************************************************
 *
 * Fichero curve_service.py
 *
 *****************************************************************************
 *
 * Luis Marcelo Chinea Rangel
 *
 * Implementación de la ECC en Python.
 *
 *
 *****************************************************************************/

from typing import Any
from sympy import isprime
from app.services.point_service import Point


class Curve:
    """Class representing an elliptic curve used in ECC.

    Attributes:
        a: Coefficient of the curve.
        b: Coefficient of the curve.
        field: Field of the curve.
        n: Order of the curve.
        points: List of points on the curve.
        base: Base point on the curve.
        public_keys: Dictionary of public keys.
        simulation: Flag to indicate if the curve is in simulation mode.
    """

    def __init__(self, a: int, b: int, field: int, simulation: bool = False) -> None:
        self.n = 0
        self.points = []
        self.public_keys = {}

        self.simulation = simulation
        self.a, self.b, self.field = a, b, field
        self.base = Point(self)

    def __setattr__(self, __name: str, __value: Any) -> None:
        if __name in ["a", "b", "field", "n"] and not isinstance(__value, int):
            raise ValueError(f"Invalid value! {__name} must be an integer.")
        if __name__ == "base" and type(__value) != Point:
            raise ValueError(f"Invalid value! {__name} must be a Point.")
        if (
            __name == "field"
```

```
        and not isprime(__value)
        and hasattr(self, "simulation")
        and self.simulation
    ):
        raise ValueError("Not a prime number!")
    super().__setattr__(__name, __value)
    if (
        __name in ["a", "b", "field"]
        and hasattr(self, "a")
        and hasattr(self, "b")
        and hasattr(self, "field")
        and getattr(self, "simulation", False)
    ):
        try:
            self.calculate_points()
        except ValueError:
            pass

def order(self) -> int:
    return len(self.points) + 1

def m(self, alph: str = None) -> int:
    return len(alph)

def calculate_points(self) -> None:
    self.points = []
    if self.a is None or self.b is None or self.field is None:
        raise ValueError(
            f"Parameters not set! a: {self.a}, b: {self.b}, field: {self.field}"
        )

    for x in range(self.field):
        y2 = (x**3 + self.a * x + self.b) % self.field
        for y in range(self.field):
            if y**2 % self.field == y2:
                self.points.append(Point(self, x, y))

def steps(self, point: "Point", scalar: int) -> list:
    result = []
    for i in range(1, scalar + 1):
        if not (point * i).at_infinity():
            result.append(point * i)
    return result

def encode(self, alph: str, msg: str) -> list:
    if not alph:
        raise ValueError("Alphabet not set!")
    if self.field <= (self.m(alph) * 2):
        raise ValueError(
            f"Invalid prime number! {self.field} must be greater than 2 * M (M = {self.m(alph)})
        )
    encoded = []
    h = self.field // self.m(alph)
    for char in msg:
        if char not in alph:
            raise ValueError(f"Invalid character! {char} not in alphabet.")
        char_i = alph.index(char)
        j = 0
```

```
            while True:
                x = (char_i * h + j) % self.field
                if x in [point.x for point in self.points]:
                    y = [point.y for point in self.points if point.x == x][0]
                    break
                j += 1
            encoded.append(Point(self, x, y))
        return encoded

    def decode(self, alph: str, points: list) -> str:
        if not alph:
            raise ValueError("Alphabet not set!")
        if self.field <= (self.m(alph) * 2):
            raise ValueError(
                f"Invalid prime number! {self.field} must be greater than 2 * M (M = {self.m(alph)})
            )
        decoded = ""
        h = self.field // self.m(alph)
        for point in points:
            x = point.x
            nearest = x
            while nearest % h != 0 and nearest >= 0:
                nearest -= 1
            decoded += alph[nearest // h]
        return decoded

    def encrypt(
        self,
        alph: str,
        msg: str,
        encryption,
        decryption,
        multiple: bool = False,
    ) -> list:
        if not self.base:
            raise ValueError("Base point not set!")
        if decryption == Point(self) and type(decryption) == Point:
            raise ValueError("Public key of the other party not set!")
        if isinstance(encryption, int) and encryption == 0:
            return [Point(self)] * len(msg)

        # Encrypt the message for multiple receivers
        if multiple:
            return [Qm + encryption for Qm in self.encode(alph, msg)]
        # Encrypt the message for a single receiver
        return [
            (Qm + (decryption * encryption), self.base * encryption)
            for Qm in self.encode(alph, msg)
        ]

    def decrypt(
        self, alph: str, points: list, private_k: int, public_k: "Point"
    ) -> str:
        if not self.base:
            raise ValueError("Base point not set!")
        msg = ""
        if private_k == 0:
            return msg
```

```
    for encrypted in points:
        decrypted = encrypted - (public_k * private_k)
        if decrypted is None:
            return ""
        msg += self.decode(alph, [decrypted])
    return msg

@staticmethod
def benchmark(curves: list, num_tries: int = 50) -> list:
    results = []
    for curve in curves:
        attacker = Setup(Curve(int(curve.a), int(curve.b), int(curve.field)))
        attacker.ec.base = Point(attacker.ec, int(curve.base.x), int(curve.base.y))
        attacker.ec.n = int(curve.n)
        success_count = 0
        start_time = time.time()
        for _ in range(num_tries):
            try:
                if attacker.generate_keys(False):
                    success_count += 1
            except Exception:
                continue
        end_time = time.time()
        elapsed_time = end_time - start_time
        results.append((success_count, success_count / num_tries, elapsed_time))
    return results
```

Apéndice B

Algoritmos implementados

B.1. SETUP

```
/*****************************************************************************
 *
 * Fichero setup.py
 *
 *****************************************************************************
 *
 * Luis Marcelo Chinea Rangel
 *
 * Implementación del ataque SETUP para ECC en Python
 *
 *
 *****************************************************************************/

import time
import secrets
import hashlib
from tabulate import tabulate

from app.services.point_service import Point
from app.services.curve_service import Curve


DELIMITER = "-----------------------------------------------------"

class Setup:
    def __init__(self, ec: "Curve", simulation: bool = False):
        self.ec = ec
        self.simulation = simulation
        self.c1 = None
        self.c2 = None

    @staticmethod
    def hash_point(point: "Point") -> int:
        return int(hashlib.sha256(point.x.to_bytes(32, "big")).hexdigest(), 16)

    def __public_key(self, n) -> "Point":
        return self.ec.base * n

    def __attack(self, a, b, h, e, V, v, M1, M2) -> tuple:
```

```
    ec = self.ec
    Z1 = M1 * a + M1 * b * v
    for j in range(2):
        for u in range(2):
            Z2 = Z1 + ec.base * h * j + V * e * u
            c2_attempt = Setup.hash_point(Z2)
            if ec.base * c2_attempt == M2:
                return c2_attempt
    return

def __verify(self, a, b, h, e, V, v, M1, M2) -> tuple:
    if self.c2 == self.__attack(a, b, h, e, V, v, M1, M2):
        return True, self.c2
    return False, None

def generate_keys(self, display: bool = True) -> None:
    if display:
        print(DELIMITER)
        print("Curve parameters:", flush=True)
        print(DELIMITER)
        time.sleep(0.1)
        print(f"a: {self.ec.a}", flush=True)
        print(f"b: {self.ec.b}", flush=True)
        print(f"p: {self.ec.field}", flush=True)
        print(f"n: {self.ec.n}", flush=True)
        print(DELIMITER)
        time.sleep(0.1)

    self.c1 = secrets.randbelow(self.ec.n - 2) + 2
    M1 = self.__public_key(self.c1)

    if display:
        print(f"Public key M1 generated", flush=True)
        print(DELIMITER)
        time.sleep(0.1)

    v = secrets.randbelow(self.ec.n - 2) + 2
    V = self.__public_key(v)
    if display:
        print(f"Public key V generated", flush=True)
        print(DELIMITER)
        time.sleep(0.1)

    a, b, h, e = [(secrets.randbelow(self.ec.n - 1) + 1) for _ in range(4)]
    j = secrets.randbelow(2)
    u = secrets.randbelow(2)
    if display:
        print(f"Generating random values", flush=True)
        time.sleep(0.1)
        print(DELIMITER)
        print(f"a: {a}", flush=True)
        print(f"b: {b}", flush=True)
        print(f"h: {h}", flush=True)
        print(f"e: {e}", flush=True)
        print(DELIMITER)
        print(f"j: {j}", flush=True)
        print(f"u: {u}", flush=True)
        print(DELIMITER)
```

```
            time.sleep(0.2)

        ec = self.ec
        Z = M1 * a + V * b * self.c1 + ec.base * h * j + V * e * u
        self.c2 = Setup.hash_point(Z)

        M2 = self.__public_key(self.c2)
        success, c2 = self.__verify(a, b, h, e, V, v, M1, M2)
        if display:
            if success:
                print("Attack successful!")
                print(DELIMITER)
                print(f"c2: {c2}")
            else:
                print("Attack failed!")
            print(DELIMITER)
        return success

    @staticmethod
    def measure_success_rate(ec_params, base_points, num_tries: int = 50) -> list:
        successes = [0 for _ in range(len(ec_params))]
        success_rates = [0 for _ in range(len(ec_params))]
        for params, base_point in zip(ec_params, base_points):
            attacker = Setup(Curve(params[0], params[1]))
            attacker.ec.field = params[2]
            attacker.ec.n = params[3]
            attacker.ec.base = Point(attacker.ec.a, attacker.ec.field, *base_point)
            success_count = 0

            for _ in range(num_tries):
                try:
                    if attacker.generate_keys(False):
                        success_count += 1
                except Exception as e:
                    continue
            successes[ec_params.index(params)] = success_count
            success_rate = success_count / num_tries
            success_rates[ec_params.index(params)] = success_rate

        return num_tries, successes, success_rates
```

B.2. Pohlig-Hellman

```
/*******************************************************************************
 *
 * Fichero pohlig_hellman.py
 *
 *******************************************************************************
 *
 * Luis Marcelo Chinea Rangel
 *
 * Implementación del algoritmo de Pohlig-Hellman ECC en Python
 *
 *
```

```
 ********************************************************************************/

import time
import math
from sympy import primefactors, mod_inverse

from app.services.point_service import Point
from app.services.curve_service import Curve

def prime_factors(n):
    factors = []

    while n % 2 == 0:
        factors.append(2)
        n //= 2

    for i in range(3, int(math.isqrt(n)) + 1, 2):
        while n % i == 0:
            factors.append(i)
            n //= i

    if n > 2:
        factors.append(n)

    return factors

class PohligHellman:
    def __init__(self, ec: "Curve", G: "Point", A: "Point"):
        self.ec = ec
        self.field = ec.field
        self.G = G
        self.A = A

    def __discrete_log(self, q: int, G: "Point", A: "Point") -> "int":
        R = Point(self.ec, -1, -1)
        for i in range(q):
            if R == A:
                return i
            R = R + G
        return -1

    def __resolve_congruences(self, congruences: list, mods: list) -> tuple:
        x = 0
        N = 1
        for ni in mods:
            N *= ni
        for ai, ni in zip(congruences, mods):
            Ni = N // ni
            mi = mod_inverse(Ni, ni)
            x = (x + ai * Ni * mi) % N
        return x

    def attack(self):
        factors = primefactors(self.field)
        congruences = []
        mods = []

        for q in factors:
```

```
            e = self.field // q
            Ae = self.A * e
            Ge = self.G * e
            log = self.__discrete_log(q, Ge, Ae)
            congruences.append(log)
            mods.append(q)

        x = self.__resolve_congruences(congruences, mods)
        return x

    @staticmethod
    def benchmark(curves: list, num_tries: int = 50) -> list:
        results = []
        for data in curves:
            curve = data[0]
            ec = Curve(int(curve.a), int(curve.b), int(curve.field))
            ec.base = Point(ec, int(curve.base.x), int(curve.base.y))
            A = Point(ec, int(data[1].x), int(data[1].y))
            attacker = PohligHellman(ec, ec.base, A)
            success_count = 0
            start = time.time()
            for _ in range(num_tries):
                alpha = attacker.attack()
                if ec.base * alpha == A:
                    success_count += 1
            end = time.time()
            elapsed_time = end - start
            results.append((success_count, success_count / num_tries, elapsed_time))
        return results
```

B.3. Baby-Step Giant-Step

```
/*****************************************************************************
 *
 * Fichero baby_step_giant_step.py
 *
 *****************************************************************************
 *
 * Luis Marcelo Chinea Rangel
 *
 * Implementación del ataque Baby-Step Giant-Step para ECC en Python
 *
 *
 *****************************************************************************/

 import time
from app.services.point_service import Point
from app.services.curve_service import Curve

DELIMITER = "---------------------------------------------------------------"

class BabyStepGiantStep:
    def __init__(self, ec: "Curve", m: int, G: "Point", A: "Point"):
```

```
        self.ec = ec
        self.m = m
        self.G = G
        self.A = A

    def __baby_step(self) -> list:
        baby_steps = [None] * self.m
        for i in range(self.m):
            result = self.G * (i + 1)
            if result.at_infinity():
                result = Point(self.ec, 0, 0)
            baby_steps[i] = result
        return baby_steps

    def __giant_step(self) -> list:
        giant_steps = [None] * self.m
        for i in range(self.m):
            result = self.G * (self.m * (i + 1))
            if result.at_infinity():
                result = Point(self.ec, 0, 0)
            giant_steps[i] = self.A - result
        return giant_steps

    def attack(self) -> int:
        found = False
        baby_steps = self.__baby_step()
        giant_steps = self.__giant_step()
        i = 0
        j = 0
        for j in range(self.m):
            for i in range(self.m):
                if baby_steps[i] == giant_steps[j]:
                    found = True
                    break
            if found:
                break

        if not found:
            return None

        alpha = i + j * self.m
        alpha = (alpha + self.ec.field) % self.ec.field
        return alpha

    @staticmethod
    def benchmark(curves: list, num_tries: int) -> None:

        results = []
        for data in curves:
            curve = data[0]
            ec = Curve(int(curve.a), int(curve.b), int(curve.field))
            ec.base = Point(ec, int(curve.base.x), int(curve.base.y))
            A = Point(ec, int(data[1].x), int(data[1].y))
            m = int(data[2])
            attacker = BabyStepGiantStep(ec, m, ec.base, A)
            success_count = 0
            start = time.time()
            for _ in range(num_tries):
```

```
        alpha = attacker.attack()
        if alpha is not None and ec.base * alpha == A:
            success_count += 1
    end = time.time()
    elapsed_time = end - start
    results.append((success_count, success_count / num_tries, elapsed_time))
return results
```

Apéndice C

Publicaciones en conferencias

Tanto la herramienta web Pandora como los resultados obtenidos de este trabajo han sido presentados y expuestos en la IX edición de las Jornadas de Investigación en Ciberseguridad (JNIC), que tuvo lugar en la Universidad de Sevilla en mayo de 2024, mediante una ponencia con el título *Visualización y Estudio de un Ataque en Criptografía de Curva Elíptica*, centrada en el análisis del ataque SETUP, presentada por mi cotutor Óscar Cigala Álvarez, y teniendo a mi tutora Pino Caballero Gil como coautora.

Además, también se ha enviado un artículo con los cambios más recientes de Pandora a la XVIII Reunión Española de Criptología y Seguridad de la Información (RECSI), que tendrá lugar en León en octubre de este mismo año 2024.

A continuación, se proporcionan los detalles del los congresos mencionados:

- **IX Jornadas Nacionales de Investigación en Ciberseguridad, Sevilla, 2024 mayo.**
  Visualización y Estudio de un Ataque en Criptografía de Curva Elíptica
  Óscar Cigala-Álvarez, Luis Chinea-Rangel, Pino Caballero-Gil
  Editorial Universidad de Sevilla, ISBN/ISSN: 978-84-09-62140-8, pp. 190-196.
  `https://2024.jnic.es/`
- **XVIII Reunión Española de Criptología y Seguridad de la Información, León, 2024 octubre.**
  Visualización e Implementación de Ataques a Sistemas Criptográficos de Curva Elíptica
  Óscar Cigala-Álvarez, Luis Chinea-Rangel, Pino Caballero-Gil
  `https://congresos.unileon.es/recsi2024/`

Bibliografía

[1] Sadurní, J. M. (2024). Alan Turing, el arma secreta de los aliados. https://https://historia.nationalgeographic.com.es/a/alan-turing-arma-secreta-aliados_16352

[2] National Institute of Standards and Technology. (2001). Advanced Encryption Standard (AES). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf

[3] IBM Integration Bus 10.0.0. (s.f.-b). https://www.ibm.com/docs/es/integration-bus/10.0?topic=overview-public-key-cryptography

[4] Rivest, R. L., Shamir, A., Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications Of The ACM, 21(2), 120-126. https://dl.acm.org/doi/abs/10.1145/359340.359342

[5] Koblitz, N. (1987). Elliptic Curve Cryptosystems. https://cdn.sanity.io/files/r000fwn3/production/72c46c82327564ca337538202ffd311234a25cbf.pdf

[6] Diffie, W., Hellman, M. (1976). New directions in cryptography. IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/document/1055638

[7] HTTPS - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN. (2023, 13 noviembre). MDN Web Docs. https://developer.mozilla.org/es/docs/Glossary/HTTPS

[8] Johnson, D., Menezes, A., Vanstone, S. (2001). The Elliptic Curve Digital Signature Algorithm (ECDSA). International Journal Of Information Security, 1(1), 36-63. https://doi.org/10.1007/s102070100002

[9] TLS (Transport Layer Security). (s.f.). Cyberzaintza. https://www.ciberseguridad.eus/ciberglosario/tls-transport-layer-security

[10] Haakegaard R., Lang J. (2015). The Elliptic Curve Diffie-Hellman (ECDH). http://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf

[11] Sutikno, S., Surya, A., Effendi, R. (1998). An implementation of ElGamal elliptic curves cryptosystems. . IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/743829

[12] Young, A., Yung, M. (1997). Kleptography: Using Cryptography Against Cryptography. In: Fumy, W. (eds) Advances in Cryptology — EUROCRYPT '97. EUROCRYPT 1997. Lecture Notes in Computer Science, vol 1233. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-69053-0_6

[13] Sommerseth, M.L. (2015). Pohlig-Hellman Applied in Elliptic Curve Cryptography. http://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Sommerseth+Hoeiland.pdf

[14] D. Shanks, “Class Number, a Theory of Factorization and Genera,” Proceedings of Symposium of Pure Mathematics, Vol. 20, 1969, pp. 415-440. - References - Scientific Research Publishing. (s.f.). https://www.scirp.org/reference/referencespapers?referenceid=245728

[15] Mohamed, E., Elkamchouchi, H. (2010). Elliptic Curve Kleptography. IJCSNS International Journal Of Computer Science And Network Security, VOL.10 No.6, June 2010. http://paper.ijcsns.org/07_book/201006/20100623.pdf

[16] Jackson, C. S. (s.f.). The Chinese remainder theorem. OPUS Open Portal To University Scholarship. https://opus.govst.edu/capstones_math/2/

[17] Equipo editorial, Etecé. (2023). Caja de Pandora - Qué es, mito, interpretación y versiones. Concepto. https://concepto.de/caja-de-pandora/

[18] Chinea Rangel, L. Pandora. (s. f.-b). https://pandora-ecc.netlify.app/

[19] Chinea Rangel, L., (s.f.-b). GitHub - iluzioDev/pandora. GitHub. https://github.com/iluzioDev/pandora

[20] Sonar. (s.f.). SonarCloud Online Code Review as a Service Tool. https://www.sonarsource.com/products/sonarcloud/

[21] Angular. (s.f.). Angular. https://angular.dev/overview

[22] Cloud Application Hosting for Developers | Render. (s.f.). https://render.com/

[23] Scale & Ship Faster with a Composable Web Architecture | Netlify. (s.f.). Netlify. https://www.netlify.com/

[24] Welcome to Python.org. (s.f.). Python.org. https://www.python.org/doc/

[25] NumPy documentation — NumPy v2.0 Manual. (s.f.). https://numpy.org/doc/stable/

[26] Seaborn: Statistical Data Visualization — Seaborn 0.13.2 Documentation. (s.f.). https://seaborn.pydata.org/

[27] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., . . . Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[28] PyTorch documentation — PyTorch 2.3 documentation. (s.f.-b). https://pytorch.org/docs/stable/index.html

[29] JavaScript with syntax for types. (s.f.). https://www.typescriptlang.org/

[30] JavaScript | MDN. (2023). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/JavaScript

[31] Welcome to Flask — Flask Documentation (3.0.x). (s.f.). https://flask.palletsprojects.com/en/3.0.x/

[32] Django. (s.f.). Django Project. https://www.djangoproject.com/

[33] Node.js — about Node.js®. (s.f.). https://nodejs.org/en/about

[34] Team, A. C. (s.f.). Angular material. Angular Material. https://material.angular.io/

[35] Material Design. (s.f.). https://m3.material.io/

[36] Chart.js | chart.js. (s.f.-b). https://www.chartjs.org/docs/latest/