Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Seguridad Informática

Software educativo para el aprendizaje de Criptografía de Curva Elíptica

Trabajo fin de estudio presentado por:	Omar Gallo Haddad
Tipo de trabajo:	Desarrollo software
Director/a:	Dr. Rafael A. Rodríguez Gómez
Fecha:	09/02/2022

Resumen

El interés por el estudio de las curvas elípticas, ha encontrado en el campo de la criptografía una aplicación práctica ampliamente difundida en los últimos tiempos denominada Cifra de Curva Elíptica (*Elliptic-curve Cryptography – ECC*). En comparación a otras formas de criptografía de clave pública, con ECC es posible obtener mayores niveles de seguridad con dimensiones de clave más cortas, permitiendo operaciones más rápidas y de menor costo computacional. Sin embargo, sus características matemáticas particulares hacen también que las curvas elípticas sean generalmente vistas como intrínsecamente más complicadas y menos intuitivas de aprender y entender que otros métodos más tradicionales.

Existen herramientas como CrypTool o JCrypTool que contienen funcionalidades para el estudio de curvas elípticas y ECC, pero que por lo general no son plataformas específicamente concebidas para el estudio de este particular elemento matemático.

En el presente trabajo, se lleva a cabo el desarrollo de una software educativo que ayuda a un estudiante de criptografía a asimilar las operaciones fundamentales involucradas en la Cifra de Curva Elíptica. Se hace énfasis en un enfoque interactivo, paso a paso, en donde se estudia:

- Suma de puntos en cuerpo finito (detalle de ecuaciones involucradas).
- Multiplicación de punto en cuerpo finito (comparación de rendimiento entre algoritmo directo y *double-and-add*).
- Propiedad abeliana de conmutatividad.
- Algoritmo de intercambio de clave de Diffie-Hellman con curva elíptica (*Elliptic-curve Diffie-Hellman – ECDH*).

El código referido al desarrollo de la aplicación descripta en el presente trabajo se encuentra publicado en el siguiente repositorio Github: https://github.com/omarsoftware/ecc

Palabras clave: Cifra de Curva Elíptica, ECC, Software Educativo, ECDH

Abstract

The interest for the study of elliptic curves has found in the field of cryptography a practical application widely spread in the recent times denominated Elliptic-curve cryptography (ECC). In comparison to other public key cryptography forms, ECC provides higher security with shorter key sizes, allowing operations to be faster and of lower computational costs. Nevertheless, its particular mathematical features also make elliptic curves to be generally seen as intrinsically more complicated and less intuitive to learn and understand than other more traditional methods.

There are tools like CrypTool or JCrypTool which contain functionalities for the study of elliptic curves and ECC, but they generally are not platforms specifically dedicated to the study of this particular mathematical element.

This work addresses the development of an educational software that helps cryptography students to assimilate the fundamental mathematical operations involved in Elliptic-curve Cryptography. An interactive and step-by-step approach is emphasized, in which the following is studied:

- Point addition over finite fields (detailed equations involved).
- Point multiplication over finite fields (performance comparison between direct algorithm and double-and-add).
- Abelian commutative property.
- Key-exchange algorithm of Diffie-Hellman with elliptic curves (Elliptic-curve Diffie-Hellman – ECDH).

The code of the application described in this work can be found in the following Github repository: https://github.com/omarsoftware/ecc

Keywords: Elliptic-curve Cryptography, ECC, Educational Software, ECDH

Agradecimientos

A Papá, Mamá, Dita y Abu por ser las personas más importantes de mi vida, y acompañarme y protegerme siempre. Los amo.

Al Dr. Rafael Alejandro Rodríguez Gómez por su constante e invalorable asistencia y guía en la realización de este trabajo.

Índice de contenidos

Índice Figuras

Índice de Tablas

Índice de Ecuaciones

1.Introducción

Con el avance de la ciencia de la computación y las comunicaciones, la seguridad informática se ha vuelto cada vez más relevante en los sistemas digitales, las vidas de las personas y el éxito de las empresas (Yildirim, 2016). Para que esto suceda, los avances y desarrollos en la criptografía han sido fundamentales, y sin los cuales no habría sido posible tal progreso (Gençoğlu, 2019), ya que es un factor esencial para poder proteger a la información sensible del accionar de actores maliciosos que busquen vulnerarla en todo tipo de ámbito y contexto (privado, público, civil, comercial, militar, financiero, etc.).

Cuando se habla de criptografía, se hace referencia a las distintas técnicas mediante las cuales se lleva a la información de su estado natural y entendible, a uno ilegible en donde solamente el receptor legítimo sea capaz de revertir esta transformación y volver a obtener los datos en su formato original. Todo aquel que intente revertir dicha transformación sin contar con las claves de cifrado y descifrado adecuadas, debe enfrentarse a una dificultad algoritmo-matemática tan grande que la fuerza de cómputo y el tiempo involucrados para tener éxito haga que éste desista de siquiera intentarlo. En definitiva, lo que se busca es proteger la confidencialidad de la información.

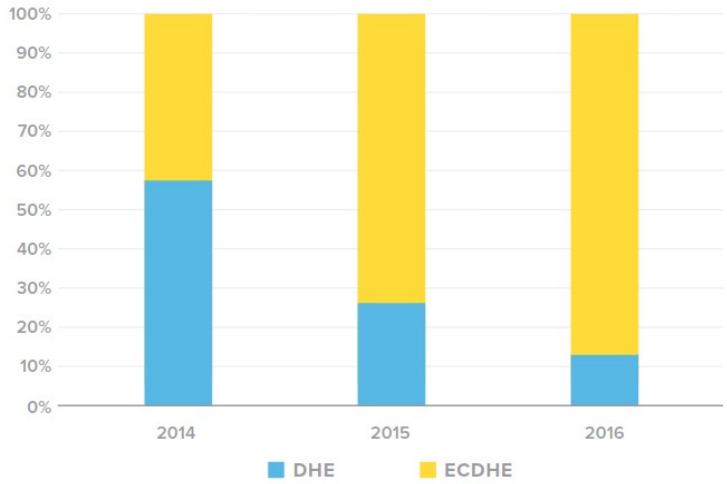
Para que éstas técnicas efectivamente funcionen y puedan proteger a la información y las comunicaciones de una forma considerada segura, es necesario implementar algoritmos que utilicen elementos y propiedades matemáticas especiales. Es de esta forma que aparecen en escena unos objetos matemáticos llamados curvas elípticas. Con ellas, resulta posible implementar esquemas criptográficos seguros que posean rasgos superadores en contraposición a esquemas similares pero construidos con otros elementos y principios matemáticos ya ampliamente conocidos. La aplicación de estas curvas en la criptografía es lo que se denomina cifra/criptografía de curva elíptica (*Elliptic-curve cryptography – ECC*), cuyo estudio y comprensión será precisamente el foco de atención del presente trabajo para el desarrollo de una aplicación educativa para estudiantes de ingeniería llevando a cabo un curso de criptografía.

1.1. Motivación

Distintos protocolos de comunicación e intercambio de clave tradicionales poseen ya su variante adaptada e implementada con curvas elípticas, cuya incorporación a la industria va aumentado gradualmente (Valenta et al., 2018). Por ejemplo, en la tabla 1 se puede observar que los autores hallaron ya para el año 2016 que un número considerable de equipos servidores/anfitriones (*hosts*) en Internet poseían compatibilidad con protocolos como TLS, SSH e IKE en sus variantes que trabajan con curvas elípticas. Las columnas ECDHE, secp*, x25519 y b-pool256r1 hacen referencia a qué curvas y algoritmos concretos existen implementados en dichos equipos en línea. Varios de ellos poseen compatibilidad con múltiples curvas en simultáneo, y la tendencia va en aumento.

Otra demostración del incremento en el uso de curvas elípticas, se encuentra en el reporte que analiza tendencias en Internet *TLS Telemetry Report* del año 2016 (Holmes, 2016). Allí, y como se puede ver en la figura 1, la empresa F5 Labs da cuenta de que para el año 2015 ya se puede apreciar que la mayoría de los servidores escaneados preferían trabajar con el algoritmo ECDHE (*Elliptic-curve Diffie-Hellman Ephemeral*), que es la versión con curvas elípticas del tradicional y aún en uso algoritmo DHE (*Diffie-Hellman Ephemeral*). Además, se puede observar que la tendencia para el siguiente año continuó siendo a la suba en favor del uso de ECDHE por sobre DHE.

Figura 1: *Uso de DHE vs ECDHE en internet.*



Fuente: Extraído de (Holmes, 2016).

Tabla 1: *Presencia de curvas elípticas en servidores de internet con los que se pudo intercambiar claves ECDHE (porcentajes con respecto a columna ECDHE).*

				Número de equipos anfitriones que dan soporte a...						
Protocolo	**Puerto**	**Fecha**	**BASE**	**ECDHE**	**secp224r1**	**secp256r1**	**secp384r1**	**secp521r1**	**x25519**	**b-pool256r1**
TLS	443 443	11/2016 08/2017	38.6M 41.0M	24.8M 28.8M	643.4K (2.6%) 811.6K (2.8%)	24.1M (97.0%) 25.0M (86.9%)	5.7M (22.9%) 9.1M (31.6%)	2.5M (10.2%) 2.2M (7.7%)	0 (0.0%) 740.7K (2.6%)	980.1K (3.9%) 2.4M (8.4%)
SSH	22	11/2016	14.5M	7.9M	0 (0.0%)	7.7M (97.8%)	7.5M (95.6%)	7.5M (95.4%)	6.1M (77.2%)	0 (0.0%)
IKEv1	500	11/2016	1.1M	215.4K	143.8K (66.8%)	211.8K (98.3%)	206.8K (96.0%)	152.8K (71.0%)	0 (0.0%)	0 (0.0%)
IKEv2	500	11/2016	1.2M	101.1K	4.1K (4.1%)	98.2K (97.1%)	98.0K (96.9%)	240 (0.2%)	0 (0.0%)	0 (0.0%)

Fuente: Elaboración propia basada en datos de (Valenta etal., 2018).

Incluso se puede observar que las principales autoridades de certificación de Internet a día de hoy (*Usage Statistics of SSL Certificate Authorities for Websites*, 2021) (quienes expiden los certificados que se emplean en las comunicaciones seguras), brindan soporte e incluso recomiendan en sus páginas web el uso de certificados basados en criptografía de curva elíptica. Algunos ejemplos son IdenTrust (*IdenTrust Global Common Certificate Policy*, 2021), DigiCert (*Elliptic Curve Cryptography (ECC)*, 2022) y Sectigo (*Elliptic Curve Cryptography vs RSA Certificates: What's the Difference?*, 2022).

La razón de que este tipo de criptografía se use cada vez más, yace en el hecho de que, en comparación a otras técnicas ya existentes, la criptografía de curva elíptica puede proveer un mayor nivel de seguridad, pero con dimensiones de clave más pequeñas. Por ejemplo, en la tabla 2 se puede observar, según distintos métodos (filas) cuáles son las longitudes de clave, en bits, que se consideran seguras según distintos algoritmos criptográficos existentes (columnas) para distintas fechas.

Tabla 2: *Dimensiones de clave en bits necesarias para obtener los mismos niveles de seguridad.*

Método	Fecha	Simétrico	Factorización en Módulo	Logaritmo Discreto		Curva Elíptica	Hash
				Clave	Grupo		
Lenstra/ Verheul	2022	87	1995 1568	154	1995	164	174
Lenstra Actualizado	2022	83	1446 1660	166	1446	166	166
ECRYPT	2018-2028	128	3072	256	3072	256	256
NIST	2019-2030	112	**2048**	224	2048	**224**	224
ANSSI	2021-2030	128	2048	200	2048	256	256
NSA	-	256	3072	-	-	384	384
RFC3766	-	-	-	-	-	-	-
BSI	2020-2022	128	2000	250	2000	250	256

Fuente: Elaboración propia basada en datos de (Giry, 2020).

Así, se puede observar que, la columna de curva elíptica posee longitudes de clave muy inferiores a, por ejemplo, los valores correspondientes a la columna de factorización de números enteros en módulo, que es justamente en el que se basa el algoritmo RSA, ampliamente utilizado dentro de la llamada criptografía asimétrica o de clave pública (Aryanti & Mekongga, 2018). En consecuencia, por ejemplo, para el *National Institute of Standards and Technology (NIST)* de los Estados Unidos, una clave de 2048 bits del algoritmo RSA es equivalente a una de 224 bits de cifra de curva elíptica (fila resaltada en color verde). Esto quiere decir que con una clave en curva elíptica de tamaño casi 10 veces inferior al de una clave RSA se puede alcanzar el mismo nivel de seguridad que ofrece este último algoritmo.

El hecho de que las claves sean más pequeñas implica tanto que los cómputos se realicen a velocidades más altas, como así también que el almacenamiento necesario para albergar a dichas claves sea inferior al del todavía vigente y conocido algoritmo RSA. Además, con una perspectiva a largo plazo, la cuestión de la longitud de las claves es crítico, puesto que conforme el tiempo avanza y la capacidad de los equipos aumenta, se necesita que las claves de los algoritmos criptográficos vayan aumentando para ir compensando el mayor poder de cómputo que va surgiendo con las nuevas generaciones.

Para que las curvas elípticas hayan ido ganando progresivamente terreno en su incorporación y uso en distintos ámbitos, han demostrado poseer una serie de características y propiedades matemáticas que permiten ventajas criptográficas por sobre otras técnicas y algoritmos anteriores. Sin embargo, ha sido precisamente la dificultad de sus principios matemáticos la que también ha hecho que su asimilación por parte de profesionales de la computación e implementación en distintos ámbitos, haya suscitado ciertas dificultades.

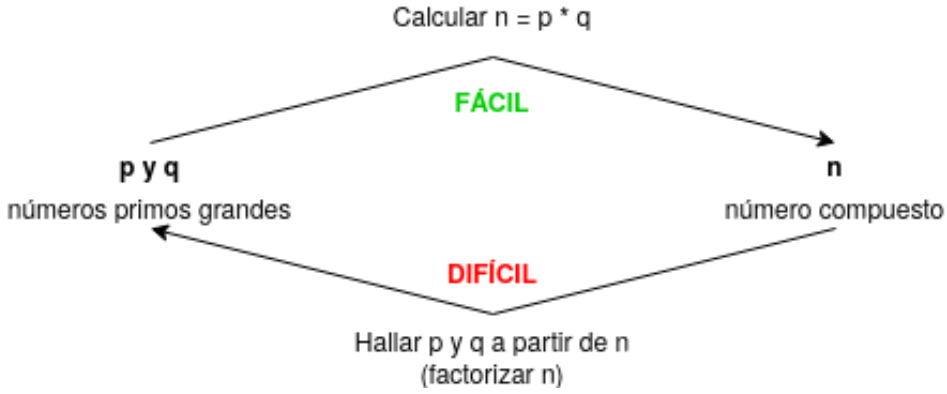
1.2. Planteamiento del trabajo

La matemática de curvas elípticas es de una naturaleza distinta y más compleja a la que se está acostumbrado para otros algoritmos criptográficos que utilizan operaciones que se pueden considerar más ampliamente conocidas o difundidas. Por ejemplo, en el caso del

algoritmo RSA, se emplea el problema de la factorización de enteros, y en Diffie-Hellman (Whitfield & Hellman, 1976) el del logaritmo discreto, en cuyos cálculos se emplean potencias y operaciones en módulo. En ambos casos, estas operaciones, si bien son elementales y rápidas de calcular cuando se dispone de todos los números involucrados, resultan ser computacionalmente muy costosas y difíciles de resolver si no se dispone de alguno de dichos valores y se intenta realizar una operación en sentido contrario (funciones unidireccionales) (Harkanson & Kim, 2017, p. 2). Por ejemplo, si un usuario busca proteger un trozo de información con una clave secreta, se buscaría que el proceso de cifrado y descifrado fuera computacionalmente rápido si el usuario legítimo introdujera su clave secreta en cualquiera de los dos sentidos, respectivamente. De lo contrario, no sería práctico proteger la información criptográficamente. Sin embargo, si el archivo cayera en las manos equivocadas, la idea es que al atacante debería resultarle computacionalmente muy costoso intentar descifrar (obtener la información en su texto plano original) dicha información sin conocer la clave original. Es decir, intentar todas las combinaciones de claves, una por una, hasta hallar la correcta debería incurrir en una dificultad computacional tal que el atacante desista de intentarlo (le tomaría demasiado tiempo y recursos computacionales hacerlo). Entonces, esta es una propiedad deseada y buscada en los esquemas criptográficos en general, y que los algoritmos de ejemplo mencionados logran realizando distintos tipos de cuentas matemáticas, las cuales, en su fundamento, no son extrañas incluso para el conocimiento común y corriente del público en general.

Un ejemplo que involucra operaciones sencillas pero sobre las cuales se puede construir un esquema criptográfico seguro, es el caso de la factorización de números enteros del algoritmo RSA (ver figura 2).

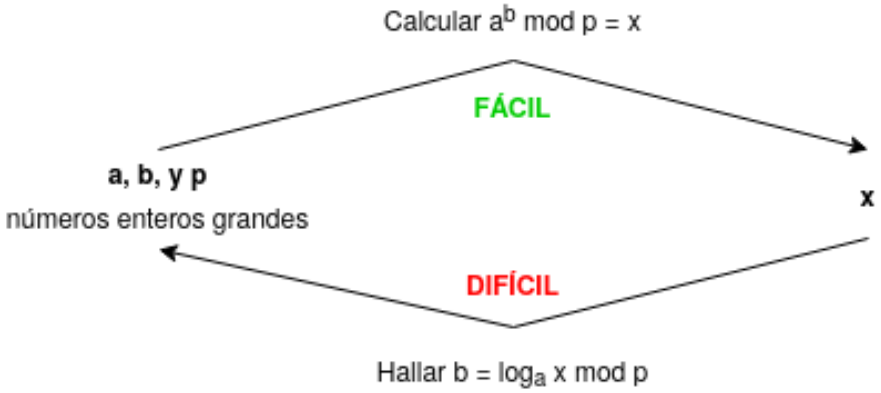
Figura 2: *Problema de factorización de números enteros (usado en RSA).*



Aquí, si poseemos dos números enteros primos *p* y *q*, resulta que la multiplicación $n = p*q$ es rápida de computar incluso siendo *p* y *q* de gran dimensión (este sería el camino sencillo en una dirección). Existen una variedad de algoritmos que permiten realizar esta operación básica de forma eficiente. Sin embargo, si a nosotros nos fuera dado el número resultante *n* y nos preguntaran cuáles son los dos valores *p* y *q* originales que multiplicados entre ellos arrojan el número *n* (los factores de n), veríamos que esto se trata, al día de hoy, de un problema considerado computacionalmente inabordable. Intentar ir probando todas las combinaciones de *p* y *q* que den como resultado *n* es muy costoso (este sería el camino en sentido contrario, y de difícil cálculo).

Por otro lado, el algoritmo de Diffie-Hellman basa su fortaleza en la dificultad de poder resolver el llamado problema del logaritmo discreto (ver figura 3). Aquí, partiendo de tres números *a*, *b* y *p*, calcular $a^b \ mod \ p = x$ es un problema considerado sencillo, fácil de computar. Sin embargo, intentar obtener el valor de *b* mediante $b = log_a \ mod \ p$ es considerado computacionalmente intratable.

Figura 3: *Problema del logaritmo discreto (usado en Diffie-Hellman).*

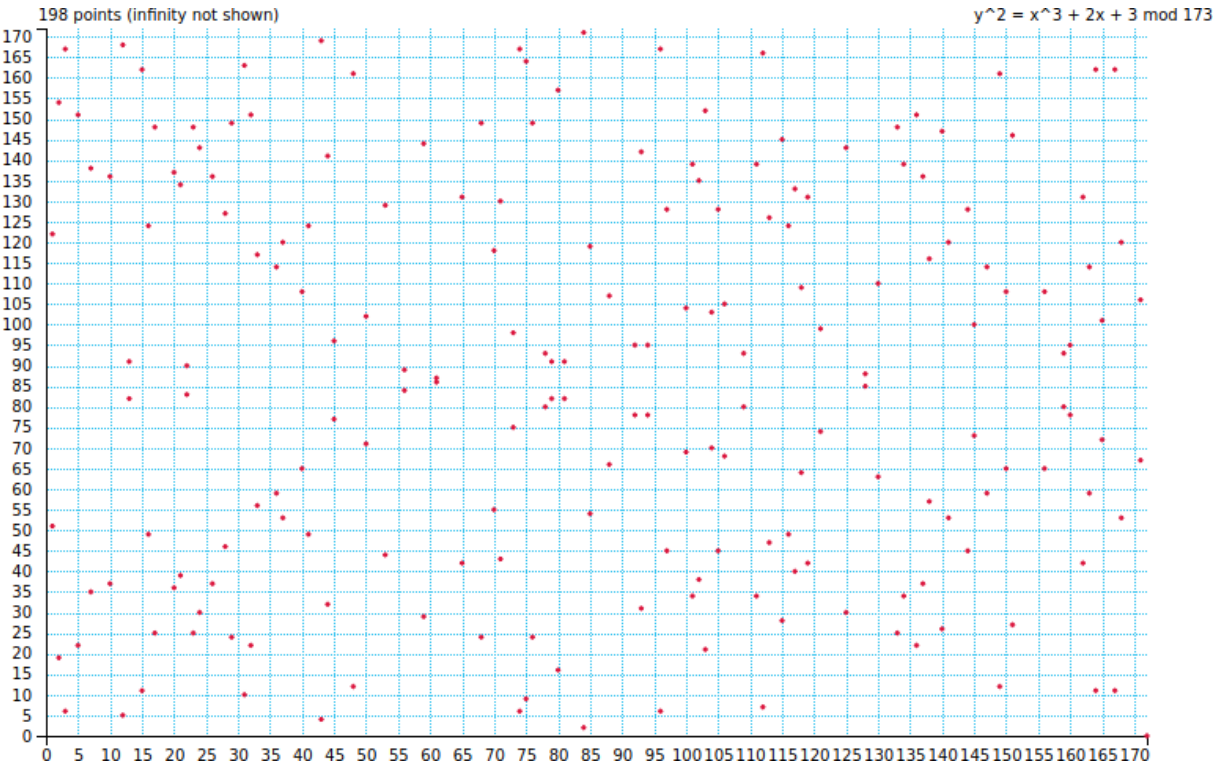


Así, en un esquema criptográfico asimétrico (se verá en el capítulo 2.3) el valor de *b* podría funcionar como la clave privada de un usuario (no se comparte con nadie y se lo protege con recelo), mientras que el valor $x=a^b mod \ p$ podría funcionar como un valor visible y accesible públicamente, ya que aunque todos tengan el valor de *x* en sus manos (no se busca ocultárselo a nadie), por el problema del logaritmo discreto le será computacionalmente muy dificultoso a cualquiera hacerse con el valor de *b*. Es como que *b* se "diluye" en el cálculo de *x*, haciendo que sea difícil obtenerlo a partir de *x*.

Entonces, estos son dos ejemplos en los que vemos cómo cuentas usualmente conocidas y sencillas (multiplicación de dos números enteros o el cálculo de logaritmos en conjunción con aritmética modular), pueden resultar en operaciones unidireccionales con características deseables para su uso en criptografía.

Ahora bien, en contraposición a lo recién explicado, en el mundo de las curvas elípticas se trabaja principalmente con una serie de operaciones básicas llamadas suma y multiplicación de puntos, cuyos significados no están vinculados con las tradicionales suma y multiplicación de números reales a los que se está comúnmente acostumbrados (E. Brown & Myers, 2002, pp. 642-643). Por el contrario, estas curvas trabajan sobre el plano, lo cual significa que se opera todo el tiempo con puntos de coordenadas *x* e *y* (abscisa y ordenada), y las operaciones matemáticas involucradas son muy particulares y con un significado especialmente concebido para el mundo de estas curvas (ver figura 4).

Figura 4: *Ejemplo de puntos sobre una curva elíptica sobre cuerpo finito.*



Fuente: Generado en (Grau, 2011).

De hecho, aquí el problema unidireccional sobre el cual se basa la fortaleza de la criptografía de curvas elípticas es el llamado logaritmo discreto para curvas elípticas (*Elliptic-curve Discrete Logarithm Problem – ECDLP*) para cuya comprensión es necesario entender las

operaciones básicas de suma y multiplicación para estas curvas. Aquí, por ejemplo, sumar puntos no equivale a sumarlos en un eje cartesiano como se estudia en geometría, sino que tiene una definición y significado distintos. Esto hace que estas operaciones propiamente dichas, como así también toda una serie de derivaciones de las mismas y de sus algoritmos asociados, resulten más difíciles de comprender para quienes no se han adentrado en el estudio de estos particulares elementos matemáticos. Una introducción a estas nociones teóricas y su aplicación concreta a una implementación algorítmica será presentada en el capítulo 2.

De esta forma, vemos que esto se torna en un inconveniente a abordar de cara a la formación de los especialistas y profesionales de la seguridad informática y en especial de la criptografía, quienes se verán cada vez más urgidos a comprender el funcionamiento de este tipo de cifrado. La criptografía de curva elíptica se va incorporando a cada vez más algoritmos y contextos referidos a la ciberseguridad y confidencialidad de la información, en donde la criptografía representa un eslabón de vital importancia en el conjunto de contramedidas que ofrece la seguridad informática ante las potenciales amenazas que sufre la información (Gençoğlu, 2019, p. 1).

Una forma de poder abordar la problemática del aprendizaje de las propiedades de las curvas elípticas y su aplicación a la criptografía, es la de desarrollar un software educativo enfocado en un estudiante de criptografía que busque asimilar dichos contenidos de una forma más sencilla e interactiva que con un abordaje totalmente teórico-matemático.

Como sustento de esto, tenemos que T. Dowling (Dowling, 2006) observa que se obtienen mejores resultados dictando un curso de criptografía a estudiantes de último año de ingeniería en software cuando se hace foco en una aproximación orientada a lo práctico en vez de hacerlo de una forma tradicional netamente matemática. Por su parte, los autores Adamović, Marko Šarac, Dušan Stamenković y Dalibor Radovanovic comunican obtener mayor asistencia de alumnos a clase, una cantidad superior de aprobados y una mejor y más alta distribución de notas al inclinarse por un dictado de contenidos mediante el uso de un software interactivo para explicar los principios fundamentales de la criptografía moderna (Adamovic et al., 2018).

Así, se alcanzará el objetivo de construir una aplicación con fines pedagógicos que le permita a un alumno de un curso universitario de criptografía para ingenieros, poder entender mejor estas particulares propiedades de las curvas elípticas y su aplicación en la llamada cifra de curva elíptica (*Elliptic-curve cryptography – ECC*).

1.3. Estructura del trabajo

El presente trabajo se encontrará organizado en los siguientes capítulos:

- **Capítulo 1**: introducción general del trabajo y a la estructura del mismo. Se hace mención a la importancia de las curvas elípticas y su cada vez mayor uso, y a la motivación de desarrollar un software interactivo para un mejor aprendizaje de sus propiedades y aplicaciones.

- **Capítulo 2**: revisión de las propiedades matemáticas y ecuaciones más relevantes referidas a las curvas elípticas y la cifra empleándolas, para comprender luego la implementación de las mismas en el desarrollo del software educativo en cuestión. Recopilación de información, características y estado del arte de otros softwares existentes que abordan la temática de ECC.

- **Capítulo 3**: planteo de objetivos generales y específicos para la confección de un software educativo enfocado en el estudio de curvas elípticas. Explicación de metodología de trabajo a emplear.

- **Capítulo 4**: análisis previo sobre cómo se llevará a cabo el desarrollo del software propuesto especificando funcionalidades, interacciones y características buscadas (casos de uso, requisitos funcionales y no funcionales).

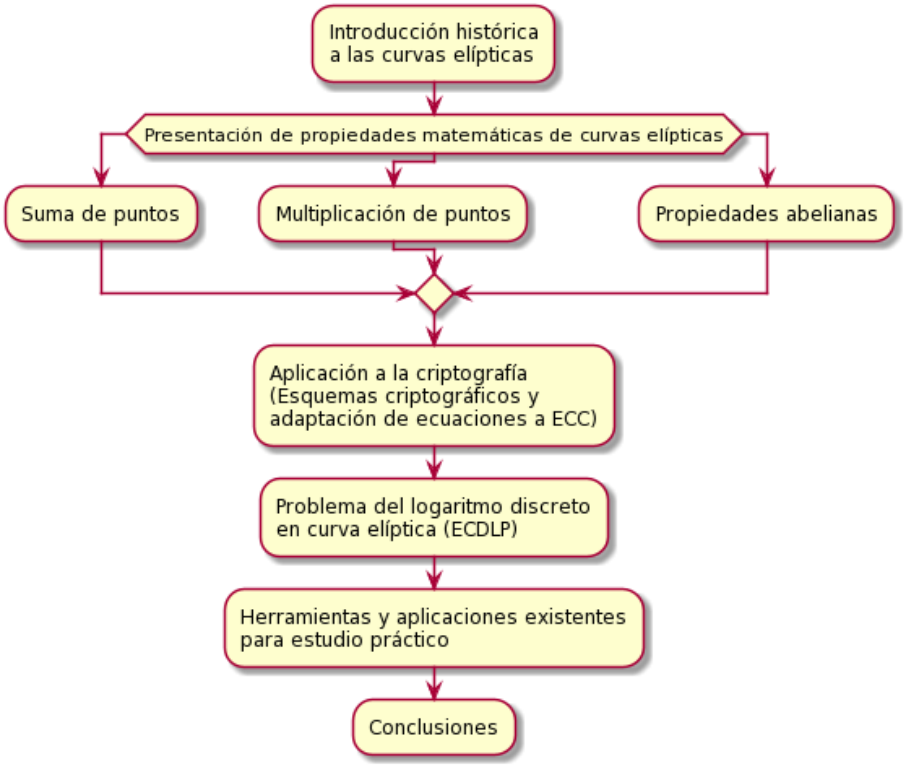
- **Capítulo 5:** cuestiones referidas al desarrollo concreto de la aplicación: tecnología a emplear, arquitectura y organización del código, relación entre los componentes involucrados, interfaces desarrolladas y pruebas realizadas.

- **Capítulo 6**: conclusiones y perspectivas a futuro. Se vincula el resultado final obtenido con los objetivos originalmente planteados, dejando abierta la posibilidad de incorporar más interfaces y funcionalidades posteriormente.

2.Estado del Arte

Para llevar a cabo el desarrollo de un software educativo, debemos primero realizar una debida introducción a las curvas elípticas, buscando comprender sus principales propiedades, aplicaciones y herramientas ya existentes para su estudio. Este será el objetivo del presente capítulo, cuyo resumen puede observarse en la figura 5.

Figura 5: *Resumen de estado del arte.*



En primer lugar, se hace una mención a los orígenes históricos del estudio de estas curvas desde un punto de vista matemático, y de los primeros signos de interés que surgieron para su uso en el campo de la computación.

A continuación, se pasa a una explicación teórica más detallada de las operaciones fundamentales de suma y multiplicación para curvas elípticas sobre el campo real, y de cómo se cumple con las llamadas propiedades de grupo abeliano.

Con esto último presente, entonces será posible entonces abordar y entender las consideraciones y adaptaciones necesarias a tener en cuenta para poder aplicar

concretamente a las curvas elípticas en el campo de la criptografía trabajando sobre cuerpos finitos.

De esta forma, se introducirá al problema del logaritmo discreto para curvas elípticas (*Elliptic-curve Discrete Logarithm Problem – ECDLP*), para comprender que sobre él yace la fortaleza criptográfica de la ECC, y que para entenderlo es necesario haber incorporado las operaciones básicas de suma y multiplicación anteriormente mencionadas.

Luego, teniendo en cuenta todas estas características básicas, se hará un análisis de distintas herramientas y librerías que se encuentran hoy en día disponibles para estudiar a las curvas elípticas y sus propiedades.

Finalmente, en las conclusiones se buscará comparar las características existentes y falencias observadas para dichas herramientas, con el objetivo de evaluar posibles puntos de mejora a incluir en el desarrollo de la aplicación con fines educativos que se busca concebir en este trabajo.

2.1. Contexto histórico de las curvas elípticas

Las curvas elípticas y sus propiedades han sido estudiadas por diversos matemáticos a lo largo de la historia, pudiéndonos remontar hasta el siglo II A.C. en los tiempos de Diofanto de Alejandría y su libro *Arithmetica* (E. Brown & Myers, 2002), marcando los inicios de la teoría de números. También han alcanzado tanto notoriedad académica como mediática, al ser eje central de la demostración del famoso Último Teorema de Fermat por parte del matemático británico Andrew Wiles (Wiles, 1995), logrando así dar respuesta definitiva a un problema planteado en el siglo XVII.

Luego de que H.W. Lenstra publicara el desarrollo de un algoritmo de factoreo de enteros con curvas elípticas (Lenstra, 1987), N. Koblitz y V. Miller proponen su uso dentro del campo de la criptografía de clave pública, dando lugar a lo que se conoce como Criptografía de Curva Elíptica (*Elliptic-curve Cryptography*, por su significado en inglés) (A. H. Koblitz et al., 2011, p. 782), en donde se explica que la principal ventaja que se encuentra con el uso de curvas elípticas en comparación a otros sistemas, es que se logra alcanzar los mismos niveles

de seguridad de éstos últimos con tamaños de clave menores. En consecuencia, tanto los tiempos de cómputo como los anchos de banda involucrados bajan y el rendimiento general sube.

Como resultado de esto, las aplicaciones que hacen uso de curvas elípticas se han ido extendiendo con el tiempo, cubriendo áreas como el intercambio de claves (*Elliptic-curve Diffie-Hellman*, *ECDH*), validaciones DNSSec, aplicaciones móviles, internet de las cosas (*Internet of Things - IOT*), comunicaciones vehiculares, Smart Grid, RFID, reconocimiento de patrón de iris, aplicaciones de salud, criptografía post-cuántica (Harkanson & Kim, 2017), e incluso firma digital (ECDSA) en sistemas de criptomonedas como bitcoin (Gennaro et al., 2016).

2.2. Matemáticas de Curvas Elípticas

En este apartado se estudiarán los conceptos básicos de la matemática de curvas elípticas, cuya comprensión es fundamental para poder llevar a adelante una exitosa implementación práctica de los algoritmos de criptografía de curva elíptica (ECC).

El desarrollo y razonamiento más detallado y formalmente presentado del contenido a continuación elaborado, puede encontrarse en (Kultinov, 2019), (Aglawe & Gajbhiye, 2012), y (Hankerson et al., 2003).

En su trabajo, D. Aglawe y S. Gajbhiye, explican las nociones teórico-matemáticas necesarias (operaciones con puntos y condiciones que deben cumplir algebraicamente las curvas elípticas) para luego implementar un software basado en MATLAB que muestra, paso a paso, si una cierta curva elíptica es además una curva abeliana cíclica, en cuyo caso se puede considerar apta para su uso en un criptosistema por formar un grupo.

Por su parte, Kirill Kultinov leva a cabo un detallado desarrollo matemático de las principales operaciones básicas de curvas elípticas (*point addition*, *point doubling*, y *point multiplication*, entre otros) como así también de algunos algoritmos que hacen uso de las mismas (por ejemplo, ECDH, ECDSA y ElGamal) acompañados de la implementación correspondiente en pseudocódigo de éstas últimas.

La forma normal de Weierstrass de una curva elíptica, responde a la siguiente ecuación:

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

Ecuación 1: *Forma normal de Weierstrass.*

Dependiendo de los valores de *a* y *b*, el gráfico de la curva resultante puede variar de distintas maneras, como en las siguientes figuras de ejemplo:

Figura 6: *Curva elíptica con a = 2 y b = 3*

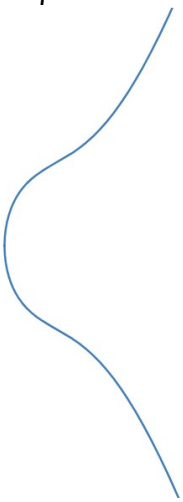
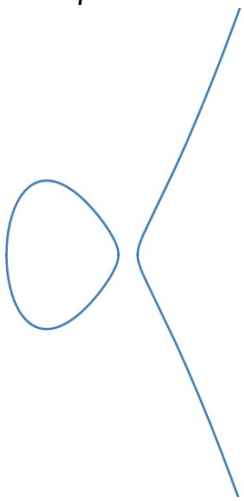


Figura 7: *Curva elíptica con a = -2 y b = -1.*



Además, se solicita como requerimiento que el discriminante sea distinto de 0:

$$4a^3 + 27b^2 \neq 0$$

Ecuación 2: *Discriminante en curva elíptica.*

Que esta condición se cumpla, asegura que la curva elíptica no sea singular (como en la figura 7), es decir, que la curva tenga raíces distintas (como en la figura 6). Esto hará que la curva sea aplicable a las operaciones de cifra con curvas elípticas a emplear posteriormente.

Además, se plantea la definición de un punto llamado “en el infinito” representado por la letra *O*, que será importante considerar en las explicaciones posteriores.

En consecuencia, se puede decir que el conjunto de los puntos que conforman a una curva elíptica está definido por:

$$E=\{(x,y): y^2=x^3+Ax+B\}\cup\{O\}$$

Ecuación 3: *Conjunto de puntos sobre una curva elíptica.*

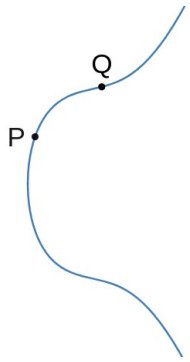
Entonces, para las curvas elípticas se define un conjunto de operaciones como la suma y la multiplicación, cuyos significados e interpretación es distinta a la que se está normalmente acostumbrado en la matemática tradicional. Esto se debe a que, en el contexto de las curvas elípticas, sumar y multiplicar involucrarán operaciones con puntos de la curva, que nos terminarán dando como resultado otros puntos de la misma curva.

La comprensión de estas operaciones es fundamental para entender luego el funcionamiento de los algoritmos de cifrado con curvas elípticas. La interpretación de dichas operaciones puede hacerse de forma geométrica, tal como se verá a continuación.

2.2.1. Suma de Puntos (*Point Addition*)

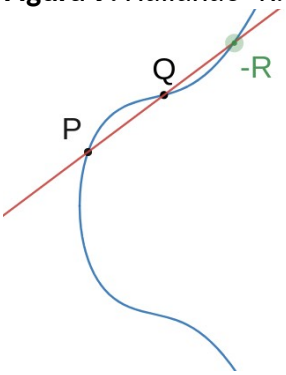
Dada una curva elíptica *E*, se parte de dos puntos *P* y *Q* distintos pertenecientes a *E*, como por ejemplo:

Figura 8: *P y Q sobre curva E.*



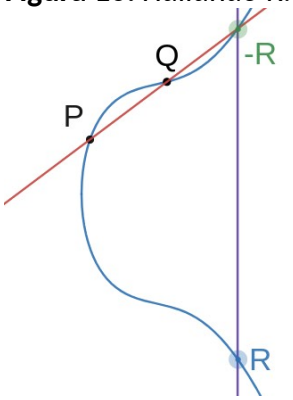
La suma de los puntos *P* y *Q* para una curva elíptica se interpreta geométricamente de una forma particular. En primer lugar, se traza una recta que pase por los puntos *P* y Q, y el punto de intersección hallado entre esta recta y la propia curva se la llama *-R*:

Figura 9: *Hallando -R.*



Siempre que se trace una recta entre dos puntos, dicha recta se encontrará con otro punto de la curva, en este caso llamado *-R*. Ahora, se refleja el punto *-R* en el eje *X*, y a dicho punto se lo denomina *R*:

Figura 10: *Hallando R.*



Decimos entonces que *P + Q = R*

Si pasamos de un análisis geométrico a uno analítico, veremos que si tenemos:

$$P(x_P, y_P) + Q(x_Q, y_Q) = R(r_P, r_P)$$

Entonces, la recta y que pasa por *P* y *Q*, y la pendiente λ de la misma son:

$$y = \lambda x + v$$

Ecuación 4: *Ecuación de una recta.*

$$\lambda = \frac{y_q - y_p}{x_q - x_p}$$

Ecuación 5: *Pendiente de la recta por la que pasan P y Q.*

Si tomamos estas ecuaciones y reemplazamos en la ecuación 1, veremos que trabajando algebraicamente arribaremos a las ecuaciones para calcular las coordenadas del punto *R*:

$$x_R = \lambda^2 - x_P - x_Q, y_R = \lambda(x_P - x_R) - y_P$$

Ecuación 6: *Coordenadas de punto P+Q, con P ≠ Q.*

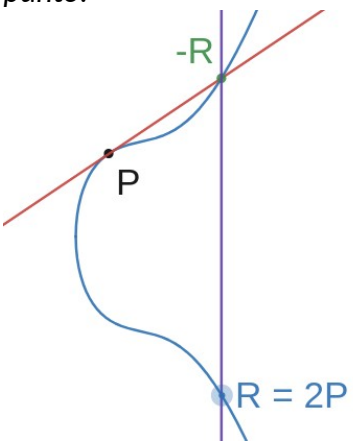
Esta operación aquí planteada será fundamental en el desarrollo del resto de las operaciones de curva elíptica como así también del cifrado de curva elíptica.

2.2.2. Duplicación de Punto (*Point-Doubling*)

Si ahora resultara que *P* y *Q* coinciden en el mismo lugar, es decir que *P = Q* (son iguales), entonces decimos que *P + P = 2P*.

En consecuencia, y en relación a lo visto anteriormente para la suma de dos puntos, como tenemos un solo punto *P*, la recta que pasa por *P* termina siendo la recta pendiente a la curva que pasa por *P* (la derivada).

Figura 11: *Duplicación de punto.*



Decimos que el punto *R* resultante es *2P*.

Entonces, partiendo de la ecuación de una recta en la ecuación 4 y reemplazándola en la ecuación 1, al derivar obtendremos:

$$2y\frac{dy}{dx}=3x+a$$

De esta forma, si reemplazamos los valores de *x* e *y* por X_P e Y_P respectivamente, obtendremos que la pendiente es:

$$\lambda=\frac{3x_p^2+a}{2y_p}$$

Ecuación 7: *Pendiente de la recta tangente que pasa por P.*

Continuando con el trabajo algebraico correspondiente, se arriba a que las ecuaciones para calcular las coordenadas del punto *R* son:

$$x_R=\lambda^2-2x_P, y_R=\lambda(x_P-x_R)-y_P$$

Ecuación 8: *Coordenadas de punto 2P.*

2.2.3. Punto en el infinito (*O*)

Cuando disponemos de un punto *P* y le sumamos el punto "en el infinito" *O*, obtendremos como resultado el mismo punto *P* en cuestión. Esto se debe a que al trazar la recta entre ambos, obtenemos una línea vertical que se cruzará con el punto *-P*. En consecuencia, al reflejar el mismo, obtendremos P:

$$P+O=P$$

Ecuación 9: *P + O.*

Además, la suma de *O* consigo mismo, da *O*:

$$O+O=O$$

Ecuación 10: *O + O.*

Por otro lado, si sumamos *P* con su reflejo *-P*, como la pendiente entre ambos es infinita por tratarse de una recta vertical, obtendremos que el único punto en el que puede intersectarse con la curva elíptica es *O*. Por ende:

$$P+(-P)=O$$

Ecuación 11: *P + (-P)*.

2.2.4. Multiplicación de Punto (*Point Multiplication*)

En base a los puntos anteriormente detallados, si ahora se nos pidiera realizar la multiplicación del punto *P* por una cantidad escalar *m*, esto equivaldría a realizar la suma de *P* consigo mismo una *m* cantidad de veces. Por ejemplo, si *m = 3* entonces veríamos que *3P = P + P + P*.

La forma de resolver esto sería partir del punto *P*, realizar una duplicación de *P*, y luego realizar ahora la suma entre *2P* (recientemente calculado) y *P*, obteniendo entonces el valor *3P* (esto lo repetiríamos *m* veces para valores superiores de *m*). A esto se lo conoce como método directo o trivial.

Sin embargo, luego se mencionará en la sección 2.4.4 al algoritmo *double-and-add*, el cual es más eficiente para calcular la multiplicación de puntos. De hecho, es con el que se pueden realizar cómputos con números de gran dimensión, puesto que, de lo contrario, con el método directo se tardaría demasiado por lo ineficiente que es.

Ahora bien, así como la operación de multiplicación se sustenta en la de la suma, los algoritmos criptográficos concretamente hablando (ej.: ECDH) terminarán realizando operaciones que involucran la multiplicación de puntos por números escalares. De hecho, es sobre esta operación en la que se basa la fortaleza de los esquemas criptográficos de curva elíptica, por la dificultad de resolver el problema del logaritmo discreto de curva elíptica (ECDLP), que se menciona en la sección 2.4.6.

2.2.5. Propiedades de la Suma

Teniendo en cuenta los temas antes vistos, es posible enunciar las propiedades de la suma de puntos en una curva elíptica *E*:

- **Cerrada**: dados dos puntos P y Q pertenecientes a E, tenemos que $P + Q = R$, en donde R es un elemento que se encontrará dentro de E.

- **Existe elemento Identidad**: el punto O funciona como el elemento identidad en la suma ya que, según la ecuación 9, $P + O = O + P = P$ para todo punto P perteneciente a E.

- **Existe elemento Inverso**: dada la ecuación 11, para cada punto P perteneciente a E existe un punto P' (su reflejo, es decir $-P$) tal que $P + P' = P' + P = O$.

  Por ende, si $P = (x, y)$ y $-P = (x, -y)$, tenemos que $P - P = O$.

- **Asociatividad**: dados tres puntos P_1, P_2, y P_3, se cumplirá que $(P_1+P_2)+P3 = P_1+(P_2+P_3)$

Al cumplir con estas cuatro propiedades, se dice que la curva E para la suma conforma un grupo (elemento matemático).

Sin embargo, también se cumple la siguiente propiedad:

- **Conmutatividad**: dados dos puntos P_1 y P_2, tenemos que $P_1 + P_2 = P_2 + P_1$.

Al cumplirse esta quinta propiedad, tenemos que no solamente se trata de un grupo como se mencionó anteriormente, sino que además es un grupo abeliano. Este hecho conllevará a que las curvas elípticas sean objeto de interés para el campo de la criptografía. Esta es una de las razones por las cuales con ellas se puedan implementar esquemas criptográficos seguros.

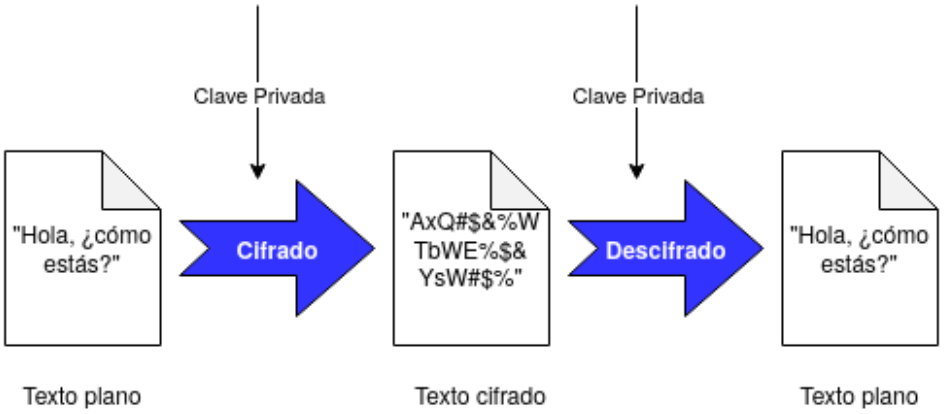
2.3. Esquemas Criptográficos

Para poder entender cómo se utilizan las curvas elípticas en el ámbito de la criptografía, es preciso entender los dos esquemas preponderantes: simétrico y asimétrico. El uso de las curvas elípticas se ubica en el marco de la criptografía asimétrica o también llamada de clave pública.

2.3.1. Esquema Simétrico o de Clave Privada

El primero de los esquemas criptográficos fue el simétrico o de clave secreta, en donde tanto emisor como receptor deben conocer y utilizar la misma clave para tanto cifrar como descifrar un mensaje, respectivamente. Las técnicas que se ajustan a esta descripción se han aplicado a lo largo de la historia, remontándonos incluso a los tiempos de los árabes, griegos y romanos (Dooley, 2013), y siguen vigentes al día de hoy para algoritmos como AES, TwoFish, Serpent y otros.

Figura 12: *Esquema de cifra simétrica o clave privada.*



Fuente: Creado con (Diagrams.Net, 2022).

En este caso, A toma un mensaje M en texto plano (por ejemplo: “Hola, ¿cómo estás?”) y lo cifra con una clave secreta, lo cual hace que el mensaje deje de ser legible, y se transforme en una cadena de caracteres y símbolos que no tienen sentido. Entonces, el receptor B toma dicho mensaje sin sentido, y utilizando la misma clave que usó A, aplica el proceso inverso y descifra el mensaje para, ahora sí, leer “Hola, ¿cómo estás?”. A continuación B, puede repetir el mismo procedimiento, para contestarle, por ejemplo, el mensaje “Bien, ¿y tú?”.

El inconveniente que tiene el esquema simétrico y los algoritmos asociados a él, es el mismo que se ha tenido a lo largo de la historia: cómo lograr que, de forma segura, tanto emisor como receptor puedan intercambiarse y ponerse de acuerdo sobre qué clave secreta van a emplear partiendo de la premisa de que el medio de comunicación no es seguro.

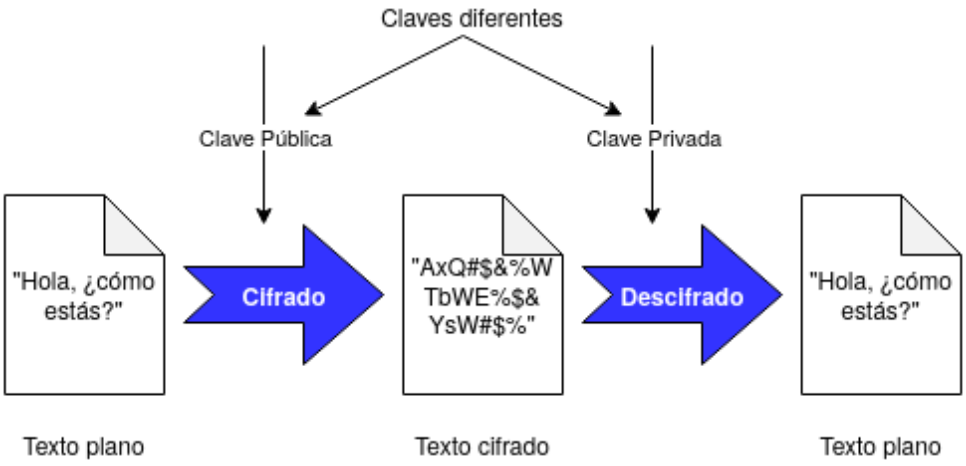
Es decir, para que este esquema funcione, se parte de la precondición de que en algún momento y de alguna forma A o B tuvieron que encargarse de que su contraparte se hiciera

con la clave secreta que ambos emplearán. Este es un punto débil en el esquema criptográfico simétrico. Por ejemplo, si A le envía la clave secreta a B por medio de una carta o bien diciéndoselo en voz alta, o escribiéndolo en un chat o en un correo electrónico, es posible que alguien pueda interceptar/ver/escuchar dicha clave enviada a través del canal de comunicación. Si un atacante logra esto, entonces la seguridad del esquema cae y la confidencialidad de la información será completamente vulnerada.

2.3.2. Esquema Asimétrico o de Clave Pública

Sin embargo, en el año 1976, los científicos de la computación Whitfield Diffie y Martin E. Hellman, producen un punto de quiebre en la historia de la criptografía, presentando el esquema criptográfico asimétrico o de clave pública (Whitfield & Hellman, 1976).

Figura 13: *Esquema de cifra asimétrica o de clave pública*



Fuente: Creado con (Diagrams.Net, 2022).

Aquí, emisor y receptor contarán, cada uno, con dos claves distintas: una privada (guardada con recelo por su dueño para que nadie la tenga) y otra pública (expuesta por su dueño y visible a absolutamente todo el mundo sin contrariedades). De esta forma, en este ejemplo de una comunicación entre dos participantes, habrá un total de 4 claves distintas. A tiene su clave privada y pública, y B tiene su clave privada y pública.

Las claves privada y pública de un participante en la comunicación son diferentes, y se encuentran matemáticamente vinculadas de tal forma que una funciona como la operación

contraria de la otra (son inversas), en donde una se utiliza para cifrar y la otra para descifrar. Lo que se hace con una, se revierte con la otra. Además, ese vínculo matemático entre ambas es tal que no será computacionalmente posible arribar a la clave privada a través de la mera posesión de la clave pública.

La idea fundamental en este esquema radica en el hecho de que A y B podrán establecer una comunicación segura entre ambos sin necesitar nunca compartir o enviar sus claves secretas en texto plano con el otro, aunque las claves públicas de ambos sí se encontrarán visibles para todos desde un comienzo.

Así, si el emisor A quiere enviarle un mensaje secreto M al receptor B, entonces A cifrará M utilizando la clave pública de B, y será solamente B quien pueda descifrar el mensaje cifrado utilizando su propia clave privada. Con esto último, se logra confidencialidad en la información (tal como sucedía en el esquema simétrico), pero este esquema permite además otras características de las cuales adolecen los simétricos. Por ejemplo, si A cifra su mensaje con su propia clave privada, entonces dicho mensaje podrá ser descifrado con la clave pública de A. Como la clave pública de A es conocida por todos (de allí que sea "pública"), entonces todos (incluso B) pueden descifrar el mensaje que ha enviado A. Esto último parecería no tener sentido desde el punto de vista de la confidencialidad, pero sí lo tiene para la autenticación del emisor (asegurarnos de que el mensaje fue enviado por quien se supone que debía enviarlo) y la integridad del mensaje (que la información enviada no ha sido alterada en su camino hasta arribar al destinatario). Esto sucede porque, ya que la clave pública de A es la única que puede revertir el efecto de cifrado que tuvo la clave privada de A sobre el mensaje, todos pueden estar seguros de que el mensaje que han descifrado le pertenece efectivamente a A (de lo contrario, no podrían haber descifrado el mismo). Si además combinamos esto con el uso de funciones hash (funciones unidireccionales que pueden ser aplicadas como huellas digitales de la información) es posible conferir también a la comunicación de la propiedad de integridad. Esta combinación es el principio fundamental de la llamada firma digital.

2.3.3. Algoritmo de Diffie-Hellman (*DH*)

El llamado algoritmo de intercambio de claves de Diffie-Hellman (DH) permite que, dos emisores A y B arriben, por separado, a una misma clave secreta compartida, realizando una serie de operaciones matemáticas en las que se verán involucradas sus claves públicas y privadas (más otros valores adicionales). Para ello, A y B se enviarán información mutuamente de tal forma que si un tercero lograra hacerse con esta información intercambiada a través del medio inseguro, éste no logrará derivar ni las claves privadas de A y B, ni tampoco la clave secreta final que ambos usarán.

Para ello, este algoritmo utiliza como base matemática el denominado problema del logaritmo discreto, que es de naturaleza unidireccional y que involucra las operaciones de potenciación y módulo. Esto quiere decir que, conociendo todos los valores involucrados, hacer los cómputos en cuestión es rápido y sencillo. Sin embargo, si ciertos valores son desconocidos, entonces intentar hallar los demás faltantes (con lo que se podría descifrar la información) por medio del uso de fuerza bruta se tornaría computacionalmente muy costoso. Esto se acentuará en la medida de que los valores que se utilicen sean cada vez más grandes.

En términos generales, las operaciones de cifra simétrica son más veloces que las de esquema asimétrico, aunque este último permite intercambio seguro de claves, firma digital y una eficiente gestión de claves, pese a ser considerablemente más lentos que los simétricos.

2.4. Aplicación de Curvas Elípticas en la Criptografía

Los tópicos previamente presentados sirven para ubicarse en contexto y poder empezar a trabajar concretamente en las aplicaciones de las curvas elípticas en el campo de la criptografía.

Las propiedades abelianas vistas para la suma de curvas elípticas, hacen que éstos elementos matemáticos sean apropiados para poder implementar sistemas criptográficos, y además su uso se puede considerar seguro, ya que la base de su fortaleza reside en el

llamado Problema del Logaritmo Discreto para Curvas Elípticas (*Elliptic-curve Discrete Logarithm Problem – ECDLP*).

Es por esto que se puede construir un esquema de cifrado asimétrico utilizando curvas elípticas, en donde dos emisores A y B dispongan de claves privadas y públicas para trabajar con las funcionalidades ya descriptas para el esquema de cifra asimétrica. Por ejemplo, es posible implementar el algoritmo Diffie-Hellman en una versión que utilice curvas elípticas (*Elliptic-curve Diffie-Hellman – ECDH*).

2.4.1. Trabajando en un cuerpo finito

Una primera cuestión a entender cuando se habla de curvas elípticas y su aplicación a la criptografía, son las consideraciones especiales a tener en cuenta a la hora de pasar de lo teórico a lo práctico.

Si bien se hizo una explicación introductoria a las operaciones fundamentales de curva elíptica (suma y multiplicación) mediante una interpretación gráfica, lo cierto es que, aunque es más sencillo intentar comprender estas operaciones en el campo de los números reales visualizando su interpretación geométrica, en el campo práctico de la criptografía computacional se suele trabajar con números enteros en campos finitos por cuestiones de seguridad y eficiencia (Harkanson & Kim, 2017). Conforme los valores involucrados en las ecuaciones de la sección 2.2 van creciendo en dimensión (por ejemplo, la operación de multiplicación), se irán sucesivamente obteniendo, de forma cada vez más notable valores fraccionarios (con coma) para las distintas coordenadas de los puntos que se computen.

Esto generará que se vaya propagando y arrastrando un error producto del redondeo de decimales y empeorando la estabilidad numérica (Higham, 2002, p. 26). En consecuencia, se va quitando precisión de forma cada vez más acentuada a los cálculos, y los mismos terminan careciendo de sentido (no se terminarían respetando las propiedades anteriormente vistas en las distintas operaciones).

Es por ese motivo que para la aplicación de curvas elípticas a la criptografía, se inclina por el uso de números primos en combinación con aritmética modular sobre cuerpos finitos, lo

cual conllevará que se puedan realizar cálculos sin errores de redondeo de decimales, puesto que se operará todo el tiempo con coordenadas de tipo entero.

Es decir, por ejemplo, partiendo de un punto de coordenadas enteras y realizando una operación como la multiplicación, se terminará devolviendo otro punto de la curva con también coordenadas enteras.

2.4.2. Cuerpo finito GF(q)

Ahora, las curvas elípticas a emplear en los procesos criptográficos, pasarán a estar representadas por un conjunto discreto de puntos que conformarán lo que se denomina como un Cuerpo o Campo de Galois.

Un cuerpo o campo es un conjunto de elementos sobre los cuales se definen dos operadores binarios + y * que a su vez cumplen las propiedades de integralidad de dominio (identidad multiplicativa e inexistencia de divisores iguales a 0) y las propiedades abelianas anteriormente descritas (Kultinov, 2019, p. 10). Además, existe inverso multiplicativo . Esto último significa que para todo elemento a perteneciente al campo, existe un elemento a^{-1} tal que $aa^{-1} = (a^{-1})a = 1$.

Por su parte, el orden de un cuerpo es el número de elementos en un cuerpo dado.

Entonces, dado un número primo q, se define un cuerpo de Galois de orden q como el conjunto de enteros $Z_p = \{0, 1, \ldots, q-1\}$ en conjunto con la operación módulo q. En este caso, el inverso multiplicativo establece que para todo elemento w perteneciente a Z_p y en donde w es distinto de 0, existe un elemento en Z_p que cumple $w * z \equiv 1 \pmod q$.

2.4.3. Adaptación de ecuaciones a ECC

Como consecuencia de trabajar sobre campos finitos, las ecuaciones desarrolladas en la sección 2.2 para las operaciones elementales de curvas elípticas sufrirán una serie de modificaciones producto de trabajar ahora con matemática modular.

La ecuación 1 que describe una curva elíptica, pasa ahora a ser:

$$y^2 \equiv x^3 + ax + b \pmod{q}$$

Ecuación 12: *Curva elíptica módulo q.*

La consecuencia de este cambio, es que ahora, si tomáramos la curva elíptica $y^2 = x^3 + 2x + 3$ (cuyo gráfico ya se mostró en la figura 6), y la ajustáramos a la ecuación 12 tomando como módulo q = 13, obtendríamos:

$$y^2 \equiv x^3 + 2x + 3 \pmod{13}$$

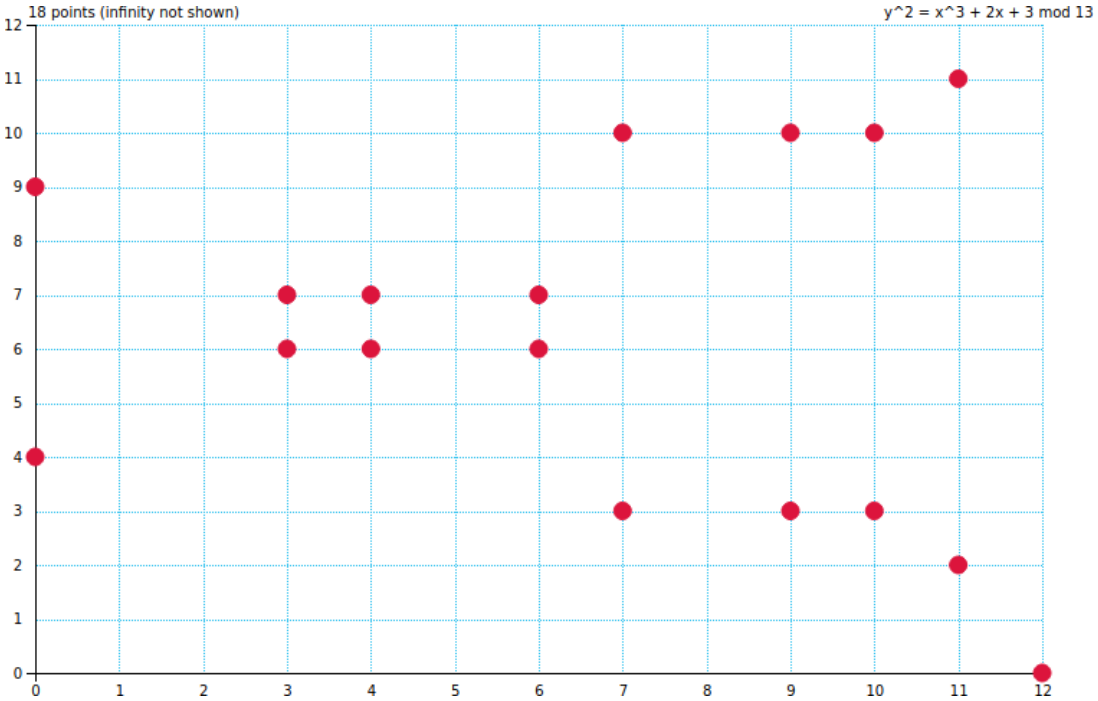
Tabla 3: *Cálculo de puntos de curva elíptica sobre cuerpo finito.*

x	$x^3 + 2x + 3 \pmod{13}$	y	$y^2 \pmod{13}$	Puntos que cumplen $y^2 \equiv x^3 + 2x + 3 \pmod{13}$
0	3	0	0	(0, 4), (0, 9), (3, 6), (3, 7), (4, 6), (4, 7), (6, 6), (6, 7), (7, 3), (7, 10), (9, 3), (9, 10), (10, 3), (10, 10), (11, 2), (11, 11), (12, 0), O (punto en el infinito)
1	6	1	1	
2	2	2	4	
3	10	3	9	
4	10	4	3	
5	8	5	12	
6	10	6	10	
7	9	7	10	
8	11	8	12	
9	9	9	3	
10	9	10	9	
11	4	11	4	
12	0	12	1	

Entonces, ya que $Z_{13} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, podemos hallar los puntos que conforman a esta curva elíptica completando la tabla 3.

Ahora, el gráfico de los puntos resultantes sería se pueden observar en la figura 14:

Figura 14: *Curva elíptica en cuerpo finito*



Fuente: Creado en (Grau, 2011).

Todas las operaciones estudiadas para curvas elípticas para los números reales, ahora se convertirán en operaciones sobre puntos empleando aritmética modular.

Por su parte, el discriminante de la ecuación 2 pasa a ser:

$$(4a^3+27b^2)\neq 0 \pmod q$$

Ecuación 13: *Discriminante módulo q.*

Las ecuaciones 6 para hallar las coordinadas del punto R final en una suma de puntos, pasan a ser:

$$x_R=\lambda^2-2x_P \pmod q,\ y_R=\lambda(x_P-x_R)-y_P \pmod q$$

Ecuación 14: *Adaptación a ECC de suma de puntos P + Q.*

Por su parte, como ahora nos manejamos en cuerpos finitos con valores discretos, la división de la ecuación 5 para el cálculo de la pendiente de la recta por la que pasan *P* y *Q* cambia a:

$$\lambda = (y_q - y_p)(x_q - x_p)^{-1} (mod\ q)$$

Ecuación 15: *Pendiente de recta que pasa por P y Q en cuerpo finito.*

en donde $(x_q\text{-}x_p)^{-1}$ es el inverso multiplicativo.

Por su parte, y con el mismo razonamiento, la ecuación 7 para computar la duplicación de punto cambia a:

$$\lambda = (3x_p^2 + a)(2y_p)^{-1} (mod\ q)$$

Ecuación 16: *Pendiente de la recta tangente que pasa por P en cuerpo finito.*

Serán estas ecuaciones las que se utilizarán más adelante en la implementación de los algoritmos criptográficos en el software a desarrollar.

2.4.4. Multiplicación en ECC (*double-and-add*)

Se ha mencionado que la multiplicación de un punto *P* por un escalar *m*, es decir *m*P*, se interpreta como una sucesión de sumatorias *P + P + … + P* una cantidad *m* de veces. Sin embargo, pese a que es posible implementar la multiplicación de esta forma directa, no resultará ser la forma más eficiente de realizarla, especialmente en cuanto las dimensiones de los valores involucrados vaya aumentando. Terminará siendo una operación lenta y costosa de realizar.

Es por eso que existen otras formas de implementar la multiplicación que son más eficientes. Entre ellas, se encuentra el algoritmo *double-and-add* (duplicar y sumar) (Silverman, 2009, p. 364), el cual, toma el valor de *m* llevado a bits y realiza una serie de cálculos de tal forma que una multiplicación *m*P* no toma más que $log_2(m)$ duplicaciones de punto y $log_2(m)$ sumas de punto.

Un pseudocódigo de la misma puede encontrarse en la tabla 4.

Tabla 4: *Pseudocódigo algoritmo double-and-add*.

Algoritmo *double-and-add*	
Objetivo	Devolver el resultado de calcular n * P.
Datos de Entrada	• n: natural • P: punto sobre curva elíptica
Precondición	-
Datos de Salida	R: punto sobre curva elíptica
Postcondición	R es el resultado de hacer n * P
Código	***// Inicialización de variables*** Q, R: punto sobre curva elíptica e: arreglo de enteros i: entero ***/* Conversión de n a su expansión binaria n = e_0 + e_1 * 2 + e_2 * 2^2 + … + e_t * 2^t, donde e_0, e_1 , …, e_t pertenecen a {0, 1} y e_t = 1 */*** e = obtenerExpansionBinaria(n) ***// e = [e_0, e_1, …, e_t]*** Q = P Si e_0 = 0 entonces R ← O Sino ***// e_0 = 1*** R ← P FinSi ***// Procesamiento*** Para i desde 1 hasta t hacer Q ← 2 * Q ***// duplicación de punto*** Si e_i = 1 entonces R ← R + Q ***// suma de puntos*** finSi finPara Devolver R ***// R equivale a n * P***

Fuente: Elaboración propia basada en (Silverman, 2009).

Si bien se han encontrado otras variantes consideradas más seguras y menos propensas a distintos tipos de ataques (Blake et al., 2005), para los fines de este trabajo, se utilizará la versión más fundamental del algoritmo *double-and-add* para realizar las operaciones de multiplicación necesarias.

2.4.5. Parámetros de dominio en ECC

A la hora de definir una curva elíptica a ser empleada en el marco de la criptografía, para la misma se definen lo que se denomina parámetros de dominio (D. R. L. Brown, 2010, p. 3), entre los cuales se encuentran los ya vistos:

- **q**: un entero que especifica el tamaño del campo finito F_p.
- **a**: coeficiente del término de grado uno del polinomio, que pertenece a F_p.
- **b**: término independiente del polinomio, que pertenece a F_p.

Pero además, se definen otros tres parámetros:

- **G**: punto generador de coordenadas (x_g, y_g) sobre $E(F_p)$.
- **n**: número primo que funciona como el orden de *G* (el orden de un punto es el número que multiplicando *G* por dicho número, obtendremos el punto en el infinito *O*).
- **h**: llamado cofactor, equivale a la cantidad de elementos en $E(F_p)/n$.

Estos seis parámetros se definen para todas las curvas que se emplean en el mundo real, algunas de las cuales serán mencionadas con ejemplos en la sección 2.4.7. Los mismos son públicos, es decir, son visibles y accesibles por todos los participantes de la comunicación desde el primer momento ya que serán empleados en los subsecuentes cálculos que se realicen.

La relación entre *G*, *n* y *h* es importante a los efectos de comprender la relación que hay entre la formación de subgrupos, los puntos dentro de ellos, la selección de un punto generador, la multiplicidad del cofactor y por qué algunas curvas se más vulnerables que otras. Sin embargo, su estudio queda fuera del alcance del presente trabajo, y amerita pensar en una o varias funcionalidades específicas a futuro para su incorporación dentro de la plataforma que se busca desarrollar.

2.4.6. Problema del Logaritmo Discreto en Curva Elíptica (*ECDLP*)

Para que los esquemas criptográficos asimétricos o de clave pública funcionen, es necesario que estos realicen operaciones matemáticas que sean fáciles de computar (complejidad polinomial) en un sentido disponiendo de todos los valores necesarios, y computacionalmente inabordables (complejidad exponencial) si se quisiera revertir su efecto no contando con todos dichos valores.

En el caso de los algoritmos tradicionales como RSA y Diffie-Hellman, tales problemas matemáticos son la factorización de números enteros o el problema del logaritmo discreto, ya mencionados. Ahora bien, en el mundo de las curvas elípticas, la versión de este problema de gran dificultad computacional es llamado Problema del Logaritmo Discreto en Curva Elíptica (*Elliptic-curve Discrete Logarithm Problem – ECDLP*) (N. Koblitz, s. f., p. 206).

Este consiste en que, dada una curva elíptica *E* y dos puntos *T* y *S* pertenecientes a *E* tal que $T = m*S$, resulta que no se ha hallado una forma sencilla o trivial de encontrar el valor de *m*, para valores de *m* muy grandes, para todo tipo de curva elíptica *E* en general. Existen distintas técnicas que se pueden intentar aplicar para encontrarlo, aunque no necesariamente son eficientes (Musson, s. f., p. 54).

Esto implica que en los distintos esquemas de cifra con curva elíptica, el valor de *m* funciona como clave privada y $m*S$ como la clave pública de un usuario. Alguien que tenga $m*S$ no podrá hallar *m* de forma sencilla. El esfuerzo computacional y el tiempo necesarios para hallar *m* es tan grande, que un atacante desistirá de siquiera intentarlo. Por ejemplo, supongamos que tenemos la curva elíptica $E = x^3 + 2x + 3 \ (mod\ 13)$ vista en la sección 2.4.3, y queremos montar un esquema criptográfico asimétrico. Para ello, se establece un llamado punto generador como, por ejemplo, $S = (4, 7)$, el cual pertenece a la curva elíptica *E* y que es visible y accesible para todos los participantes en la comunicación desde un principio (*S* no es un valor secreto, sino que es dado como parámetro de dominio junto a *E*).

Entonces, un usuario llamado Pedro, decide participar en este esquema, motivo por el cual debe poseer una clave privada y una pública. Así, elige un valor escalar $m = 5$, que funcionará como su clave privada secreta. A continuación, calcula el producto $T = m*S$ y arriba al valor *T*

= *(7, 10)* la cual también pertenece a la curva elíptica *E*. Este punto *T*, será su clave pública (se puede utilizar alguna de sus dos coordenadas, como por ejemplo T_x).

Como *T* conforma su clave pública, la cual es accesible por todos, y *T = m*S*, entonces un atacante podría buscar hacerse con *m*, que es la clave privada de Pedro. Para ello, el agente malicioso podría intentar ir probando uno por uno todos los valores posibles de *m*S* hasta, tal vez, llegar a encontrar el punto resultante *T* (ver tabla 5):

Tabla 5: *Búsqueda de clave privada en ECDLP.*

m	S	m*S
1	(4,7)	(4, 7)
2	(4,7)	(9, 3)
3	(4,7)	(10, 3)
4	(4,7)	(11, 2)
5	(4,7)	**(7, 10)**

Como suele suceder en los problemas matemáticos de esta naturaleza, en el ejemplo plasmado en la tabla 5 se logró llegar con éxito al punto *T* con facilidad en pocos pasos, porque se estableció una clave de dimensión pequeña, y porque la curva elíptica es de un grado también muy acotado. Si en vez de esto se hubiesen tomado valores iniciales más grandes, el problema del logaritmo discreto de curva elíptica se hubiese manifestado plenamente, y un abordaje como el aquí intentado no hubiese dado fruto. No se habría podido llegar al punto *T* fácilmente, lo cual quiere decir que encontrar el valor *m* (clave secreta de Pedro) es difícil.

Ejemplos de curvas elípticas que se consideran seguras y que se usan en la criptografía son, por ejemplo, las del NIST, algunas de las cuales se mencionan en la sección 2.4.7.

Ahora bien, es también importante aclarar que, como se mencionó en la sección 2.4.4, el algoritmo *double-and-add* es eficiente para realizar la operación de multiplicación *m*S*, pero hay que tener en cuenta que cuando se realiza esta operación, tanto el valor de *m* como *S* son ambos conocidos. La dificultad del ECDLP radica en justamente no poder hallar fácilmente *m* partiendo del resultado *m*S*.

2.4.7. Curvas usadas en la industria

Con motivo de fomentar la existencia de un marco común de entendimiento entre las distintas organizaciones que tengan que implementar mecanismos de seguridad, se han creado distintos estándares definiendo curvas elípticas entre las cuales poder elegir para operar. De esta forma, se busca mayor interoperatividad entre los agentes participantes.

Algunas son más usadas que otras debido a su rendimiento y nivel de seguridad (Shaikh et al., 2017). Se analizan características matemáticas más avanzadas y precisas para cada una de ellas, principalmente para detectar si son más o menos susceptibles a que el ECDLP sea vulnerado en ellas, producto de alguna falla o defecto en la elección de sus parámetros de dominio.

Tabla 6: *Ejemplos de parámetros de dominio en curvas NIST.*

secp192r1	
q	0xfffffffffffffffffffffffffffffffeffffffffffffff
a	0xfffffffffffffffffffffffffffffffefffffffffffffc
b	0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1
G	(0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012,0x07192b95ffc8da786310 11ed6b24cdd573f977a11e794811)
n	0xffffffffffffffffffffff99def836146bc9b1b4d22831
h	0x1
secp256r1	
q	0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff
a	0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc
b	0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
G	(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296, 0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5)
n	0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551
h	0x1

Fuente: extraído de (D. R. L. Brown, 2010).

Por ejemplo, el Instituto Nacional de Normas y Tecnología (*NIST*), que pertenece al Departamento de Comercio de los Estados Unidos, ha publicado un *Standards for Efficient Cryptography 2 (SEC2*), en donde se recomiendan un listado de curvas elípticas específicas para su uso (D. R. L. Brown, 2010, p. 2).

Cada curva elíptica propuesta por el NIST posee un identificador, y tiene asociado un conjunto de valores de parámetros de dominio (ejemplos en la tabla 6).

Los valores de estas curvas son definidos verificadamente al azar mediante el uso de una semilla del algoritmo de hashing SHA-1, especificado en ANSI X9.62. Esto sucede de esta forma por dos motivos: evitar que atacantes puedan encontrar algún patrón que los lleve a poder realizar un ataque, como así también evitar la desconfianza para con aquellos que establecieron los valores de las curvas NIST, y no suponer que en dichos valores se encuentra escondida alguna vulnerabilidad o trampa que sus creadores pudieran haber dejado allí deliberadamente plantada para explotar llegado el caso.

Además, es importante mencionar que, al momento de generar estos valores de dominio, se tuvieron en cuenta distintas consideraciones para que dichos valores fueran lo más óptimos a la hora de aplicarse las operaciones de cifrado de curvas elípticas en cuestión. Por ejemplo, existen otros métodos más eficientes de calcular la multiplicación de un punto con un número escalar sobre una curva elíptica por los explicados en la presentación del presente trabajo. Sin embargo, su estudio y análisis quedan fuera del alcance de éste.

Por ejemplo, se sabe que la curva secp256k1 no cumple con todas las propiedades deseadas de una curva elíptica segura, y sin embargo se utiliza en el protocolo de Bitcoin al considerarse que dichas falencias no son de relevancia en el uso que se le da a dicha curva elíptica dentro de la lógica de la mencionada criptomoneda. Por su parte, la curva secp256r1 es considerada plenamente segura, y se recomienda su utilización en un contexto que requiera de criptografía ligera.

2.5. Eficiencia

La razón por la cual las curvas elípticas han ido tomando cada vez más importancia a lo largo de los últimos tiempos, se basa en la seguridad alcanzada en base a la dimensión en bits de sus claves en comparación a otros algoritmos (ver tabla 7).

Tabla 7: *Comparación de longitud de claves según esfuerzo computacional.*

Dimensión de clave simétrica en bits	Dimensión clave RSA en bits	Dimensión clave ECC en bits
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Fuente: Extraído de (Kultinov, 2019)

Aquí se puede observar que si tenemos una clave RSA (otro algoritmo de tipo asimétrico) de 1024 bits, con la criptografía de curva elíptica (ECC) se obtiene el mismo nivel de seguridad pero con una clave de 160 bits, es decir una clave mucho más chica.

Mientras el tamaño de las claves RSA deben ir creciendo en forma exponencial para incrementar su seguridad, las claves de curva elíptica lo hacen en proporción lineal. Esto hace que, considerando el avance de la tecnología y particularmente la creciente capacidad de cómputo de los dispositivos y equipos, es necesario ir aumentando el tamaño de las claves a emplear para asegurar que las comunicaciones sean seguras. En este contexto, la proyección futuro en el incremento del uso de la ECC en contraste a RSA es muy favorable a la primera.

Por ejemplo, en el caso de los establecimientos de comunicaciones seguras SSL/TLS por medio de la entrega de certificados de autoridades de certificación (CA), muchos de estos cómputos se realizan en el navegador del usuario, motivo por el cual es importante que las mismas sean lo más rápidas posibles (Barker, 2016).

Incluso el NIST, considerando las necesidades de entornos seguros para distintas áreas emergentes como el Internet de las cosas (*Internet of Things - IoT*), sistemas de control

distribuido, redes de sensores, industria de la salud, etc., ha iniciado un proceso de concurso para evaluar la regulación de algoritmos criptográficos ligeros que se puedan aplicar a estos ámbitos (Sonmez Turan et al., 2021). En línea con esto, van surgiendo cada vez más estudios que analizan la factibilidad de aplicar curvas elípticas en dichos contextos, en lo que es un proceso en pleno proceso de investigación y cambio (Lara-Nino et al., 2018).

2.6. Herramientas y aplicaciones existentes

Existen una variedad de herramientas y aplicaciones con fines educativos que atienden a la problemática de acercar el estudio de las técnicas criptográficas a estudiantes de criptografía por medio de elementos visuales e interactivos. A continuación se observarán las características de las más destacadas para evaluar los puntos a mejorar o directamente incorporar de cara a la creación de una aplicación educativa especialmente enfocada en el estudio de la criptografía de curva elíptica.

2.6.1. GenRSA

Esta aplicación de D. Rodrigo Díaz Arroyo se centra en su totalidad en el estudio del algoritmo RSA (Díaz Arroyo, 2018), y aunque el interés del presente trabajo radica en el estudio de las curvas elípticas, genRSA posee ciertas cualidades de interés para el desarrollo del software educativo que aquí se busca desarrollar.

Actualmente se encuentra en la versión 2.1 con fecha 30 de Junio de 2018.

Desarrollado en Java y JavaFX, genRSA cuenta con una amplia y completa gama de pantallas y pruebas de características propias del algoritmo RSA que facilitan la asimilación y aprendizaje del mismo por parte de un estudiante de criptografía. Entre ellas encontramos:

- Pantalla dedicada a la generación de claves pública y privada RSA pudiendo observar en detalle la composición y propiedades de todos los números involucrados en el mismo.

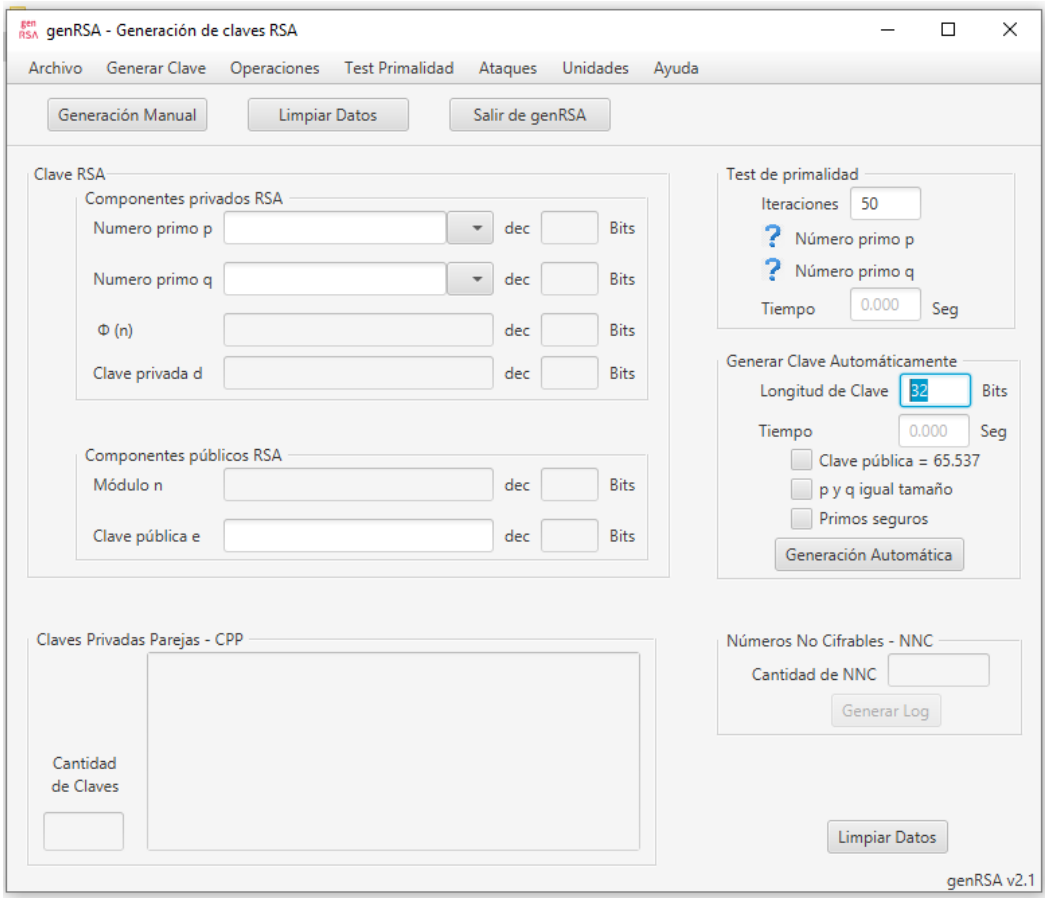
- Cifrado y descifrado de datos empleando claves RSA y pudiendo observar todos los parámetros de prueba involucrados, y que el usuario puede introducir por pantalla.

- Pantalla exclusiva para el análisis del proceso de firmado y validación de firma de datos con RSA.

- Pantallas en donde se pueden estudiar ataques que afectan particularmente al algoritmo RSA (paradoja del cumpleaños, ataque cíclico y ataque por factorización) pudiendo observar las vulnerabilidades expuestas por cada uno.

- Pantallas y apartados en donde se pueden analizar ciertas propiedades matemáticas de ciertos componentes que se utilizan en alguna parte del cifrado con RSA.

Figura 15: *Menú principal y generación de claves – genRSA.*



Fuente: Extraído de (Díaz Arroyo, 2018).

Es interesante observar que en muchas de estas pantallas, el programa también posee la posibilidad de autocompletar o rellenar ciertos campos en caso de que el usuario así lo

desee, facilitando así la comprensión de lo que está sucediendo por parte del estudiante ante la duda de no saber o comprender completamente qué valores debería introducir en todos los campos desplegados ante la posibilidad de que no disponga de la información concreta y correcta para rellenar todos ellos.

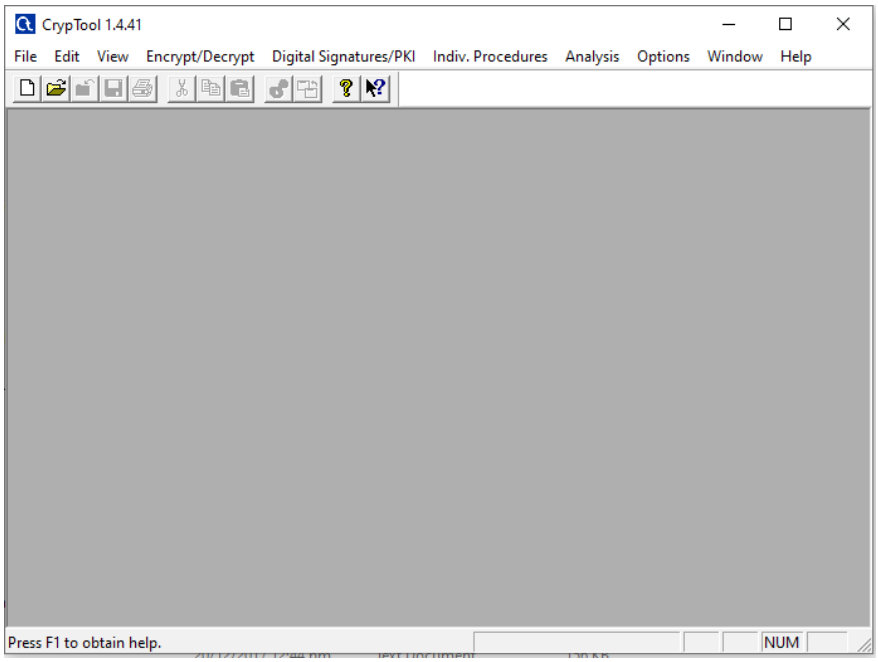
2.6.2. CrypTool

La herramienta CrypTool (*Cryptool*, s. f.) posee dos versiones distintas (v1.0 y v2.0) con diferentes funcionalidades cada uno, pero con la característica de que se trata de una aplicación que posee apartados destinados a toda una amplia suite de algoritmos de todo tipo (cifrado simétrico, cifrado asimétrico, funciones hash, esteganografía, etc.). Se trata de una aplicación integradora de la mayoría de las ramas de la criptografía (ver figura 16).

Esta herramienta incorpora una pantalla en la que se puede estudiar de forma visual cómo se va desarrollando el paso a paso en el protocolo de Diffie-Hellman tradicional (ver figura 18) indicando con verde aquellos campos que se van completando exitosamente, y con rojo los que aún falta completar, brindando la posibilidad de llevar a cabo un rellenado automático si el usuario así lo desea para facilitar la tarea del usuario. Esto es una idea útil a incorporar de cara al desarrollo de una aplicación cuyo fin sea el de ayudar a un alumno a asimilar la aplicación de las curvas elípticas a los algoritmos criptográficos existentes. Es pedagógicamente positivo inclinar el desarrollo de tal aplicación hacia dicho rumbo.

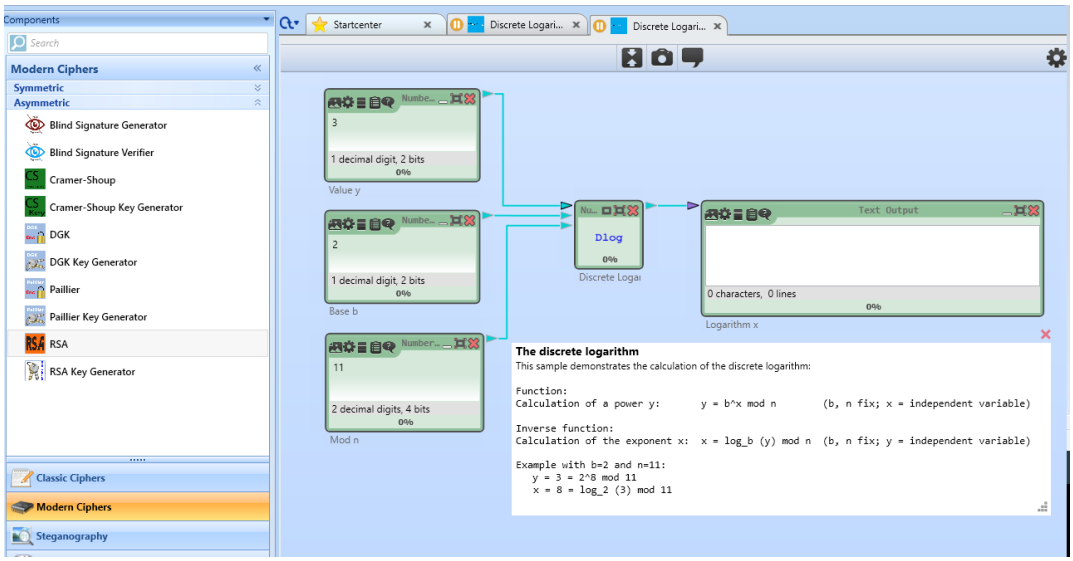
Incluso se observa que se ha destinado una pantalla específicamente al estudio del problema del logaritmo discreto (ver figura 17), el cual es la base de la fortaleza del algoritmo de Diffie-Hellman, con lo cual amerita que dicho tema posea una pantalla dedicada exclusivamente a su estudio y análisis.

Figura 16: *Menú principal - CrypTool v1.0.*



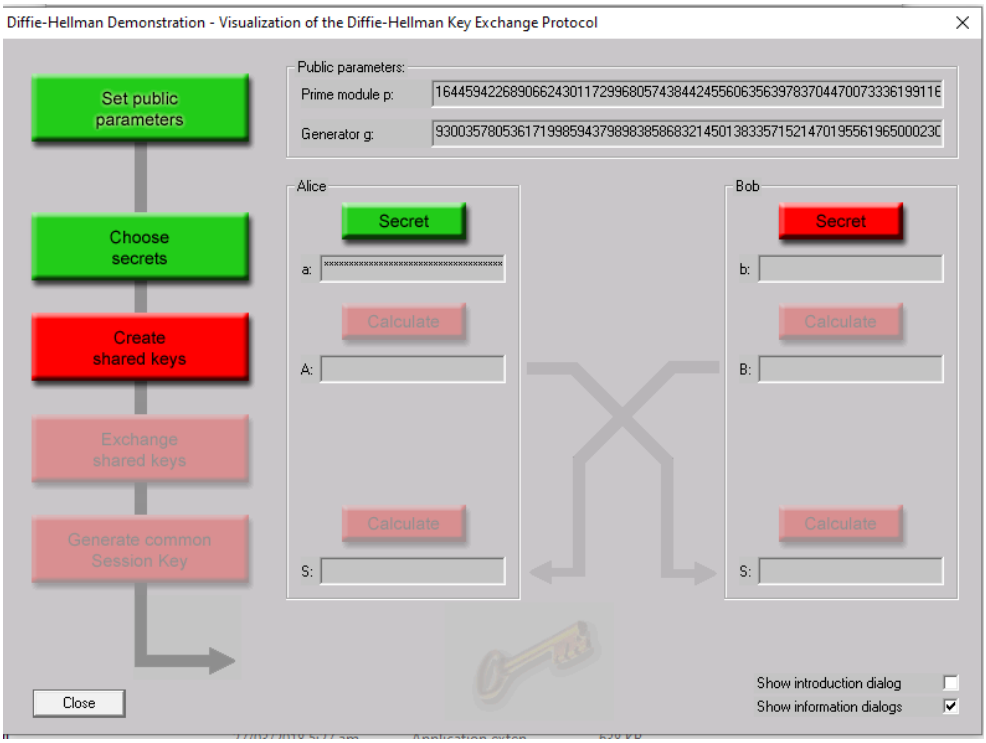
Fuente: Extraído de (Cryptool, s.f.).

Figura 17: *Logaritmo discreto en RSA - CrypTool v2.0.*



Fuente: Extraído de (Cryptool, s.f.).

Figura 18: *Demostración de Diffie-Hellman RSA - CrypTool v2.0.*

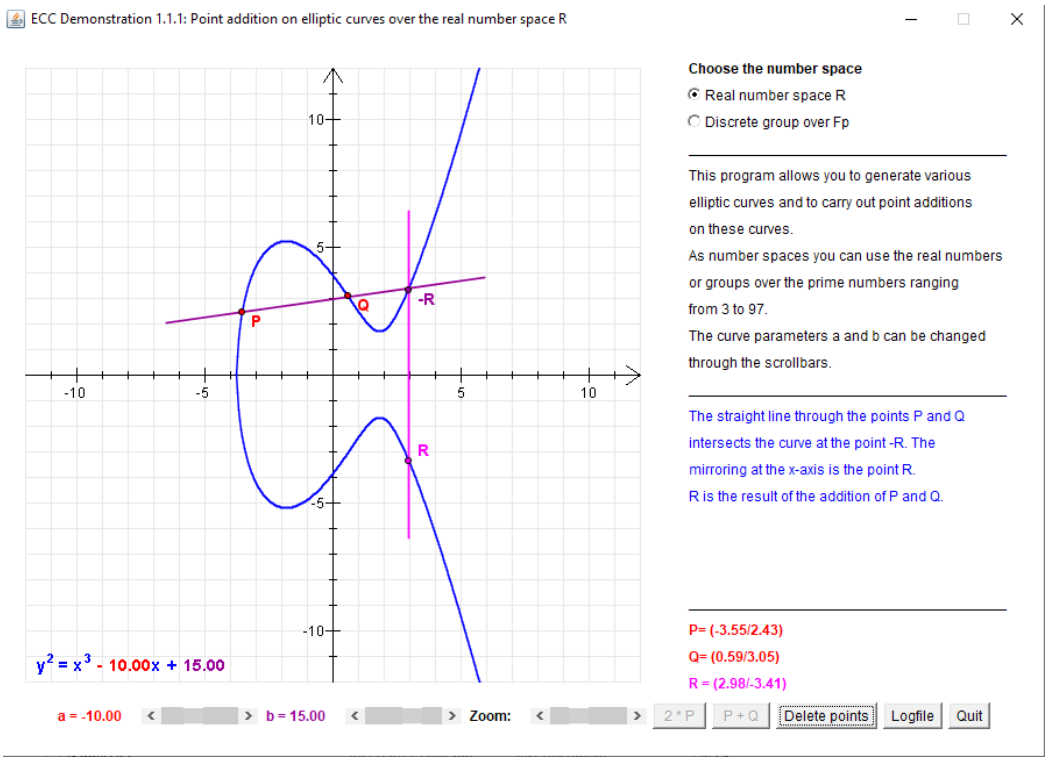


Fuente: Creado por (Cryptool, s.f.).

Ahora bien, en esta aplicación es posible estudiar de forma visual las operaciones de suma y multiplicación de curvas elípticas de forma gráfica e interactiva. El usuario crea una curva elíptica modificando a mano los valores de las constantes *a* y *b* en la ecuación general de la cuerva. La graficación de la misma se da tanto en el espacio de los números reales como del grupo discreto sobre F_p (campo finito con p elementos primos), es decir graficando una curva propiamente dicha o un conjunto de puntos módulo *p*. De esta forma, una vez graficada la curva elíptica en cuestión, con el mouse se eligen dos puntos *P* y *Q* sobre la curva, y a continuación se traza una recta a través de ambos e interceptando a la curva en otro punto -*R*, sobre el cual luego se busca el punto espejo que finalmente se llama *R*, cuyo resultado se considera el resultado del a operación *P* + *Q*.

No se encuentran representaciones para los algoritmos de Diffie-Hellman o DSA en sus versiones con curvas elípticas.

Figura 19: *Suma de puntos en curvas elípticas - CrypTool v2.0.*



Fuente: Extraído de (Cryptool, s.f.).

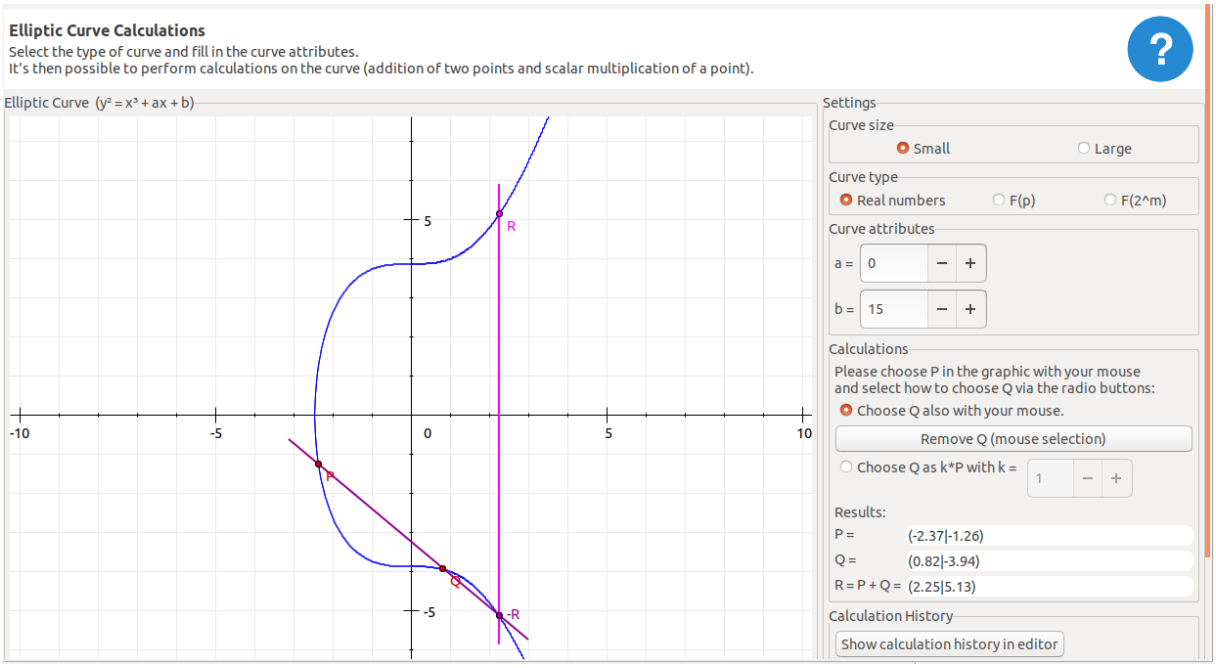
2.6.3. JCrypTool

Se trata de otra plataforma de CrypTool (*JCryptool*, s. f.), programada en Java, y que puede ejecutarse en Linux, macOS y Windows. Si bien su foco radica en que aporta algoritmos de cifra post-cuánticos, esta versión incluye, en lo que respecta a curvas elípticas, una pantalla de análisis visual con operaciones de suma y resta (tanto en el campo Real como finito); una pantalla dedicada a Diffie-Hellman para curvas elípticas; e incluso una pantalla sobre firma digital con curvas elípticas.

En la pantalla de la figura 22 se puede apreciar que es posible graficar e interactuar con una curva elíptica de dimensiones y atributos acotados, tanto sobre los números naturales como así también en un cuerpo finito, viendo el resultado de realizar operaciones de suma y multiplicación. Además, todas las interacciones y operaciones quedan registradas en un historial que el usuario puede llegar a extraer en un archivo si así deseara hacerlo.

Además, en esta pantalla existe una opción por medio de la cual es posible trabajar con curvas reales con atributos predefinidas con características similares a las de la sección 2.4.7. Si bien no se se visualizan las mismas debido a la cantidad de puntos en cuestión, sí es posible probar la suma y la multiplicación entre distintos puntos P y Q. Incluso se puede elegir que el software solo utilice puntos al azar para realizar estas operaciones.

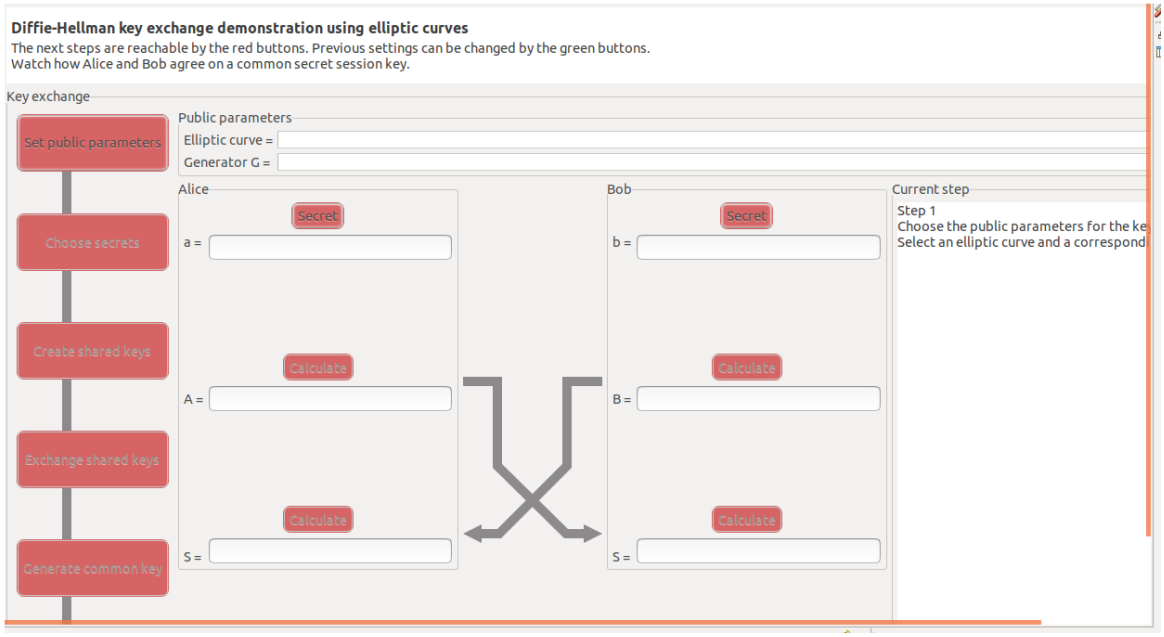
Figura 20: *Análisis gráfico – JCrypTool.*



Fuente: Extraído de (JCryptool, s.f.).

En lo que respecta a ECDH (figura 23), se trata de una pantalla en la que se van seleccionando, paso a paso, los valores de los parámetros y claves involucradas en este protocolo de intercambio de clave, hasta llegar a que ambos participantes de la comunicación arriben por su cuenta a la misma clave secreta final.

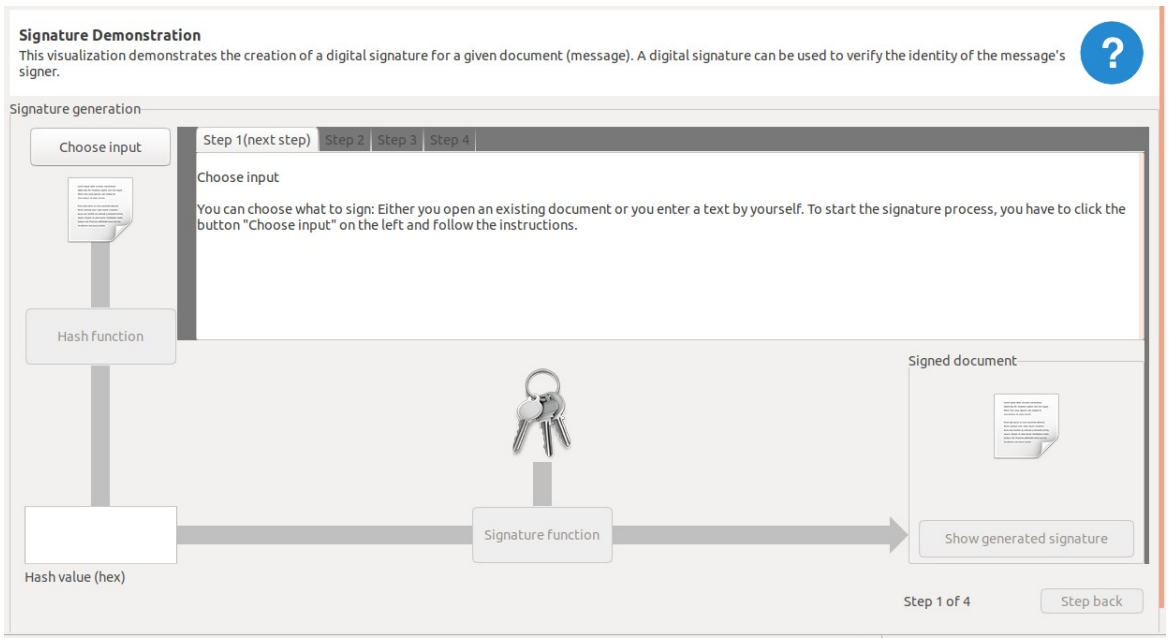
Figura 21: *Interfaz de ECDH – JCrypTool.*



Fuente: Extraído de (JCryptool, s.f.).

Por su parte, existe una pantalla en la que se puede hacer un paso a paso completo para ver el proceso de firma digital (figura 24) para un dato de entrada dado, en donde se nos permite operar con curvas elípticas y funciones hash.

Figura 22: *Interfaz de firma digital – JCrypTool.*



Fuente: Extraído de (JCryptool, s.f.).

No se observa que esta herramienta cuente con alguna pantalla en la que se analicen las propiedades abelianas de una curva elíptica.

2.6.4. SageMath

La aplicación SageMath (*SageMath*, s. f.) es una aplicación construida principalmente sobre Python para estudios de matemática pura o aplicada, yendo de funcionalidad básicas hasta avanzadas. Si bien es posible realizar gráficos e interactuar con una cierta interfaz web, está principalmente pensado para ser utilizado por consola para realizar cálculos avanzados.

En lo que respecta a curvas elípticas, incluye comandos para realizar operaciones sobre puntos, como así también operaciones sobre anillos generales campos generales o campos finitos.

Figura 23: *Suma de puntos – SageMath.*

```
sage: E = EllipticCurve('389a1')
sage: P = E(-1,1); P
(-1 : 1 : 1)
sage: Q = E(0,-1); Q
(0 : -1 : 1)
sage: P+Q
(4 : 8 : 1)
sage: P-Q
(1 : 0 : 1)
sage: 3*P-5*Q
(328/361 : -2800/6859 : 1)
```

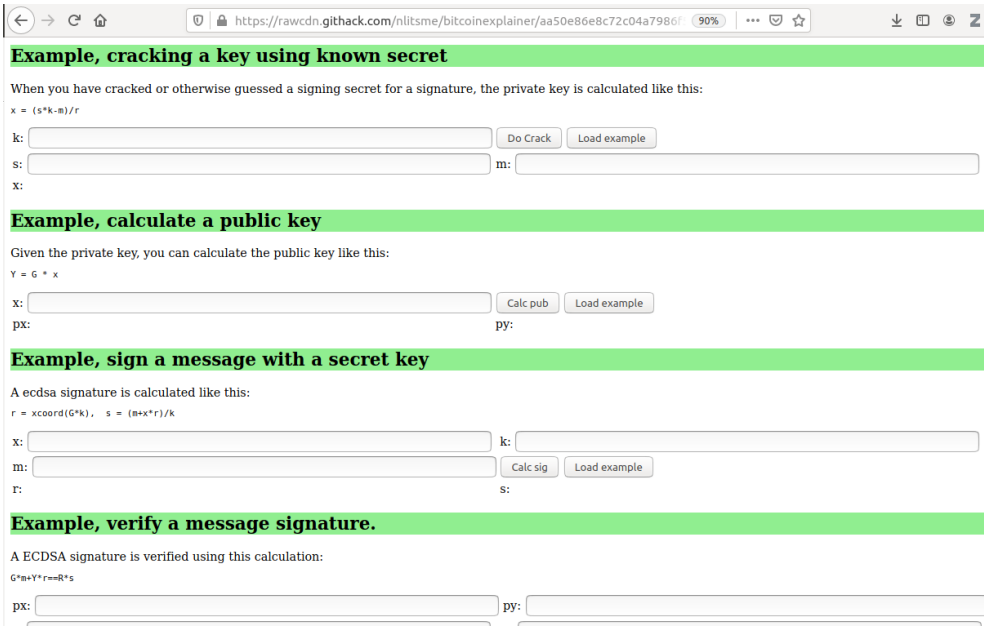
Fuente: Extraído de (SageMath, s.f.).

2.6.5. bitcoinexplainer

El repositorio Github bitcoinexplainer (*bitcoin explainer*, s. f.) está orientado a la comprensión pedagógica de las operaciones de curva elíptica en el contexto de las operaciones de bitcoin. Es una aplicación desarrollada en Javascript y que corre enteramente en el navegador del cliente.

Aquí se puede encontrar código que implementa las operaciones de puntos con curvas elípticas, como así también su aplicación a las operaciones de firma digital con curvas elípticas. Incluye una pantalla en donde es posible rellenar unos formularios con distintos parámetros para probar los resultados de operaciones propias de firma digital (figura 24) como crackeo de clave reutilizando un secreto; crackeo de clave utilizando secreto conocido; cálculo de una clave pública; firma de un mensaje con clave secreta; verificación de firma de mensaje; suma de puntos; multiplicación de puntos; etc.

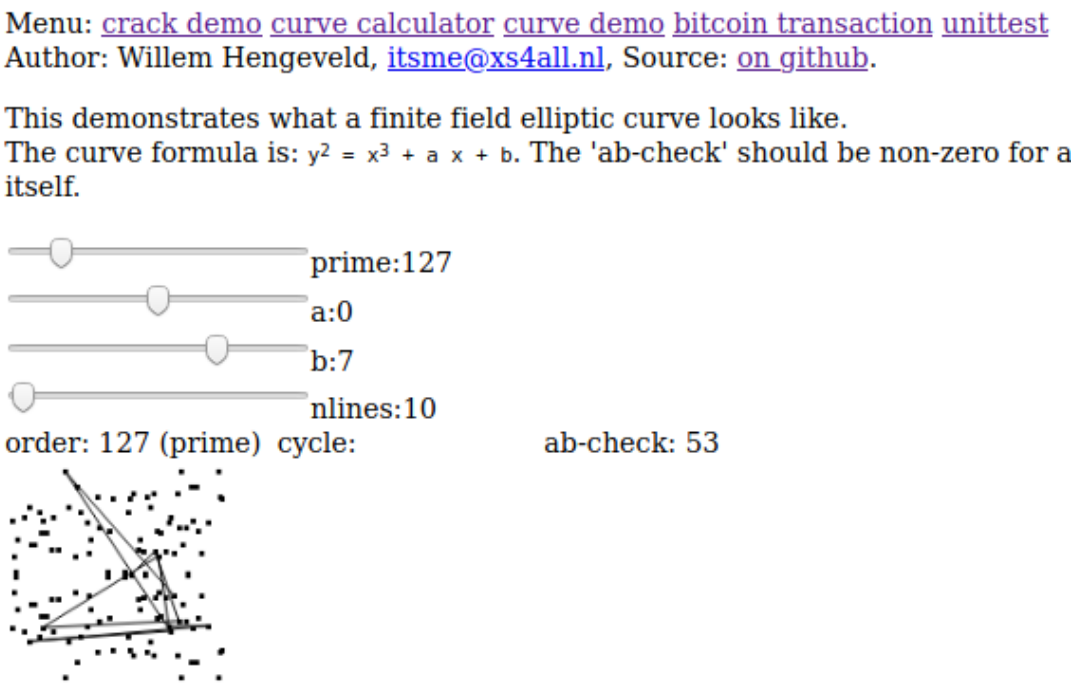
Figura 24: *Pruebas con firma digital con curva elíptica – bitcoinexplainer.*



Fuente: Extraído de (bitcoin explainer, s.f.).

Se destaca además la existencia de una pantalla en donde se observa una interpretación geométrica de las curvas elípticas (figura 25), pero en este caso para un campo finito, lo cual es distinto a la interpretación geométrica tradicional en R que es de más fácil asimilación para un estudiante de criptografía.

Figura 25: *Representación de curva elíptica en cuerpo finito – bitcoinexplainer.*



Fuente: Extraído de (bitcoin explainer, s.f.).

2.7. Conclusiones del Estado del Arte

Al relevar las principales herramientas existentes, se lleva a cabo una comparación entre ellas en base a las distintas funcionalidades que las mismas exponen (ver tabla 8).

Se observa que si bien existen diferentes herramientas disponibles para el estudio de curvas elípticas, se registra una carencia de una herramienta que esté totalmente dedicada a la comprensión de curvas elípticas en el mismo sentido de cómo está concebido genRSA para el algoritmo RSA.

Tabla 8: *Comparaciones entre herramientas existentes.*

	CrypTool	JCryptTool	Sagemath	Bitcoin explainer	D. Aglawe y S. Gajbhiye
Exclusivo para ECC				X	X
Análisis de propiedades de grupo abeliano					X
Análisis gráfico	X	X			
Pantalla ECDH		X			
Pantalla ECDSA		X		X	
Análisis vulnerabilidades			X		
Trabajar con curvas del NIST		X			
Incluye soporte teórico		X			

En su mayoría, existe una variedad de aplicaciones con interfaz y sin interfaz, librerías, módulos, etc., para estudiar distintas características de las curvas elípticas, pero por lo general lo hacen de forma aislada resaltando alguna funcionalidad en particular, y no de forma cohesiva con el fin de estudiar integralmente el uso de las curvas elípticas en criptografía.

En términos generales, no se hayan funcionalidades en las que se puedan estudiar cosas tales como la comparación entre la implementación de distintos algoritmos entre sí, como por ejemplo la multiplicación en curvas elípticas (multiplicación directa vs algoritmo *double-and-add*) u otras variantes más seguras y/o más rápidas de dicha operación fundamental que es crítica para el correcto funcionamiento del esquema criptográfico planteado. Lo mismo sucede con distintos ataques cuyo funcionamiento se podría intentar plasmar en alguna pantalla para que un alumno lograra incorporar más fácilmente las vulnerabilidades a las que podrían estar expuestas las curvas elípticas y los algoritmos/protocolos que las emplean.

Por otro lado, resultado interesante mencionar que si bien la mayoría de las aplicaciones de curva elíptica poseen alguna interfaz en la cual se realizan operaciones de suma de puntos, no se muestra explícitamente qué cuentas concretas se llevaron a cabo. Teniendo en cuenta que dichas cuentas pueden variar dependiendo de si los puntos *P* y *Q* elegidos son distintos o iguales, o incluso si alguno o ambos de ellos son el punto en el infinito (*O*), puede resultarle confuso a un alumno saber qué fórmulas exactas se están llevando a cabo.

A su vez, si bien la aplicación JCryptTool invita a un abordaje más amplio de los distintos aspectos involucrados en ECC dentro de la misma aplicación, en verdad se trata de una plataforma para el estudio de la criptografía en general, y no está exclusivamente enfocada en curvas elípticas. Posee funcionalidades para numerosos algoritmos, protocolos y problemas de distintos esquemas criptográficos y subcategorías dentro de las mismas

También es observable que las herramientas en general podrían contener más apoyo teórico en cada pantalla, más allá de lo que figura en los manuales y material de consulta que cada uno posee. En algunos casos pareciera que el alumno que ejecuta estos programas ya debería estar sabiendo de antemano qué está visualizando, a dónde debe llegar, y cómo lo logrará.

3. Objetivo

Habiendo realizado una presentación de las principales propiedades de las curvas elípticas tanto en el plano teórico y práctico, como así también de las principales herramientas existentes enfocadas en su aprendizaje, en el presente capítulo se plantearán los objetivos que se buscarán alcanzar con el desarrollo de una software educativo orientado al estudio de la criptografía de curva elíptica.

En primer lugar, se plasma el objetivo general que establecerá la visión macro a conseguir, y luego se procederá a establecer objetivos específicos y concretos con los cuales lograrlo. Además, se plantea la metodología de trabajo y la organización de las distintas tareas con las cuales implementar dichos objetivos.

3.1. Objetivo General

Desarrollar una aplicación con interfaz gráfica de fines educativos orientada a estudiantes de ingeniería que estudien criptografía, enfocada totalmente en en el estudio de las curvas elípticas y su aplicación a la ECC. Se privilegiarán características tales como la interactividad, el autocompletado y poder avanzar y retroceder en cada paso, para así mejorar la asimilación de las principales propiedades de las operaciones matemáticas relacionadas a las curvas elípticas, como así también su uso en algoritmos concretos que forman parte de la criptografía de curvas elípticas.

3.2. Objetivos específicos

- **Estudiar la teoría de curva elíptica,** logrando que queden armados los cimientos básicos para la creación de una plataforma/laboratorio de aprendizaje dedicado exclusivamente al estudio integral de las propiedades de las curvas elípticas, y su aplicación a la criptografía de curva elíptica.

- **Lograr que sea comprensible para estudiantes de ingeniería**: la aplicación estará enfocada en las aplicaciones a la criptografía para curvas elípticas que estudiantes de ingeniería requerirían a la hora de hacer un curso sobre esta temática. El sistema deberá ser lo suficientemente claro como para ofrecerle al estudiante probar distintas funcionalidades en un orden que tenga sentido secuencialmente hablando (algunos temas deberían verse antes que otros por cuestiones de correlatividad de contenidos). Por ejemplo, para estudiar Diffie-Hellman, sería necesario que el usuario se topara antes con el análisis de la operación de multiplicación de puntos, la cual a su vez requiere de la comprensión de la suma de puntos en primer lugar. El alumno debe interactuar exitosamente con cada interfaz propuesta, brindándole el sistema alguna asistencia para poder avanzar y no detenerse en ningún paso.

- **Construir la aplicación de tal forma que sea interactiva**, fomentando un aprendizaje orientado a la práctica. El usuario debe tener la posibilidad de avanzar y retroceder en cada funcionalidad estudiada, para así poder ver en el momento el resultado de cambiar ciertos parámetros específicos sobre los pasos ya realizados. En la medida que se trate de curvas elípticas con parámetros de dominio no muy grandes, se buscará graficar los puntos sobre el cuerpo finito para que el usuario pueda seleccionar dichos puntos haciendo clic en los mismos.

- **Comprender y probar la operación de suma de puntos,** ya que se trata de la operación más fundamental sobre la cual se construye el resto de las propiedades y características de las curvas elípticas. Sin ella, no es posible pasar a comprender la multiplicación de puntos. En esta interfaz debería ser ser posible distinguir entre la multiplicación de dos puntos *P* y *Q* cuando ambos son distintos y cuando ambos son iguales (*point doubling*), ya que las ecuaciones cambian en ambos casos. Sería beneficioso para el estudiante poder visualizar exactamente qué ecuaciones son las que se están poniendo en juego en cada selección distinta de pares de puntos que desee probar.

- **Comprender y probar la multiplicación de puntos**, ya que se trata de la operación crítica entorno a la cual gira el problema del logaritmo discreto para curva elíptica (ECDLP). Se busca no solamente realizar la multiplicación $m * S = R$ y mostrar el

resultado en sí mismo, sino además poder comparar el tiempo de ejecución entre una implementación de multiplicación directa y una con el algoritmo *double-and-add* de mucho mayor rendimiento. En la medida que el valor de *m* aumente, la diferencia entre ambos métodos debe acentuarse más, quedando visualmente demostrado que esto es verdaderamente así.

- **Comprender y probar alguna de las propiedades de grupo abeliano de una curva elíptica**, como por ejemplo la conmutatividad en donde si tenemos dos puntos *P* y *Q*, entonces $P + Q = Q + P$. Si bien puede parecer una operación sencilla y trivial para dedicarle una interfaz propia, ya que se está acostumbrado a la conmutatividad de la matemática tradicional, es importante tener presente que la suma en curvas elípticas es de una naturaleza distinta. Es justamente por esta y otras propiedades que las curvas elípticas pueden ser usadas para implementar un esquema criptográfico: conformar un grupo abeliano en base a la operación de la suma. Es importante como tópico en sí mismo, y en un sentido más amplio para entender, integralmente, como un todo, los temas contenidos en esta plataforma/laboratorio de estudio de curvas elípticas (sin haber visto la suma de puntos antes, no se podría avanzar a este punto).

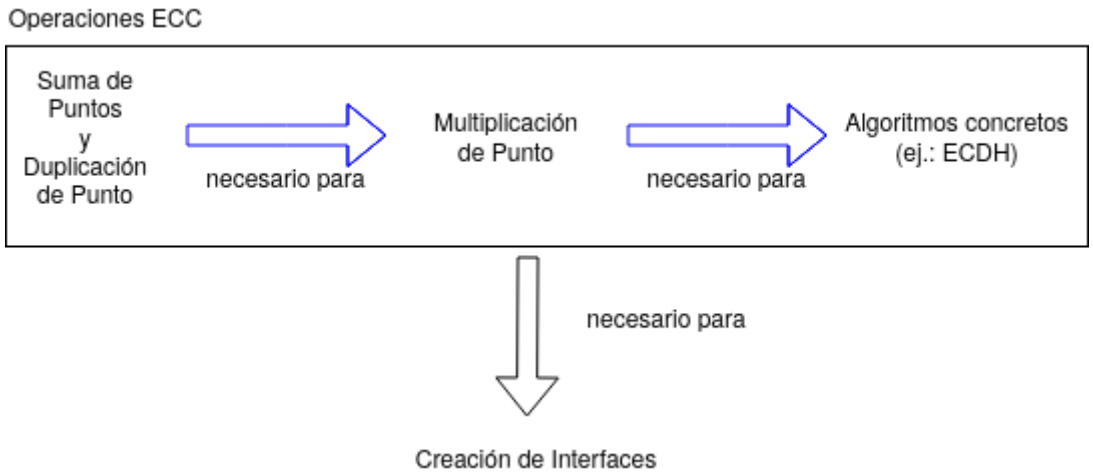
- **Comprender y probar el algoritmo de Diffie-Hellman para curva elíptica (ECDH)**, el cual sería ya un algoritmo concreto que hace uso de todos los temas anteriormente vistos (suma, multiplicación y propiedades abelianas) para dar lugar a la variante de curvas elípticas de este conocido protocolo de intercambio de claves. El usuario podrá ir viendo, paso a paso, cómo dos usuarios arriban a la misma clave secreta, partiendo tanto de una curva sencilla que decidan crear a mano, como así también eligiendo curvas predefinidas reales de estándares como del NIST o Brainpool.

3.3. Metodología de Trabajo

Teniendo en cuenta la naturaleza subyacente de esta aplicación, es importante comprender que para poder avanzar en su desarrollo, es necesario entender que existe un encadenamiento de dependencias ineludible en el plano de las operaciones ECC.

Como se observa en la figura 26, es necesario primero construir y probar las funcionalidades de suma y duplicación de puntos, para luego poder pasar a la multiplicación de puntos, y finalmente poder llevar todo eso a la implementación de un algoritmo de cifrado concreto como, por ejemplo, Diffie-Hellman con curva elíptica (ECDH).

Figura 26: *Interdependencia de los componentes ECC.*



Fuente: Creado en (Diagrams.Net, 2022).

A su vez, será necesario acompañar cada uno de estos pasos con sus correspondientes pruebas automáticas (*tests*) para estar seguros de que se está avanzando hacia un camino correcto, y no caer en un arrastre de errores de los pasos anteriores.

En consecuencia, se plantea un desarrollo paso a paso incremental con tareas secuencialmente ordenadas, tal como se muestra en el gráfico de la figura 27. Cada una de dichas tareas será descrita a continuación:

- **Configuración de entorno de desarrollo**: el lenguaje de programación elegido para trabajar es Python, y para una mejor e independiente gestión de los paquetes utilizados, se configura un entorno virtual para una más prolija gestión de las dependencias empleadas en el proyecto. Además, se configura el controlador de versiones Git, y su correspondiente repositorio en Github, que es en donde residirá el código para poder ser compartido. El versionador de código es un elemento importante para el desarrollo de una aplicación, puesto que permite registrar el historial de cambios que va sufriendo el código, permitiéndole al desarrollador tener control sobre la evolución de su código. Así, puede corregir, retroceder o crear líneas

paralelas de código sobre las cuales poder avanzar con ciertas funcionalidades, para luego decidir si quiere unirlas o no con otras ramas ya existentes.

Es necesario completar esta tarea antes de continuar con las demás.

- **Preparación del Entorno Gráfico**: para el desarrollo de la interfaz gráfica se elige trabajar con el paquete tkinter, el cual funciona con Python, y cuyos elementos conformarán las bases sobre las cuales se construirán las distintas interfaces de la aplicación. Aquí se deberá establecer el marco general lógico de la parte gráfica de la aplicación, como por ejemplo la ventana, la disposición del menú general, el funcionamiento que se le dará a los botones, etiquetas, cajas de texto y desplegables, como así también el ordenamiento de la lógica que se ejecutará mediante el desencadenamiento de los distintos eventos que el usuario generará en su interacción con las interfaces.

- **Implementación de Operaciones de Punto**: conforma la implementación concreta de la suma, duplicación y multiplicación de puntos, como así también de las dos versiones distintas que se harán para la multiplicación: método directo y *double-and-add*. Al tratarse del corazón de la lógica de toda la aplicación, se decide completar la implementación de todas las operaciones de ECC antes de proceder al desarrollo de las interfaces gráficas, para asegurar su correcto funcionamiento a nivel algorítmico y matemático. Para ello, será necesario acompañar la misma de extensas y profundas pruebas previo a su uso.

  Es necesario completar esta tarea antes de proceder al avance del desarrollo de las distintas interfaces.

- **Implementación Interfaz Suma de Puntos**: es la interfaz en la que se plasma al estudio de la suma de puntos. Esta interfaz, como el resto, debe constar de una serie de pasos secuenciales, en las cuales el usuario sea capaz de avanzar y retroceder entre cada uno. Puntualmente, esta debe ser la primera interfaz a la que se invite al usuario a ingresar, ya que es la operación más fundamental fundamental de la ECC. El usuario debe poder introducir una curva elíptica (incluyendo una opción de autocompletado), y visualmente elegir dos puntos de entre los tantos graficados en

el plano (se limitará la cantidad de puntos a una cantidad manejable por la interfaz gráfica). Entonces, no solamente se realizará la suma correspondiente, sino que deben aparecer las ecuaciones involucradas en la suma dependiendo de los puntos que el usuario haya elegido.

- **Implementación Interfaz Multiplicación de Puntos**: es la interfaz en la que se plasma el estudio de la multiplicación de un punto por un escalar. Esta interfaz, como el resto, debe constar de una serie de pasos secuenciales, en las cuales el usuario sea capaz de avanzar y retroceder entre cada paso. Aquí, el usuario debe poder introducir una curva elíptica (incluyendo una opción de autocompletado), y visualmente elegir un punto de entre los tantos graficados en el plano. A continuación, se invitará al usuario a realizar la selección de un número escalar de tal forma que pueda elegir un valor grande si así lo desea, pero no tanto como para detener al programa demasiado tiempo haciendo cálculos. El objetivo es que el usuario pueda luego ver la diferencia de tiempo empleada por el algoritmo directo en comparación al algoritmo *double-and-add*, en donde este último deberá resolverlo de forma mucho más veloz que el primero.

- **Implementación Interfaz Conmutatividad**: es la interfaz en la que se plasma el estudio de una de las propiedades abelianas, particularmente la conmutatividad. Una vez más, el usuario introduce una curva elíptica (incluyendo una opción de autocompletado), y visualmente elige dos puntos P y Q de entre los múltiples graficados en el plano. A continuación, se procede a computar, por separado, la operación $P + Q$ y $Q + P$, para corroborar que ambas deben devolver el mismo resultado.

  Al tratarse de una interfaz que utilice funcionalidad y elementos gráficos previamente utilizados, la duración en su desarrollo es menor a las anteriores.

- **Implementación Interfaz ECDH**: en esta interfaz confluyen todos los temas anteriormente vistos para reflejar el protocolo de intercambio de claves de Diffie-Hellman entre dos usuarios ficticios llamados Bob y Alice, que son justamente los dos nombre que se suelen utilizar en muchas referencias bibliográficas de la temática, lo cual también ayuda a ubicarlos contextualmente.

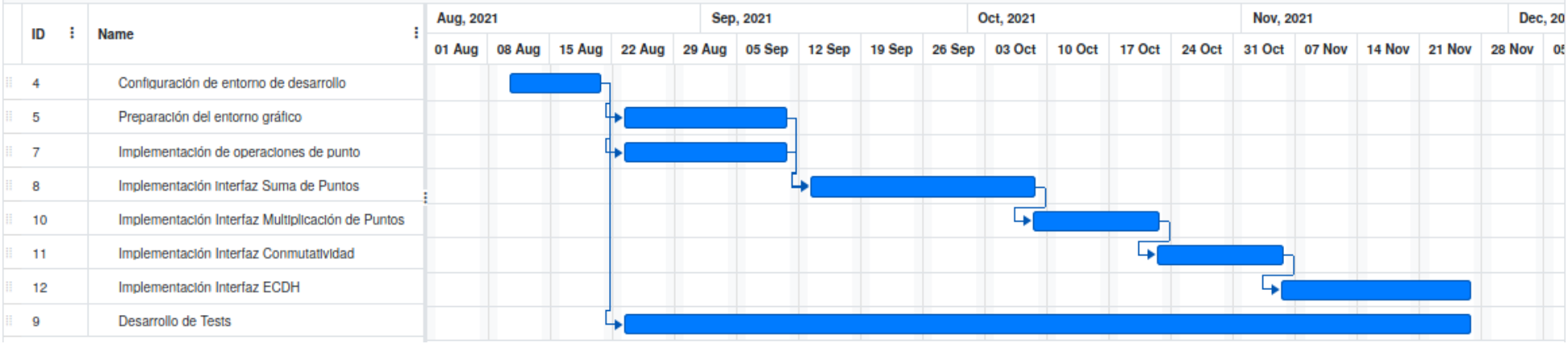
En esta caso, además de poder ingresar manualmente los parámetros de dominio para una curva elíptica, es posible seleccionar algunas curvas predefinidas de los estándares del NIST y Brainpool. Al tratarse de curvas con parámetros de gran dimensión, cuando se elija una curva predefinida, entonces no se procederá a desplegar un gráfico para elegir un punto, puesto que el punto base generador ya viene establecido en el mismo estándar para cada curva. Esto le permitirá al estudiante realizar pruebas con curvas elípticas reales.

Sin embargo, si esta fue ingresada manualmente, es decir que es de una dimensión acotada, entonces sí se procederá a elegir un punto cualquiera de un gráfico para su uso como punto generador. Cabe destacar que, por una cuestión de alcance del presente trabajo, se permitirá que cualquier punto pueda ser elegido como base generador, pese a que la teoría de curva elíptica indica que algunos puntos pueden ser mejores que otros. Esto se debe a que, en algunos casos, un punto puede generar el resto de todos los puntos de la curva elíptica, pero en otros casos no. De hecho, se busca seleccionar puntos cuyo orden sea el mayor posible para alcanzar mayores niveles de seguridad. Sin embargo, la lógica para realizar seleccionar los mejores puntos generadores no será incluido. Su correcta implementación implicaría analizar con mayor profundidad cómo explicarle y transmitirle al usuario en detalle el significado de todos los parámetros de dominio que se vinculan entre sí, es decir el orden, el cofactor e incluso el mismo valor de q.

Se asiste al usuario en el establecimiento de las claves privadas de Bob y Alice, es decir dos valores escalares, y a proceder a realizar el intercambio de valores y multiplicaciones correspondientes, para finalmente dar como resultado que ambos arriban a la misma clave secreta compartida.

- **Desarrollo de Tests**: se realizan tests unitarios para todas las operaciones de criptografía de curva elíptica. Esta tarea se extiende a lo largo de toda la línea de tiempo, puesto que todo el tiempo se irán revisando las mismas y reforzándolas con más casos y pruebas que puedan ir surgiendo a medida que se avanza en el desarrollo.

Figura 27: *Planificación de tareas - Diagrama de Gantt.*



Fuente: Creado con (Online Gantt, 2022)

4.Análisis y diseño de la solución

En este apartado se plantearán los casos de uso, requisitos funcionales y requisitos no funcionales involucrados en la confección de la aplicación que se busca construir.

En los requisitos funcionales y no funcionales se definirán las funciones y características que el sistema deberá poseer, en base a los objetivos que se han originalmente planteado.

Así, en los casos de uso se describirá, paso a paso, e interacción a interacción, el comportamiento del sistema para cada funcionalidad provista, contemplando tanto el flujo principal como los alternativos que puedan ir surgiendo.

4.1. Requisitos funcionales

Tabla 9: *Requerimiento funcional suma de puntos.*

Nro.	RF1
CU relacionado	CU1.
Nombre	Suma de Puntos en Cuerpo Finito.
Descripción	El usuario podrá interactuar con la operación suma sobre curvas elípticas en cuerpos finitos, propiedad fundamental de la teoría e implementación de curva elíptica.
Criticidad	Alta.

Tabla 10: *Requisito funcional operación de multiplicación.*

Nro.	RF2
CU relacionado	CU2.
Nombre	Multiplicación de Puntos en Cuerpo Finito.
Descripción	El usuario podrá interactuar con la operación de multiplicación de puntos sobre curvas elípticas, fundamental para las operaciones de ECC derivadas.
Criticidad	Alta.

Tabla 11: *Requisito funcional propiedad conmutativa.*

Nro.	RF3
CU relacionado	CU3.
Nombre	Propiedad Conmutativa.
Descripción	El usuario podrá llevar a adelante una breve prueba de una de las propiedades abelianas (conmutatividad) con las que cumple la suma de los puntos sobre una curva elíptica.
Criticidad	Baja.

Tabla 12: *Requisito funcional protocolo ECDH.*

Nro.	RF4
CU relacionado	CU4.
Nombre	Interfaz de ECDH.
Descripción	El usuario podrá interactuar con los pasos y parámetros necesarios para establecer una clave secreta compartida entre dos participantes, según el algoritmo de Diffie-Hellman en curva elíptica.
Criticidad	Alta.

4.2. Requisitos no funcionales

Tabla 13: *Requisito no funcional de aplicación portable.*

Nro.	RNF 1
Nombre	Aplicación portable.
Descripción	El sistema debe ser una aplicación que pueda ejecutarse en múltiples dispositivos con facilidad.
Criticidad	Alta.
Prioridad	1.

Tabla 14: *Requisito no funcional de diseño sencillo.*

Nro.	RNF 2
Nombre	Diseño sencillo.
Descripción	La aplicación debe resultar sencilla teniendo en cuenta sus fines educativos.
Criticidad	Alta.
Prioridad	1.

Tabla 15: *Requisito no funcional de fácil interacción.*

Nro.	RNF 3
Nombre	Aplicación interactiva
Descripción	La aplicación debe permitirle al usuario interactuar en cada uno de los pasos involucrados
Criticidad	Alta
Prioridad	1

4.3. Casos de uso

4.3.1. Suma de puntos en cuerpo finito

Se busca probar la suma de puntos para cuerpos finitos, desplegando un gráfico con los puntos que conforman a la curva elíptica, y permitiendo al usuario elegir los puntos sobre la misma, para luego realizar la suma correspondiente. Se busca mostrar, para cada selección de puntos, no solamente el resultado final de la suma, sino también una explicación sobre qué ecuaciones puntuales se utilizaron en la resolución de dicha operación.

Figura 28: *Caso de uso suma de puntos en cuerpo finito.*

Tabla 16: *Caso de uso de suma de puntos.*

Nro.	CU1
Nombre CU	Sumar puntos.
Descripción	Realización de la operación de suma entre dos puntos en una curva elíptica en un cuerpo finito.
Actores	Estudiante.
Precondiciones	-
Flujo de eventos	1. El usuario elige la opción “Suma” dentro de “Operaciones”.

	2. El usuario establece los parámetros de dominio para la curva elíptica con la que desea trabajar. [O1: el usuario elige la opción “Autocompletar”] 3. El sistema valida los datos y en caso exitoso habilita el siguiente paso de selección de puntos. 4. El usuario selecciona dos puntos P y Q cualesquiera sobre la curva elíptica dibujada en cuerpo finito, y cierra la ventana. 5. El sistema realiza el cálculo correspondiente según las ecuaciones de ECC que corresponda en base a los valores de P y Q seleccionados, y despliega el resultado junto a las ecuaciones que se emplearon para realizarla. 6. El usuario puede 6.1. Volver al Punto 1 y reanudar el proceso desde allí haciendo clic en “Editar”. 6.2. Volver al punto 4 y reanudar el proceso desde allí haciendo click en “Seleccionar Puntos”.
Flujos optativos	1. O1: el usuario elige la opción “Autocompletar”. 1. El sistema completa los campos de parámetro de dominio con unos valores válidos ya establecidos para poder continuar. Continúa en el punto 3.
Postcondiciones	Se calcula el resultado de R. Se grafica R. Se muestran las ecuaciones utilizadas para arribar a R. Quedarán registrados todos los pasos que tomó el usuario para llegar al resultado.

4.3.2. Multiplicación de puntos en cuerpo finito

Se busca probar la multiplicación de puntos en cuerpo finito, y poder comparar el resultado de ejecutar el algoritmo de multiplicación directa con el algoritmo *double-and-add* para observar que uno es mucho más veloz que el otro.

Figura 29: *Caso de uso multiplicación de puntos en cuerpo finito.*

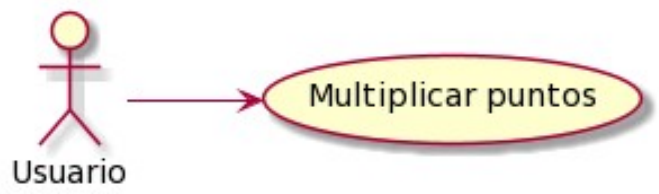


Tabla 17: *Caso de uso de multiplicación de puntos.*

Nro.	**CU2**
Nombre	Multiplicar puntos.
Descripción	Multiplicación sobre curva elíptica en cuerpo finito y comparación de rendimiento de algoritmos.
Actores	Estudiante.
Precondiciones	-
Flujo de eventos	1. El usuario elige la opción “Multiplicación” dentro de “Operaciones”. 2. El usuario establece los parámetros de dominio para la curva elíptica con la que desea trabajar. [O1: el usuario elige la opción “Autocompletar”] 3. El sistema valida los datos y en caso exitoso habilita el siguiente paso de selección de punto. 4. El usuario selecciona un punto P cualquiera sobre la curva elíptica dibujada. 5. El usuario selecciona un valor k escalar dentro de un rango establecido. 6. El usuario selecciona realizar la operación de multiplicación. 7. El sistema realiza la multiplicación por medio del algoritmo directo y double-and-add. 8. Se muestra el resultado de ambos (debe coincidir) además del tiempo que le tomo a cada uno resolver el problema. 9. El usuario puede 1. Volver al punto 2 y reanudar el proceso desde allí haciendo clic en “Editar”. 2. Volver al punto 4 y reanudar el proceso desde allí haciendo clic en “Editar”. 3. Volver al punto 6 y reanudar el proceso desde allí haciendo clic en “Editar”.
Flujos optativos	O1: el usuario elige la opción “Autocompletar”. 1. El sistema completa los campos de parámetro de dominio con unos valores válidos ya establecidos para poder continuar. Continúa en el punto 3.
Postcondiciones	Se muestra el resultado de calcular k*P. Se grafica el punto k*P . Se muestra el tiempo que se tardó en computar k*P según método directo y double-and-add. Quedarán registrados todos los pasos que tomó el usuario para llegar al resultado.

4.3.3. Propiedad Conmutativa en cuerpo finito

Se busca probar una de las propiedades de grupo abeliano que cumplen las curvas elípticas, que es la de conmutatividad, en la cual, dado dos puntos *P* y *Q* pertenecientes a una curva, entonces el resultado de hacer *P + Q* y *Q + P* debería ser el mismo punto *R* final.

Figura 30: *Caso de uso prueba conmutatividad.*

Tabla 18: *Caso de uso de propiedad de conmutatividad.*

Nro.	CU3
Nombre	Probar conmutatividad.
Descripción	Se verifica la propiedad de conmutatividad de la operación suma en una curva elíptica.
Actores	Estudiante.
Precondiciones	-
Flujo de eventos	1. El usuario elige la opción “Conmutatividad” dentro de “Propiedades”. 2. El usuario establece los parámetros de dominio para la curva elíptica con la que desea trabajar. [O1: el usuario elige la opción “Autocompletar”] 3. El sistema valida los datos y en caso exitoso habilita el siguiente paso de selección de puntos. 4. El usuario selecciona dos puntos P y Q cualesquiera sobre la curva elíptica dibujada en cuerpo finito, y cierra la ventana. 5. El sistema realiza el cálculo correspondiente según las ecuaciones de ECC que corresponda en base a los valores de P y Q seleccionados, y despliega de hacer tanto P+Q como Q+P. 6. El usuario puede 1. Volver al Punto 1 y reanudar el proceso desde allí haciendo clic en “Editar”. 2. Volver al punto 4 y reanudar el proceso desde allí haciendo clic en “Seleccionar Puntos”.
Flujos optativos	O1: el usuario elige la opción “Autocompletar”. 1. El sistema completa los campos de parámetro de dominio con unos valores válidos ya establecidos para poder continuar.

	Continúa en el punto 3.
Postcondiciones	Se corrobora que P+Q = Q+P. Se grafica el punto resultante. Quedarán registrados todos los pasos que tomó el usuario para llegar al resultado.

4.3.4. Diffie-Hellman en curva elíptica (ECDH)

Se busca probar de forma interactiva el paso a paso en el funcionamiento del protocolo de intercambio de clave de Diffie-Hellman para curva elíptica (ECDH). El usuario puede probar su funcionamiento con curvas de pequeño tamaño, como así también elegir entre curvas con parámetros de dominio predefinidos.

Figura 31: *Caso de uso analizar ECDH.*

Tabla 19: *Caso de uso de Analizador de Diffie-Hellman con curvas elípticas.*

Nro.	**CU4**
Nombre CU	Analizador de Diffie-Hellman con curvas elípticas.
Descripción	Se analizan los cálculos necesarios para ejecutar el protocolo de Diffie-Hellman utilizando curvas elípticas.
Actores	Estudiante.
Precondiciones	-
Flujo de eventos	1. El actor establece los valores de los parámetros de dominio [O1: el usuario los introduce a mano] [O2: el actor selecciona entre curvas preestablecidas] 2. El actor establece el punto generador G a emplear: [O3: el actor introduce las coordenadas del punto manualmente] [O4: las coordenadas del punto se encuentran ya completadas al haberse elegido previamente una curva preestablecida] 3. El actor establece los valores de las claves privadas y públicas para Alice y Bob. [O5: el actor introduce las claves privadas de Bob y Alice manualmente]

	[O6: el autor elige que los valores de las claves privadas se completen automáticamente] 4. El sistema realiza los cálculos correspondientes a cada uno para luego calcular la clave final a la que ambos arriba por su cuenta utilizando la combinación de valores públicos y privados correspondientes. 5. El usuario puede reanudar los pasos realizados volviendo a cualquiera de los puntos anteriores, manteniendo los datos los datos que se hayan dejado seleccionados de los puntos previos al seleccionado. Para ello hará clic en el botón "Editar" que desee.
Flujos Optativos	1. O1: el usuario introduce los valores de a, b y q de forma manual 1.1. El sistema valida los parámetros ingresados Continúa en el punto 2 1. O2: el actor selecciona entre curvas preestablecidas 1.1. El sistema despliega curvas con valores a, b y q previamente cargadas 1.2. El actor elige entre las curvas desplegadas 1.3. El sistema rellena los valores de a, b y q en base a la curva preestablecidas anteriormente seleccionadas Continúa en el punto 2 1. O3: el actor introduce las coordenadas del punto manualmente 1.1. El sistema valida las coordenadas ingresadas Continúa en el punto 3 1. O4: las coordenadas del punto se encuentran ya completadas al haberse elegido previamente una curva preestablecida 1.1. El sistema rellena los valores de las coordenadas Gx y Gy en base a la curva preestablecida anteriormente seleccionada 1.2. El actor deja los campos rellenados o cambiar el valor de G 1.3. El sistema valida las coordenadas vigentes Continúa en el punto 3 1. O5: el actor introduce las claves privadas de Bob y Alice manualmente 1.1. El actor introduce las claves privadas de Bob y Alice manualmente 1.2. El sistema valida las claves privadas ingresadas y genera las claves públicas asociadas a dichas claves privadas Continúa en el punto 4 1. O6: el autor elige que los valores de las claves privadas se completen automáticamente 1.1. El actor selecciona generar las claves privadas de forma automáticamente 1.2. El sistema autocompleta las claves privadas Continúa en el punto 4
Postcondiciones	-

5.Desarrollo de la solución

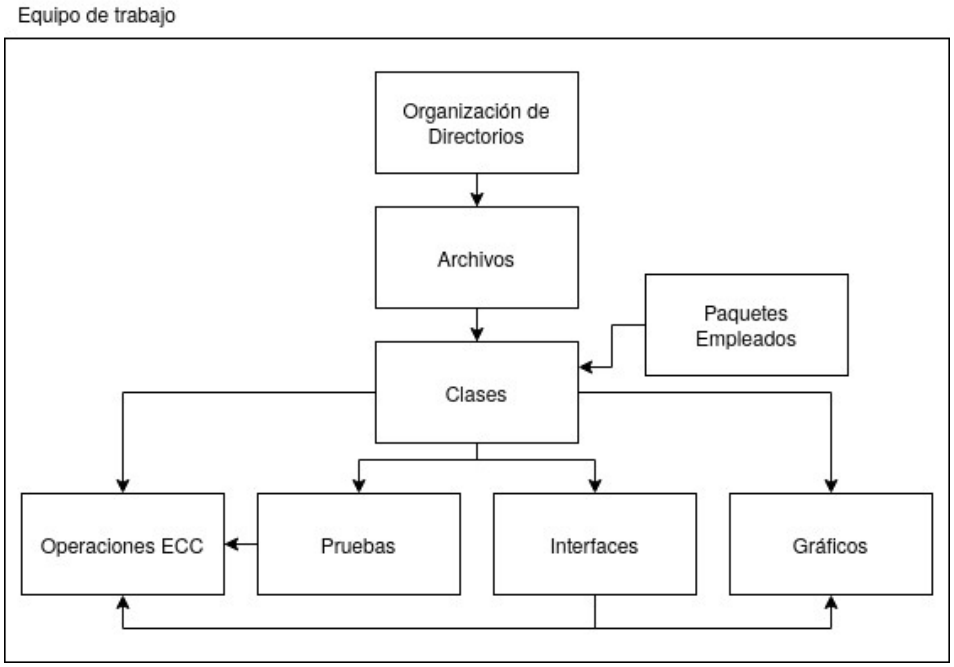
En este capítulo se procederá la describir la implementación concreta de las funcionalidades anteriormente relevadas y analizadas, en búsqueda de cumplir con los objetivos originalmente planteados.

Las instrucciones para realizar la instalación de la aplicación se pueden encontrar detalladas en el Anexo A – Guía de instalación.

5.1. Arquitectura

La aplicación será de naturaleza monolítica, con lo cual toda la lógica del programa correrá sobre un mismo programa montado en un mismo equipo.

Figura 32: *Esquema de arquitectura del proyecto.*



Como se puede observar en la figura 32, la aplicación se encuentra organizada en una serie de directorios, que contendrán archivos con objetivos diferentes cada uno de ellos.

Tales archivos serán los que contendrán la lógica propiamente dicha del programa, en forma de clases con sus correspondientes métodos. Todo lo referido a las operaciones de cifra con

curva elíptica, pruebas (*tests*), interfaces de usuario y generación de gráficos, se encuentran detallados de esta forma y vinculados entre ellos. Esto sucede de tal forma que las clases que gestionan las interfaces de usuario usarán a la clase que contiene las operaciones de ECC, que a su vez que fueran debidamente evaluadas por las clases de pruebas automáticas. A su vez, la interfaces harán uso de la clase de generación de gráficos para encargarse de mostrar los puntos de las curvas elípticas por interfaz para su interacción por parte del usuario.

Además, para que todas las clases puedan operar, será necesario incorporar en ellas distintos paquetes y módulos del lenguaje de programación elegido, los cuales ya brindan ciertas funcionalidades disponibles para su uso por parte de los desarrolladores. Esto facilita el trabajo del desarrollado, y le permite concentrarse en la construcción propiamente dicha de la aplicación en cuestión.

5.2. Lenguaje y paquetes empleados

El lenguaje seleccionado para este desarrollo es Python 3.8 debido a su ampliamente usado y confiable abanico de paquetes para operaciones matemáticas, gráfico de datos y herramientas de interfaz gráfica ya incorporadas por defecto.

```
cycler==0.10.0
ddt==1.4.4
Jinja2==3.0.1
kiwisolver==1.3.1
MarkupSafe==2.0.1
matplotlib==3.4.2
mpld3==0.5.5
numpy==1.21.1
Pillow==8.3.1
pyparsing==2.4.7
python-dateutil==2.8.2
six==1.16.0
```

Figura 33: *Dependencias del proyecto (requirements.txt)*

Si bien en la figura 33, se puede observar el contenido del archivo *requirements.txt*, el cual indica las dependencias con las que trabaja el proyecto, procederemos a describir los más

importantes de ellos a los fines del presente trabajo, incluyendo otros que no figuran explícitamente en este archivo por ya venir incluidos en la instalación del lenguaje mismo:

- **tkinter**: la aplicación está desarrollado con el paquete tkinter para construir una interfaz gráfica con la cual interactúa el usuario. Ésta incluye elementos gráficos como ser botones, cajas de texto, etiquetas, etc., para su uso.

  Dichos elementos se agrupan en marcos (*frames*), para facilitar su gestión a la hora de establecer el comportamiento deseado en las distintas interfaces.

- **Pillow**: se utiliza esta librería para desplegar algunas imágenes correspondientes a íconos a la hora de avanzar en el paso a paso de cada interfaz.

- **random**: empleada para generación de números aleatorios en la lógica de algunos puntos de la aplicación.

- **matplotlib**: con esta librería se realizan los gráficos de curva elíptica dentro de la aplicación.

- **unittest**: librería para tests unitarios, que se usarán para poner a prueba, en distintos casos, a todas las clases y métodos utilizados en la aplicación.

- **ddt**: librería de testeo que complementa a *unittest* para poder ejecutar una lista de valores precargados a cada test.

5.3. Organización de Directorios

Las carpetas y los archivos que estas contienen se organizaron en la forma que muestra la figura 34. A continuación se describe cada uno de ellos:

- ***ecc/*:** es el directorio raíz del proyecto, del cual cuelgan el resto de los demás directorios y archivos de la aplicación.

- ***main.py:*** es el archivo de inicialización de la aplicación. Establece el marco general de la aplicación sobre el cual se despliegan los elementos del resto de las interfaces, incluyendo el menú superior horizontal por el cual navega el usuario.

- ***constants.py***: es un archivo en el que se definen distintas constantes de tipo global que serán empleadas a lo largo del resto de la aplicación. La idea de esto es que si en algún momento se desea cambiar el valor de alguna de estas constantes, su efecto se propagará a todos aquellos lugares en donde se haya usado dicha constante, evitando que se tenga que ir a cada lugar y modificarlo a mano uno por uno.

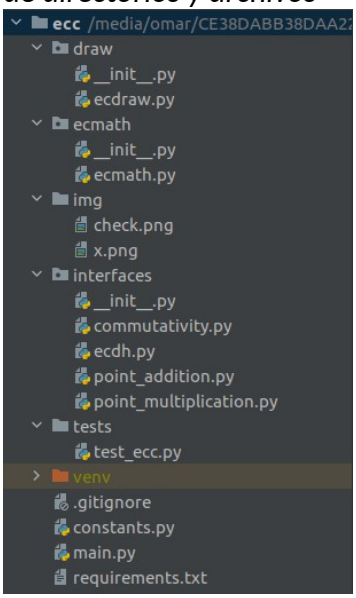
  Entre otras cosas, aquí se ubica el listado de curvas elípticas predefinidas de los estándares NIST y Brainpool que pueden luego ser elegidas en la interfaz de ECDH.

- ***ecmath/***: es el directorio que contiene el grueso de la lógica de la criptografía de curva elíptica.

  - ***ecmath.py***: es el archivo donde se ubican las clases con la lógica de las clases de curva elíptica, punto, usuario y ECDH que se utilizan en la distintas interfaces de la aplicación.

- ***img/***: contiene algunas imágenes que son usadas como íconos en la aplicación.

Figura 34: *Organización de directorios y archivos*



- ***interfaces/***: es el directorio en donde se encuentran, en archivos separados, las distintas interfaces de la aplicación.

    - ***point_addition.py***: es el archivo en donde se encuentra la clase encargada de implementar la lógica de la interfaz de suma de puntos. Utiliza las clases de *ecmath.py* para la lógica de suma de puntos sumado a las clases dentro de *ecdraw.py* para graficar los puntos en cuestión.

    - ***point_multiplication.py***: es el archivo en donde se encuentra la clase encargada de implementar la lógica de la interfaz de multiplicación de puntos. Utiliza las clases de *ecmath.py* para la lógica de la multiplicación directa y double-*and-add* de puntos. Además, utiliza *ecdraw.py* para graficar los puntos en cuestión.

    - ***ecdh.py***: archivo en donde se definen los detalles de implementación de la interfaz para Diffie-Hellman con curva elíptica. Utiliza todas las clases de *ecmath.py* y *ecdraw.py* para graficar los puntos en cuestión.

    - ***commutativity.py***: archivo en donde se se definen los detalles de implementación de la interfaz para la propiedad de conmutatividad de la operación de suma en curva elíptica. Utiliza las clases de *ecmath.py* para la lógica de suma de puntos sumado a las clases dentro de *ecdraw.py* para graficar los puntos en cuestión.

- ***tests/***: es el directorio que contiene la lógica de las pruebas automáticas a las que se someten los métodos de las clases del archivo *ecmath.py*.

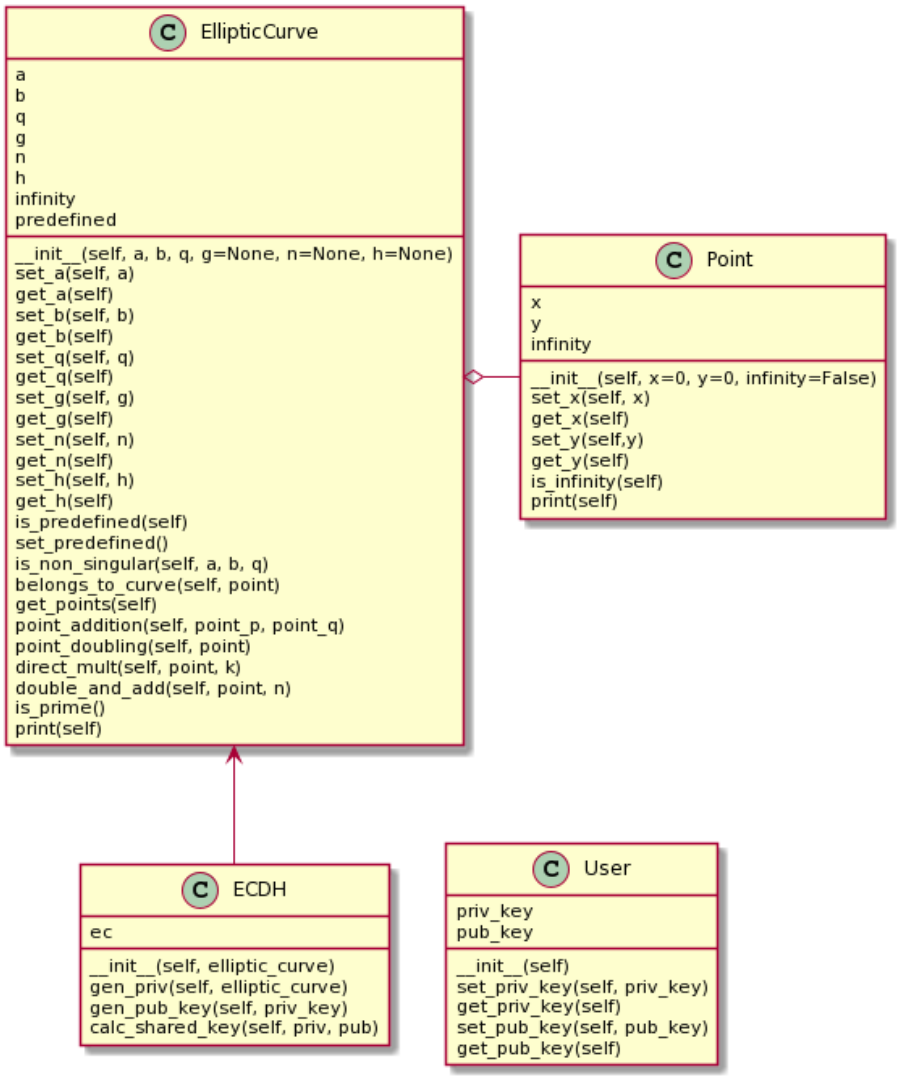
    - ***test_ecc.py***: archivo en donde ubican las pruebas unitarios (*unit tests*) para todas las clases y métodos del archivo *ecmath.py* que se utilizan luego en todas las interfaces de la aplicación (evaluación de rangos correctos de parámetros de dominio, singularidad de la curva elíptica, suma de puntos, multiplicación de puntos, conmutatividad, operaciones con curvas predefinidas y ECDH).

- ***draw/***: directorio que contiene la lógica correspondiente a la creación de gráficos en la aplicación

- ***ecdraw.py***: archivo en donde se ubican las clases correspondientes a la creación de gráficos para cada una de las interfaces. Como la lógica de cada interfaz es distinta, entonces hay clases que manejan la selección de los puntos de forma distinta a las otras (por ejemplo, no es lo mismo seleccionar un punto para luego multiplicarlo en la interfaz de multiplicación, a tener que elegir dos para la interfaz de suma, ya que en el último caso podrían darse distintos casos como elegir dos veces el mismo punto, o elegir puntos distintos, etc.).

5.4. Clases

Figura 35: *Diagrama de clases.*



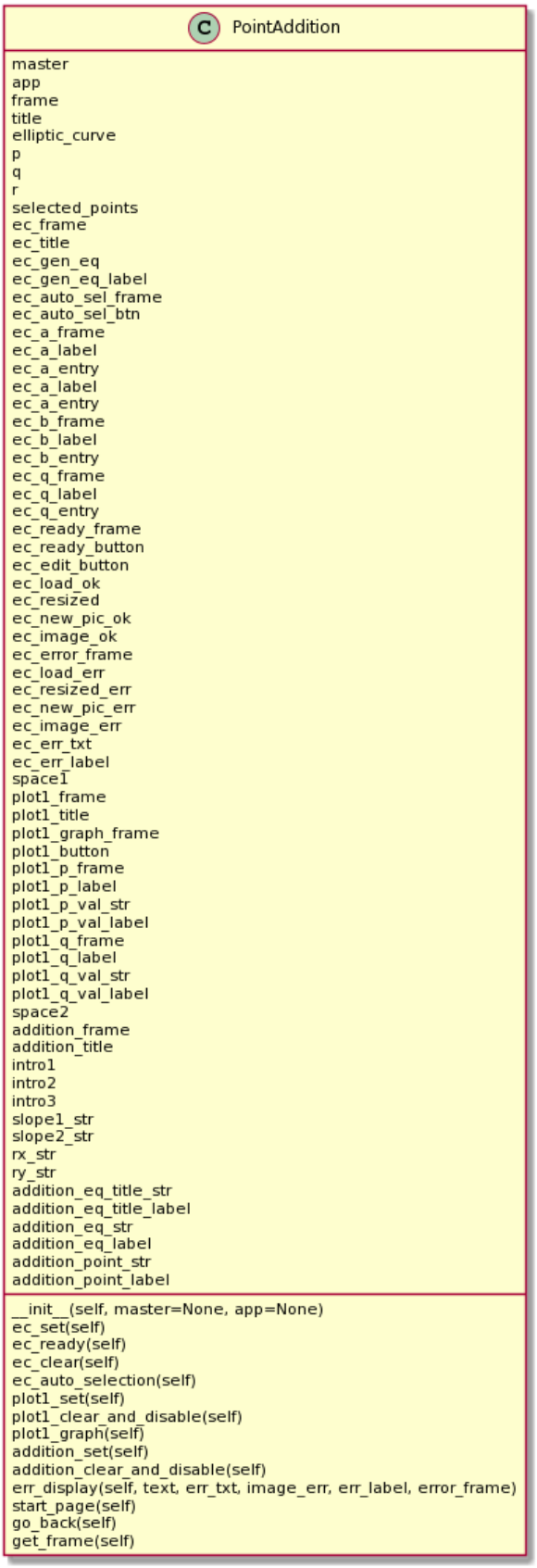
Las clases de la figura 35 son las que se utilizan en el corazón de la lógica del programa. De ellas se destacan la clase *EllipticCurve* y *Point*, la cuales son la representación de una curva elíptica y un punto, respectivamente. La clase *ECDH* para Diffie-Hellman utiliza las operaciones ya definida en los métodos de la clase *EllipticCurve*, mientras que *User* no es más que una clase que simplifica la organización y comprensión del código al asociar la noción de clave privada y pública a un usuario (simplemente realiza algunas validaciones para el protocolo ECDH, pero sin aportar cálculos importantes que vayan más allá de los que se hace en *EllipticCurve*).

Cabe destacar que no se han incluido en dicho diagrama las clases correspondientes a las interfaces, debido a espacio que ocuparían las mismas debido a su larga extensión de marcos, etiquetas, botones, desplegables e íconos que cada uno posee, sumado a los métodos que se encargan de definir los elementos, validar su rellenado, habilitarlos o deshabilitarlos en la medida que el usuario decide avanzar o retroceder, etc.

Un ejemplo de esto último lo podemos ver en la figura 36, la cual corresponde a la clase que se encarga del manejo de todo lo referido a la lógica de la interfaz de la suma de puntos (*PointAddition*). Allí se define una gran cantidad de elementos gráficos, que son los que conformarán la interfaz misma. Como se verá más adelante en el detalle de cada interfaz, dichos elementos están agrupados en secciones, las cuales si bien son todas visibles desde un comienzo (para que le usuario tenga noción de hacia dónde se quiere llegar), las mismas se encuentran deshabilitadas, y se irán progresivamente habilitando a medida que el usuario vaya cumpliendo con cada uno de los pasos propuestos. Por su parte, si decide volver atrás y cambiar el valor de algún parámetro ya ingresado, entonces las secciones que siguen a la sección modificada se limpian y deshabilitan, a la espera de que el usuario retome su avance desde el punto modificado. Toda la lógica de este comportamiento se encuentra distribuida entre los distintos métodos en cuestión.

A continuación se mostrarán algunas partes del código de estas clases que consideramos más relevantes a los fines de la realización del presente trabajo.

Figura 36: *Clase PointAddition*



5.4.1. Clase Point

Maneja la lógica correspondiente a un punto en particular, con sus coordenadas x e y. Además, maneja la noción de punto en el infinito, lo cual será importante en varios momentos.

```
class Point:
    def __init__(self, x=0, y=0, infinity=False):
        self.x = x
        self.y = y
        self.infinity = infinity

    def set_x(self, x):
        self.x = x

    def get_x(self):
        return self.x

    def set_y(self, y):
        self.y = y

    def get_y(self):
        return self.y

    def is_infinity(self):
        return self.infinity

    def print(self):
        str_print = ""
        if self.infinity:
            str_print = "punto en el infinito"
        else:
            str_print = "(" + str(self.x) + ", " + str(self.y) + ")"
        return str_print

    def __eq__(self, other_point):
        if other_point.is_infinity() and self.is_infinity():
            return True
        if (not other_point.is_infinity() and self.is_infinity()) or (
                other_point.is_infinity() and not self.is_infinity()):
            return False
        return self.get_x() == other_point.get_x() and self.get_y() == other_point.get_y()
```

Figura 37: *Implementación de la clase de un punto*

Esta clase posee también la lógica necesaria como para imprimirse en formato de punto "*(x, y)*" y la rutina *__eq__()*, maneja lo referido a la comparación entre dos puntos para saber si uno es igual al otro o no, especialmente teniendo en cuenta la noción de punto en el infinito. Esto es lo que nos permitirá en otras partes del código el poder compara dos puntos *P* y *Q* diciendo simplemente if *P == Q:* .

5.4.2. Clase EllipticCurve

Es responsable de la lógica correspondiente a las operaciones propias de curva elíptica como ser, desde el establecimiento de los valores de sus parámetros de dominio, hasta fundamentalmente, las operaciones de suma y multiplicación.

Se validan los parámetros de iniciación de la clase en su constructor, arrojando las excepciones que correspondan de acuerdo a la situación.

Incluye además una serie de métodos para determinar si la curva es no-singular, si un punto dado pertenece a la curva, obtener los puntos de la curva, etc.

5.4.3. Método is_non_singular()

Verifica que se cumpla la condición de que el determinante sea distinto a 0 para determinar si la curva es o no singular, es decir, apta para su uso en cifra de curva elíptica o no.

```
def is_non_singular(self, a, b, q):
    if q:
        return (4 * (a ** 3) + 27 * (b ** 2)) % q != 0
    else:
        return (4 * (a ** 3) + 27 * (b ** 2)) != 0
```

Figura 38: *Implementación rutina booleana para curva no-singular*

5.4.4. Método belongs_to_curve()

Contiene la lógica necesaria para corroborar si un punto dado pertenece o no a la curva elíptica presentada, devolviendo el valor booleano correspondiente.

```
def belongs_to_curve(self, point):
    if not isinstance(point, Point):
        raise AssertionError("parámetro de tipo incorrecto")

    return (point.get_y() ** 2) % self.q == \
           ((point.get_x() ** 3) + self.a * point.get_x() + self.b) % self.q
```

Figura 39: *Implementación chequeo de pertenencia de punto a una curva*

5.4.5. Método point_addition()

Contiene la lógica propia de la suma de puntos para una curva elíptica, considerando las distintas situaciones planteas en el apartado teórico.

```
def point_addition(self, point_p, point_q):

    if point_p.is_infinity() and point_q.is_infinity():
        return self.infinity

    if point_p.is_infinity() and not point_q.is_infinity():
        return point_q

    if point_q.is_infinity() and not point_p.is_infinity():
        return point_p

    if point_p.get_x() == point_q.get_x() and \
            (point_p.get_y() != point_q.get_y() or point_p.get_y() == 0):
        return self.infinity

    if point_p.get_x() == point_q.get_x():
        slope = (3 * point_p.get_x() * point_p.get_x() + self.a) * \
                self.inv(2 * point_p.get_y(), self.q) % self.q
        pass
    else:
        slope = (point_q.get_y() - point_p.get_y()) * \
                self.inv(point_q.get_x() - point_p.get_x(), self.q) % self.q
        pass

    point_r = Point()

    x = (slope * slope - point_p.get_x() - point_q.get_x()) % self.q
    y = (slope * (point_p.get_x() - x) - point_p.get_y()) % self.q

    point_r.set_x(x)
    point_r.set_y(y)

    return point_r
```

Figura 40: *Implementación de suma de puntos en campo finito*

Como se puede ver al comienzo del mismo, se contemplan las situaciones específicas en la que se tenga que lidiar con puntos en el infinito, tal como se mencionó en el apartado teórico.

Luego, se consideran los casos de que ambos puntos sean iguales o no, en cuyo caso se aplica la fórmula de pendiente (*slope*) de la duplicación de punto para cuerpos finitos, o por el contrario la pendiente para suma de dos puntos distintos.

5.4.6. Método direct_mult()

Se trata aquí de la implementación directa de la multiplicación de puntos sobre un campo finito.

```
def direct_mult(self, point, k):
    point_r = self.infinity
    for x in range(k):
        point_r = self.point_addition(point_r, point)
    return point_r
```

Figura 41: *Implementación de la multiplicación directa de puntos.*

La misma se encarga de sumar sucesivamente el punto *point* tantas veces como lo indica el escalar *k*. Es por este motivo que no se tratará de una implementación eficiente, lo cual se podrá ver y contrastar con el algoritmo *double-and-add* en la interfaz de multiplicación de punto.

5.4.7. Método double-and-add()

Aquí se implementa la multiplicación de puntos en campo finito, según el algoritmo *double-and-add*, que opera a nivel de bit y cuya ejecución es más veloz que la multiplicación sucesiva simple mencionada en el apartado teórico.

```
def double_and_add(self, point, k):
    point_r = self.zero
    m = point

    while k > 0:
        if k & 1 == 1:
            point_r = self.point_addition(point_r, m)
            pass
        k, m = k >> 1, self.point_addition(m, m)
        pass
    return point_r
```

Figura 42: *Implementación de multiplicación de puntos double-and-add*

Al realizarse menos adiciones dependiendo de la lectura de corrimiento de bits que se hace, con valores de *k* de gran dimensión obtenemos un rendimiento mucho más elevado, permitiéndonos realizar operaciones sobre número de longitud considerable (por ejemplo, como las de las curvas predefinidas en los estándares).

5.5. Evaluación (*testing*)

A fines de probar el correcto funcionamiento de las clases y los métodos aquí implementados, se incluyeron una serie de tests automáticos en la carpeta /tests en el archivo test_ecc.py.

Los valores calculados por la aplicación fueron validados contra los resultados provistas por otras aplicaciones para las mismas operaciones. Por ejemplo, se emplearon resultados obtenidos en JCrypTool, Sagemath, bitcoinexplainer, y otras publicaciones referenciadas en este trabajo.

5.5.1. Funcionamiento

Los *tests* (pruebas) están organizados en clases que deben comenzar con el prefijo "*Test*", y dentro de ellas estarán los métodos que comenzarán con el prefijo "*test*". Dentro de dichos métodos se encontrará la lógicamente propiamente que se busca probar, la cual consistirá en básicamente invocar a las clases previamente mencionadas que conforman al sistema y le dan su funcionalidad. Al invocarlas y crear los objetos correspondientes, se los somete a distintas operaciones cuyo resultado, debería ser el que uno espera.

Así, estos *tests* corroboran que, si ante ciertos datos de entrada el algoritmo probado devolvió la respuesta esperada, entonces el *test* devolverá "*Ok*", y el mismo se considerará exitoso. Por el contrario, si la respuesta no fue la esperada, el *test* devolverá "*Not passed*" y nos lo avisará. Si esto sucede, sabremos que hay algo mal en nuestro código. Esto es muy útil para ir corroborando que los cambios que vamos introduciendo en nuestro programa no vayan generando errores en otros puntos en donde no había inconvenientes.

Para este concepto de testeo llamado testeo unitario, se utiliza el paquete *unittest* de Python quien ya sabe automáticamente qué rutinas debe ejecutar por la nomenclatura recién explicada. Las ejecuta todas, y las mismas devuelvan verdadero o falso.

```
def test_multiplication(self):
    curve = ec.EllipticCurve(10, 15, 23)
    point_p = ec.Point(3, 7)
    point_r = curve.point_mult(point_p, 19)
    point_f = ec.Point(22, 2)
    self.assertEqual(point_r, point_f)
```

Figura 43: *Test de multiplicación con unittest.*

Por ejemplo, en el código de la figura 43, se puede ver que se pone a prueba la multiplicación de curva elíptica implementada. Entonces, el test "*test_multiplication()*" crea una curva elíptica *curve* con sus parámetros de dominio, y luego un punto *point_p* en las coordenadas (3, 7). Luego, se multiplica *point_p* por 19 invocando al método *point_mult()* del objeto *curve*. El resultado de la multiplicación se guarda en la variable *point_r*. Por su parte, nosotros sabemos que el resultado correcto esperado es (22, 2), y en consecuencia creamos un punto *point_f* con dichas coordenadas.

La cuestión entonces es probar si el punto *point_r* es efectivamente igual al punto *point_f* que sabemos es el que consideramos como la respuesta correcta. De ello se encargará *assertEqual()*, el cual tomará a ambos puntos y los comparará en base a la función *__eq__()* definida en la clase *Point()*. En caso afirmativo, veremos por consola el mensaje "Ok". Caso contrario, un mensaje de error. Esto se repetirá para todos los métodos existentes en el archivo.

En conjunción con *unittest*, se incorpora el uso del paquete *ddt*, el cual nos permite adjuntarle a cada *test* una lista de datos de prueba. Entonces, ahora no solamente se ejecutarán los *tests*, sino que cada *test* se ejecutará por cada conjunto de parámetros que venga en la lista de datos de prueba. Un mismo test se ejecutará varias veces con diferentes parámetros.

```
@data((-5, 15, 23), (23, 15, 23), (24, 15, 23))
def test_domain_a_range(self, value):
    with self.assertRaises(AssertionError) as context:
        elliptic_curve = ec.EllipticCurve(*value)
    self.assertTrue("a debe ser mayor o igual a 0 y menor a q" in str(context.exception))
```

Figura 44: *Test con unittest y ddt.*

En el ejemplo de la figura 44, la rutina a evaluar es la creación misma de una curva elíptica, puntualmente verificar si los rangos de valores de *a* son correctamente validados o no. En este caso, la lista de valores a evaluar se encuentra precedida por "*@data*", y se puede ver que son tres conjuntos de datos, es decir, tres valores para *a*, *b* y *q* distintos. En este caso, los tres conjuntos deberían ser rechazados por la clase *EllipticCurve()* ya que si *q = 23*, entonces *a* no puede ser negativo (-5) ni valer exactamente lo mismo que *q* (23), ni valer más que *q* (24).

Entonces, el *test* está armado para que se detecte que efectivamente *EllipticCurve()* arroje una excepción con el texto "*a debe ser mayor o igual a 0 y menor a q*" para los tres conjuntos de datos. Efectivamente, *unittest* en conjunto con *ddt* ejecutan el test *test_domain_a_range* tres veces y las tres veces se detecta que *EllipticCurve* arrojó la excepción de error correspondiente al no tener el parámetro *a* los valores correctos esperados.

```
Process finished with exit code 0

Ran 25 tests in 0.032s

OK
```

Figura 45: *Test automáticos exitosos con unittest y ddt.*

El mensaje devuelto dice "*Ran 25 tests in 0.032s*" porque en el momento de la confección de este texto, existen más tests en el presente archivo. Dentro las 25 aquí nombradas, se encuentran las 3 recién analizadas.

5.5.2. Clase TestEllipticCurve

En esta clase se testean las rutinas que conforman a la clase *EllipticCurve*, abarcando el correcto rango de sus parámetros de dominio, no-singularidad de la curva, suma de puntos, multiplicación de puntos y conmutatividad.

```
@ddt
class TestEllipticCurve(unittest.TestCase):

    @data(("3", "4", "5"), (3.3, 4.4, 5.5), ("hola", "como", "estas"), ("", "", ""))
    def test_domain_params_are_int(self, value):
        with self.assertRaises(AssertionError) as context:
            elliptic_curve = ec.EllipticCurve(*value)
        self.assertTrue("a, b y q deben ser números" in str(context.exception))

    @data((10, 15, 24))
    def test_domain_q_is_prime(self, value):
        with self.assertRaises(AssertionError) as context:
            elliptic_curve = ec.EllipticCurve(*value)
        self.assertTrue("q debe ser un número primo" in str(context.exception))


    @data((-5, 15, 23), (23, 15, 23), (24, 15, 23))
    def test_domain_a_range(self, value):
        with self.assertRaises(AssertionError) as context:
            elliptic_curve = ec.EllipticCurve(*value)
        self.assertTrue("a debe ser mayor o igual a 0 y menor a q" in
str(context.exception))

    @data((10, -15, 23), (10, 23, 23), (10, 50, 23))
    def test_domain_b_range(self, value):
        with self.assertRaises(AssertionError) as context:
            elliptic_curve = ec.EllipticCurve(*value)
        self.assertTrue("b debe ser mayor o igual a 0 y menor a q" in
str(context.exception))

    @data((1, 1, 2))
    def test_domain_q_range(self, value):
        with self.assertRaises(AssertionError) as context:
            elliptic_curve = ec.EllipticCurve(*value)
        self.assertTrue("q debe ser mayor a 2" in str(context.exception))

    @data((2, 3, 5))
    def test_curve_non_singularity(self, value):
        with self.assertRaises(AssertionError) as context:
            elliptic_curve = ec.EllipticCurve(*value)
        self.assertTrue("curva elíptica singular" in str(context.exception))
```

Figura 46: *Muestra de tests para clase EllipticCurve*

5.5.3. Clase TestECDH

En esta clase se testean las rutinas que conforman a la clase ECDH.

En el ejemplo de la figura 47, se prueba el correcto funcionamiento del protocolo ECDH haciendo uso de los valores de una curva del estándar Brainpool, operando con las operaciones de multiplicación implementadas en este trabajo.

Como se puede observar, el resultado final es corroborar que la clave secreta compartida calculada por Bob (shared_secret_by_bob) es igual a la clave secreta compartida calculada poralice (shared_secret_by_alice). Además, se busca chequear que, más allá de que ambas sean iguales o no entre sí como producto de la operación de multiplicación de punto, hay que comparar que dicho valor sea el mismo que el que nosotros esperábamos de antemano obtener, representado por el punto *point_r*.

```
def test_predefined_curve_2(self):
    curva = cons.EC_LIST["brainpoolP192r1"]
    self.elliptic_curve = ec.EllipticCurve(curva["a"], curva["b"], curva["q"],
ec.Point(curva["g"][0],
                                curva["g"][1]), curva["n"], curva["h"])
    ecdh = ec.ECDH(self.elliptic_curve)
    bob_priv_key = 0x378394C3274253FD15531812
    bob_pub_key = ecdh.gen_pub_key(bob_priv_key)

    alice_priv_key = 0x151FF34164E0A753BAE0B506
    alice_pub_key = ecdh.gen_pub_key(alice_priv_key)

    shared_secret_by_bob = ecdh.calc_shared_key(bob_priv_key, alice_pub_key)
    shared_secret_by_alice = ecdh.calc_shared_key(alice_priv_key, bob_pub_key)

    # expected point as result:
    point_r =
ec.Point(4239331162207121876592347022408768142110490207776903487994,

26332994000319712378309775259470799284032961321417380362 52)

    self.assertEqual(shared_secret_by_bob, shared_secret_by_alice)
    self.assertEqual(shared_secret_by_bob, point_r)
```

Figura 47: *Muestra de tests para clase ECDH.*

5.6. Interfaz de Usuario

Pasaremos a revisar las distintas interfaces que conforman a la aplicación.

Al ejecutar la misma, nos encontraremos con la página principal (ver figura 48), y lo primero que podremos observar es la estructura general de la misma, la cual posee un menú superior horizontal dividido en tres secciones:

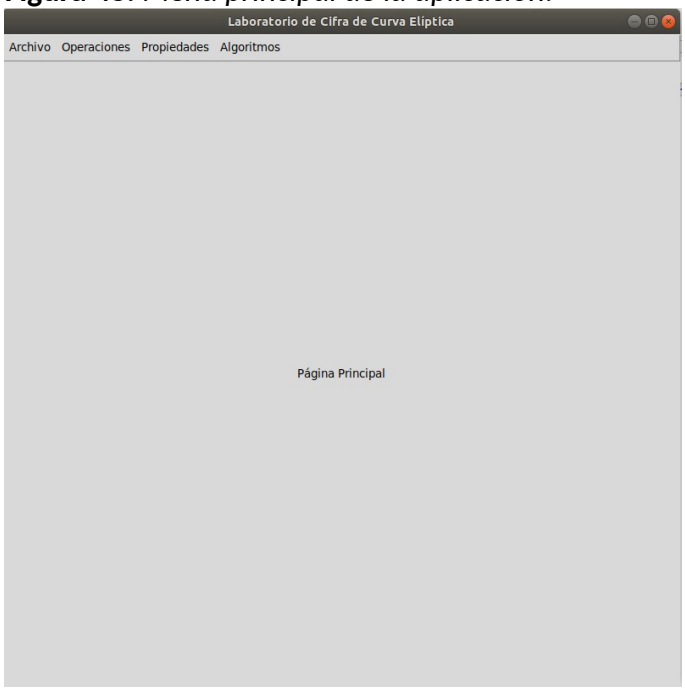
- **Operaciones**: concebida para desplegar el acceso a las interfaces de operaciones básicas (ej.: Suma de Puntos y Multiplicación de Puntos).

- **Propiedades**: contiene interfaces donde se estudian propiedades de las curvas elípticas, como por ejemplo la propiedad de conmutatividad.

- **Algoritmos**: despliega las interfaces de algoritmos concretos que hacen uso de las operaciones y propiedades propuestas en los menúes anteriores. Por ejemplo, allí se encuentra listado el acceso a la interfaz de ECDH.

En el borde superior de la ventana se puede observar que se la puesto a la aplicación el nombre de “Laboratorio de Cifra de Curva Elíptica”.

Figura 48: *Menú principal de la aplicación.*



Cabe destacar que en las siguientes interfaces a revisar, todas ellas poseen controles específicos con mensajes de error que se realizan a la hora de que el usuario vaya llenando las secciones paso a paso. Se mostrarán algunas a modo de ejemplo, pero no se buscará mostrar en este documento todos los mensajes de error correspondientes a la totalidad de los controles que la aplicación realiza.

5.6.1. Interfaz de Suma de Puntos

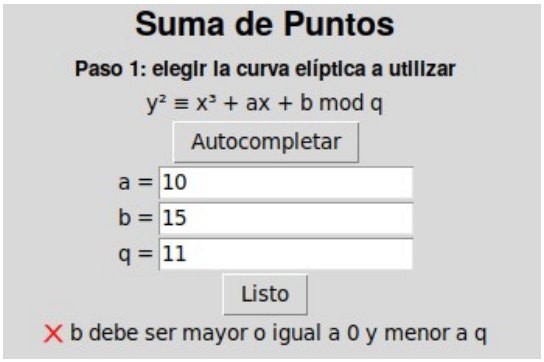
Para llegar a esta interfaz es necesario ir a **Operaciones → Suma** (ver figura 49).

Figura 49: *Inicio de interfaz de suma de puntos.*

Se trata de la primera interfaz de la aplicación, y se puede observar que la misma constará de tres pasos, de los cuales solamente se encuentra habilitada la primera, es decir, la selección misma de la curva elíptica sobre la cual se realizarán los consiguientes cálculos.

El usuario puede elegir él una curva elíptica en forma manual, y el sistema no le permitirá avanzar en caso de que alguno de los valores ingresados de su parámetro de dominio no cumplan con alguna regla necesaria (ver figura 50). También dispone de un botón “Autocompletar” para caso de que el usuario desee avanzar rápido o simplemente no esté seguro de lo que está completando, y así no quedar trabado en este paso.

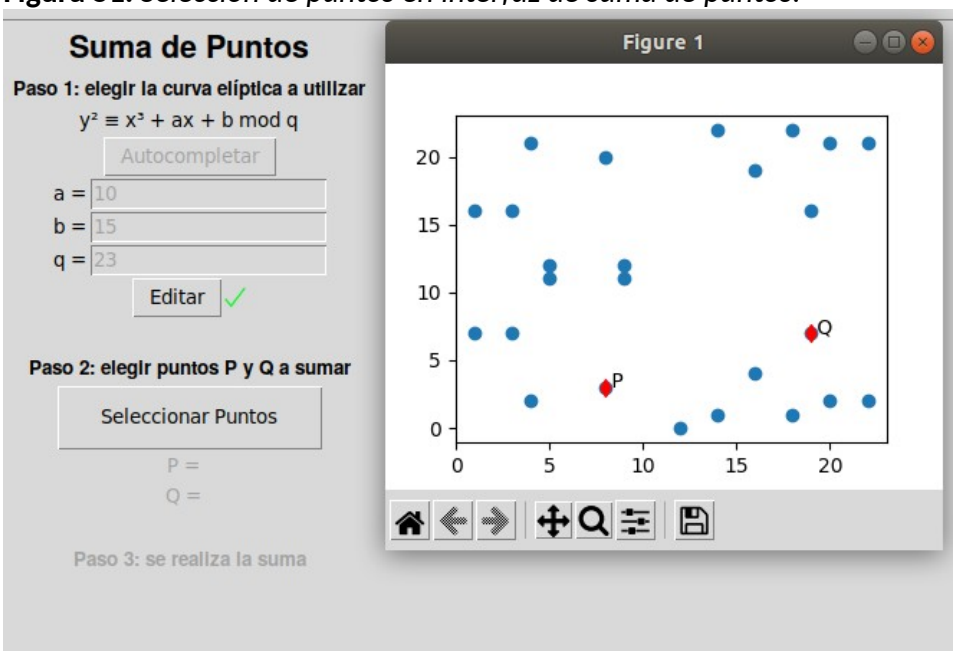
Figura 50: *Mensajes de error en interfaz de Suma de Puntos.*



A continuación, si se ingresan valores aceptados, se continúa al siguiente paso, el cual consiste en la selección de puntos sobre la curva en cuerpo finito recién ingresada, haciendo clic en el botón "Seleccionar Puntos" (ver figura 51). Esto lanzará una ventana con un gráfico sobre el cual se grafican los puntos que conforman a la curva.

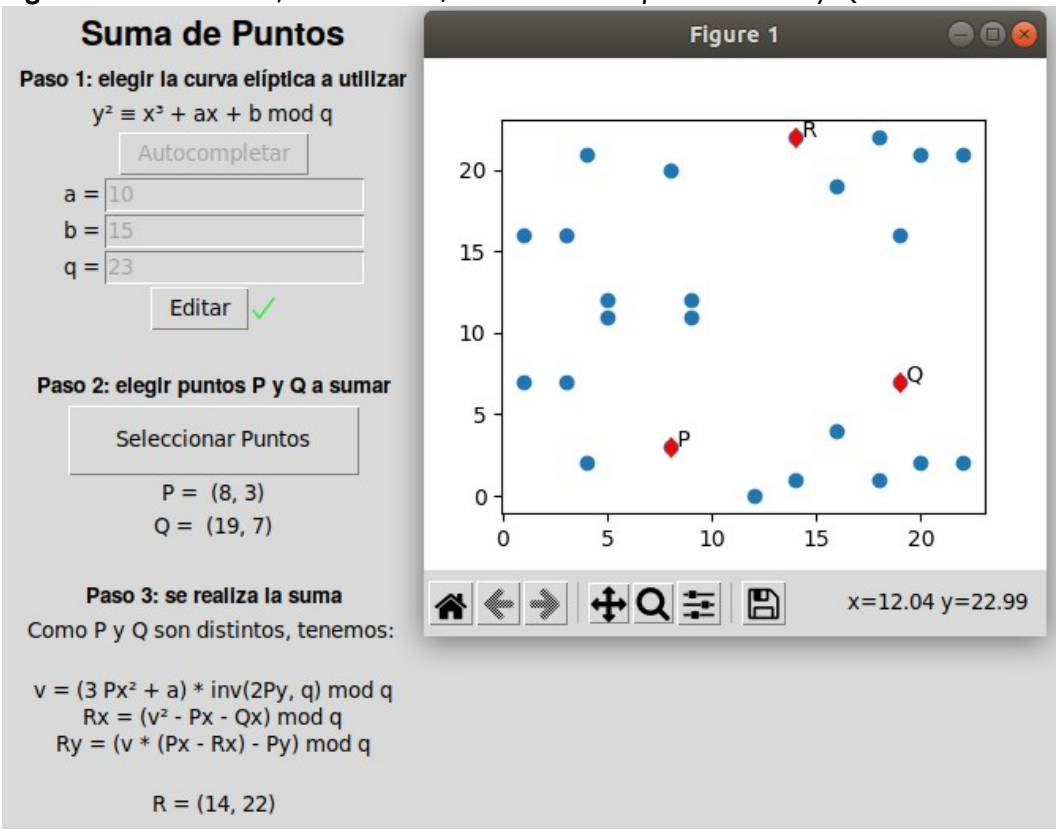
Aquí, como se trata de la suma de dos puntos, debemos seleccionar *P* y *Q*, que pueden ser distintos, o el mismo punto.

Figura 51: *Selección de puntos en interfaz de suma de puntos.*



Entonces, procedemos a cerrar la ventana con la selección realizada, y automáticamente se registran ambos valores, se procede a hacer el cálculo del punto *R* resultante (aparece otro gráfico en el que además de *P* y *Q* ahora figura *R*), acompañado del listado de ecuaciones utilizadas de acuerdo dependiendo de los puntos seleccionado (no es lo mismo si *P* y *Q* son puntos distintos o iguales, por ejemplo).

Figura 52: *Resultado final de interfaz de suma de puntos con P y Q distintos.*

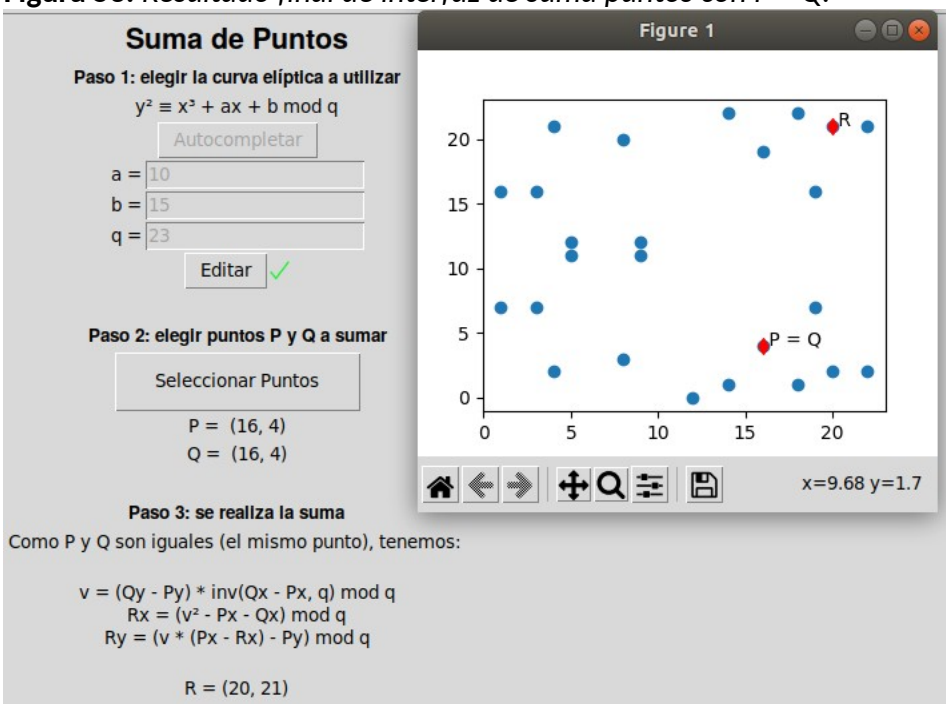


Así, si el usuario hace clic en el botón “Seleccionar Puntos” y elige otro par de puntos, al cerrar la ventana el valor de *R* y las ecuaciones empleadas se actualizarán automáticamente (ver figura 53). Para el estudiante conlleva un valor agregado por ver qué ecuaciones se están efectivamente usando en cada caso.

A su vez, si en este paso o en el anterior, el usuario hubiese querido volver a atrás haciendo clic en el botón “Editar” para cambiar de curva elíptica, entonces toda la interfaz se hubiese limpiado del contenido desplegado hasta ese momento, y volvería al estado original que tenía en la figura 49.

Estas características en el manejo del avance y retroceso en cada paso, en lo que respecta al manejo de mensajes de error, activación y desactivación de distintos elementos visuales, gestión de los gráficos, etc., se repetirá en las demás interfaces con la misma dinámica que se pudo observar en esta interfaz.

Figura 53: *Resultado final de interfaz de suma puntos con P = Q.*



5.6.2. Interfaz de Multiplicación de Puntos

Para llegar a esta interfaz es necesario ir a **Operaciones → Multiplicación** (ver figura 54).

En esta interfaz se puede observar que el primer paso (selección de curva elíptica) es la mismas que se emplea en la interfaz de suma de puntos ya vista.

Las diferencias comienzan a partir del segundo paso, en el cual se le solicita el usuario que ingrese un solo punto en vez de dos, puesto que aquí el objetivo será multiplicar dicho punto por un valor escalar.

Entonces, una vez elegido el punto, en el siguiente paso se solicita ingresar el valor *n* del escalar al usar en la multiplicación (ver figura 55).

Figura 54: *Inicio de interfaz de multiplicación de punto.*

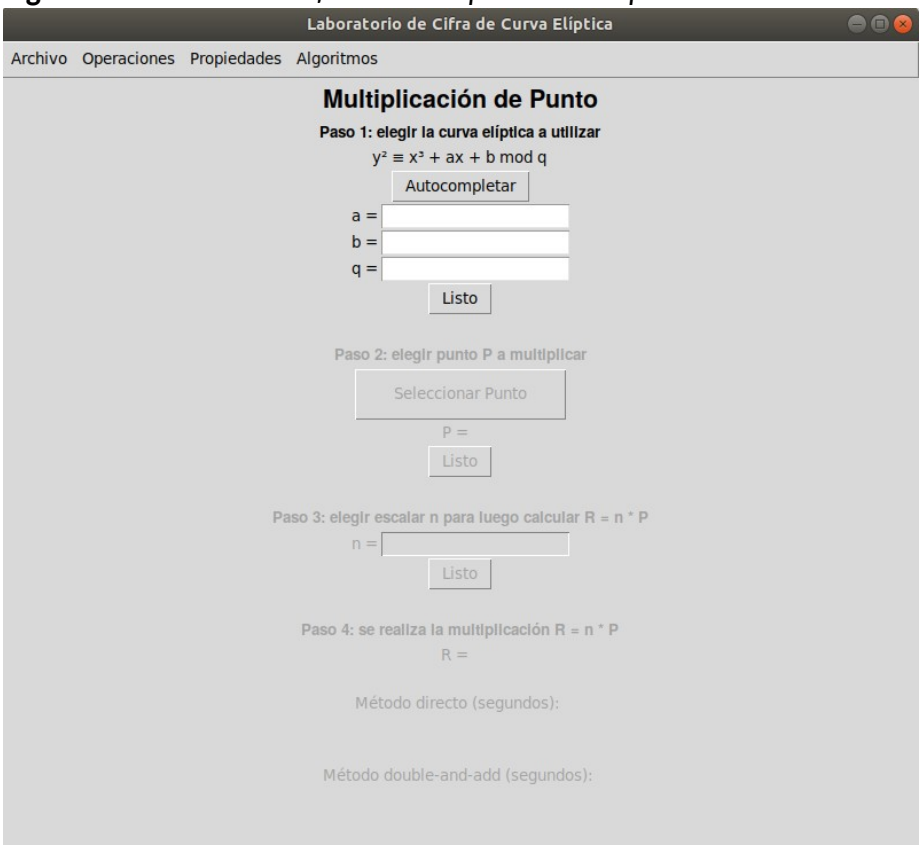
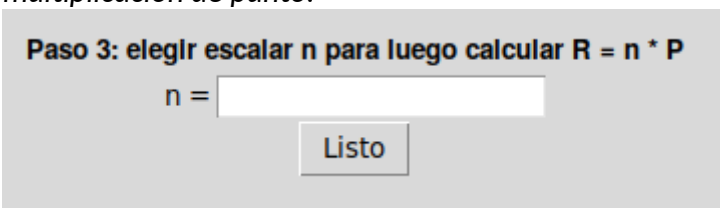
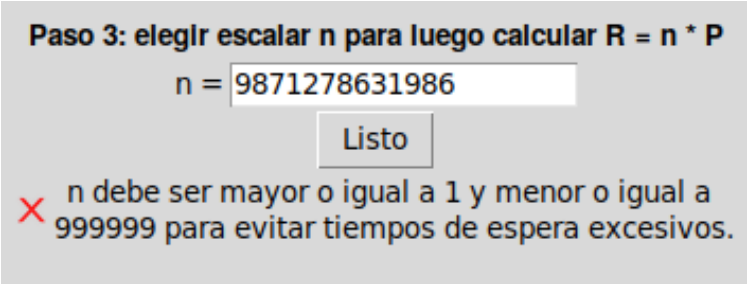


Figura 55: *Paso de selección de valor escalar para multiplicación de punto.*



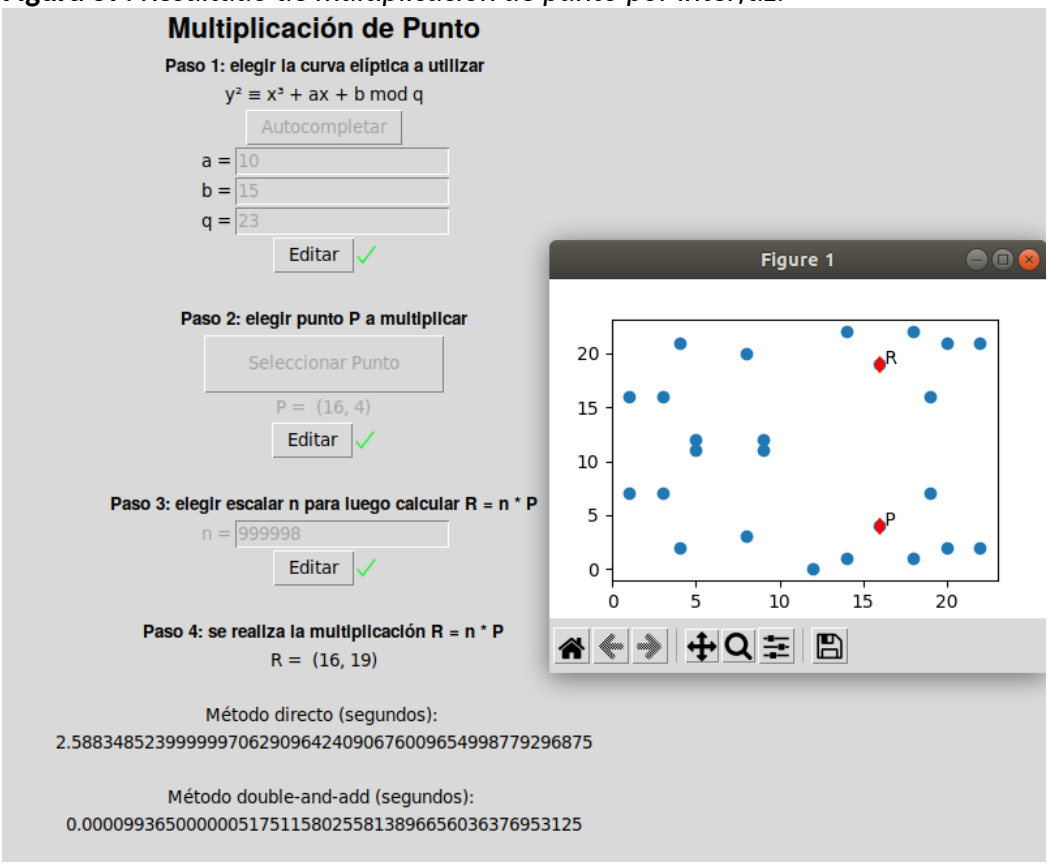
Debido a que el objetivo de esta interfaz no es solamente observar el resultado de realizar la multiplicación en sí misma, sino también poder compara el rendimiento de realizarla mediante el método directo en contraste al algoritmo *double-and-add*, se ha restringido la longitud de *n* a introducir en esta casilla, a un valor de un número máximo de seis dígitos, establecido en el archivo *constants.py* (ver figura 56).

Figura 56: *Mensaje de error ante escalar demasiado grande en interfaz de multiplicación.*



Entonces, si el estudiante ingresa un valor de *n* dentro del rango permitido pero lo suficientemente cercano a 999999, entonces podrá apreciar como resultado final que el algoritmo *double-and-add* logra realizar el cálculo de forma mucho más rápida que por el método directo mediante la medición en segundos necesaria por ambos algoritmos para realizarlo (ver figura 57). Incluso experimentará un leve retraso que evidenciará cuánto ha tardado el método directo en comparación a *double-and-add* en terminar sus cálculos.

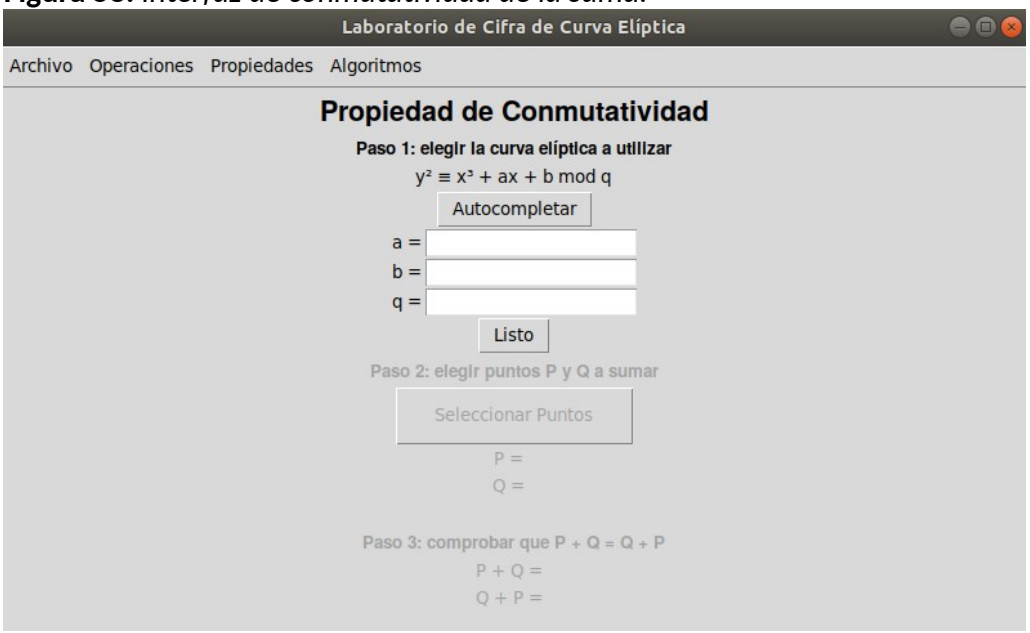
Figura 57: *Resultado de multiplicación de punto por interfaz.*



5.6.3. Interfaz de Propiedad de Conmutatividad

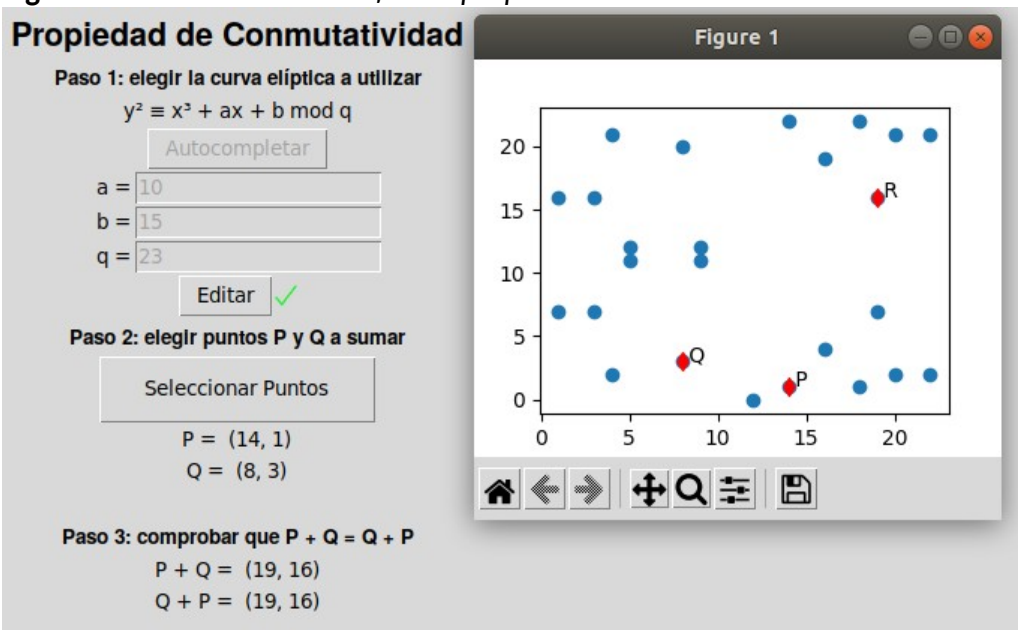
Para llegar a esta interfaz es necesario ir a **Propiedades → Conmutatividad** (ver figura 58).

Figura 58: *Interfaz de conmutatividad de la suma.*



El caso de esta interfaz es similar al de la suma de puntos en lo que respecta a sus dos primeros pasos: selección de curva elíptica, y dos punto *P* y *Q* sobre ella.

Figura 59: *Resultado de interfaz de propiedad conmutativa.*

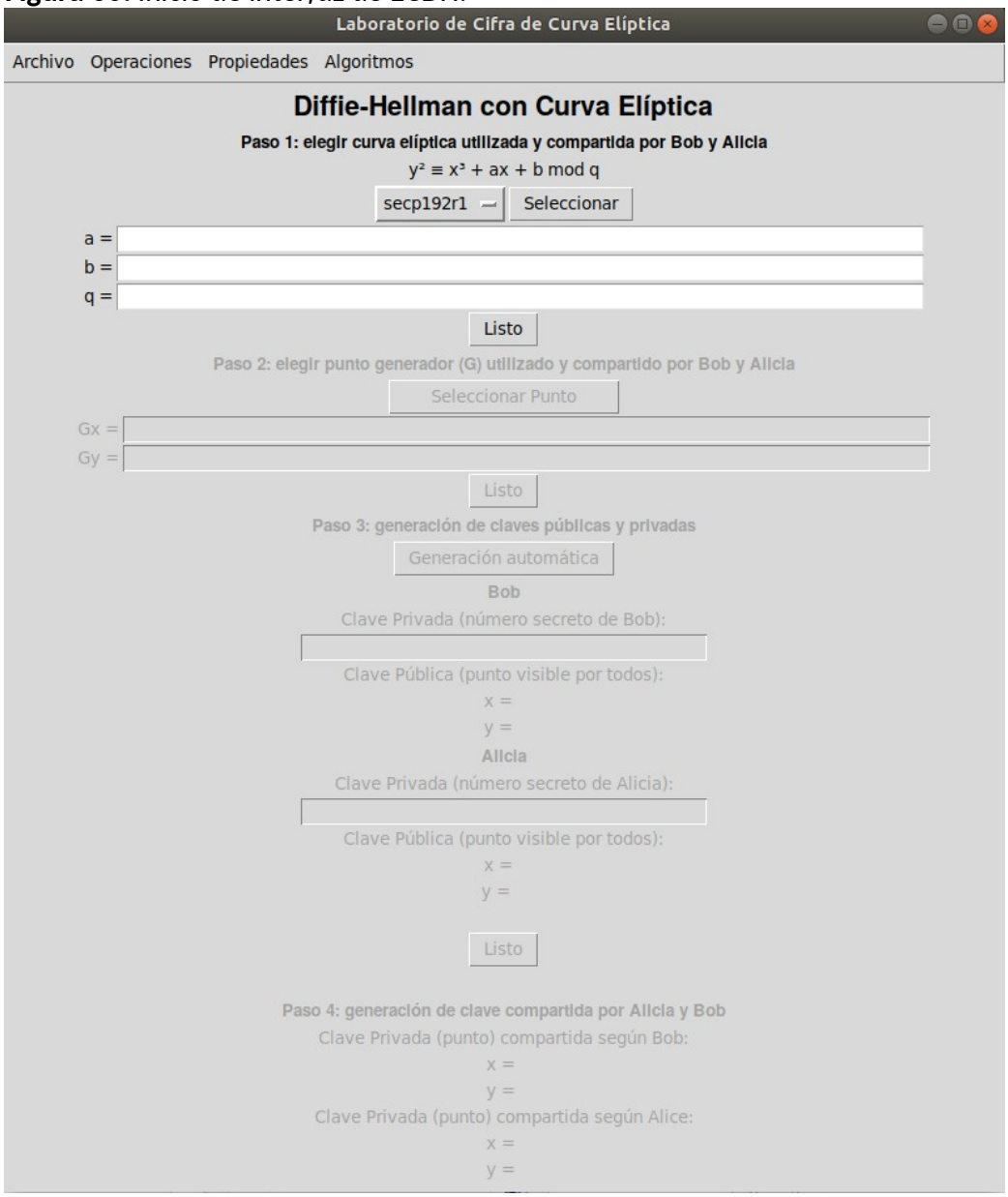


La diferencia radica en que, en esta interfaz, se verifica explícitamente que hacer *P + Q* es equivalente a hacer *Q + P*, conformando una de las propiedades abelianas de la suma sobre curvas elípticas, que es la conmutatividad (ver figura 59).

5.6.4. Interfaz de ECDH

Para llegar a esta interfaz es necesario ir a **Algoritmos → Diffie-Hellman con curva elíptica (ECDH)** (ver figura 60).

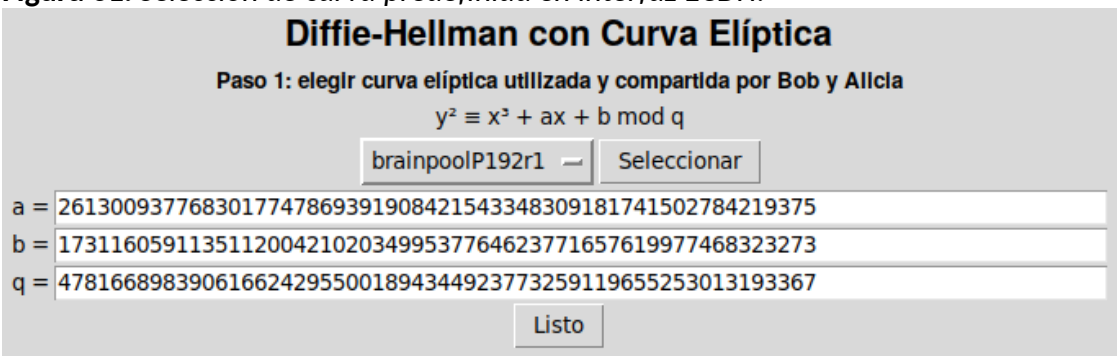
Figura 60: *Inicio de interfaz de ECDH.*



Allí se puede observar que la interfaz se encuentra organizada en 4 pasos, entre los cuales el usuario puede avanzar y retroceder en la medida que éste así lo desee a lo fines de probar el resultado de cambiar ciertos valores. En la misma se simula el paso a paso que dos partes comunicantes (Bob y Alice) llevan a cambo para arribar, por separado, a la misma clave secreta en común.

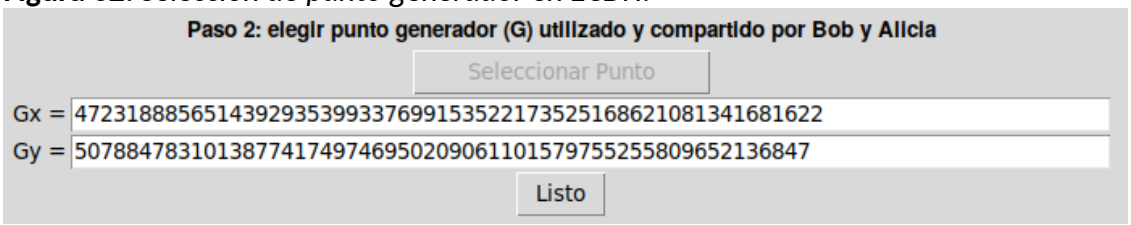
En primer lugar, el usuario tiene la posibilidad de determinar él mismo qué valores de parámetros de dominio emplear para una curva elíptica (tal y como sucedía en las demás interfaces), pero esta tiene la particularidad de que también permite elegir, por medio de un desplegable, algunas curvas de los estándares NIST o Brainpool para poder trabajar con ellas de forma interactiva (ver figura 61). En este último caso, los valores de cada curva rellenarán automáticamente campos de la interfaz.

Figura 61: *Selección de curva predefinida en interfaz ECDH.*



La consecuencia de esto es que también se rellenarán automáticamente los campos del punto generador G, el cual es el punto que ambas partes comunicantes en el protocolo de Diffie-Hellman compartirán para realizar ambos multiplicaciones sobre dicho punto (ver figura 62).

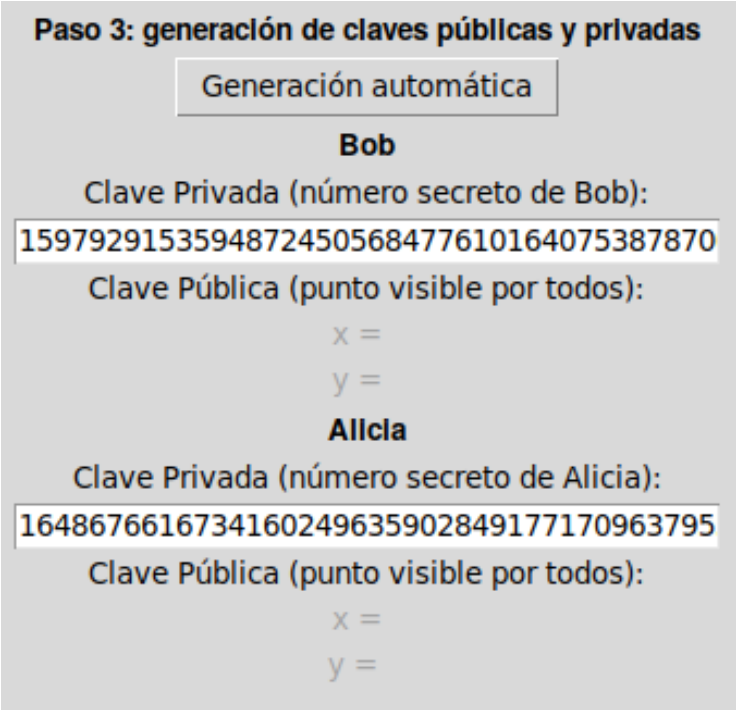
Figura 62: *Selección de punto generador en ECDH.*



Como se ve en dicha imagen, el botón “Seleccionar Punto” se encuentra deshabilitado, porque en el paso anterior se seleccionó una curva predefinida del desplegable. Cuando eso sucede el punto G ya viene determinado por los estándares de las curvas en cuestión, y el mismo es respetado. Sin embargo, si el usuario hubiese elegido una curva elíptica ingresando sus parámetros de dominio en forma manual, entonces este botón se encontraría habilitado, y podríamos seleccionar alguno de los puntos graficados de la misma forma que se lo hace en otras interfaces ya vistas.

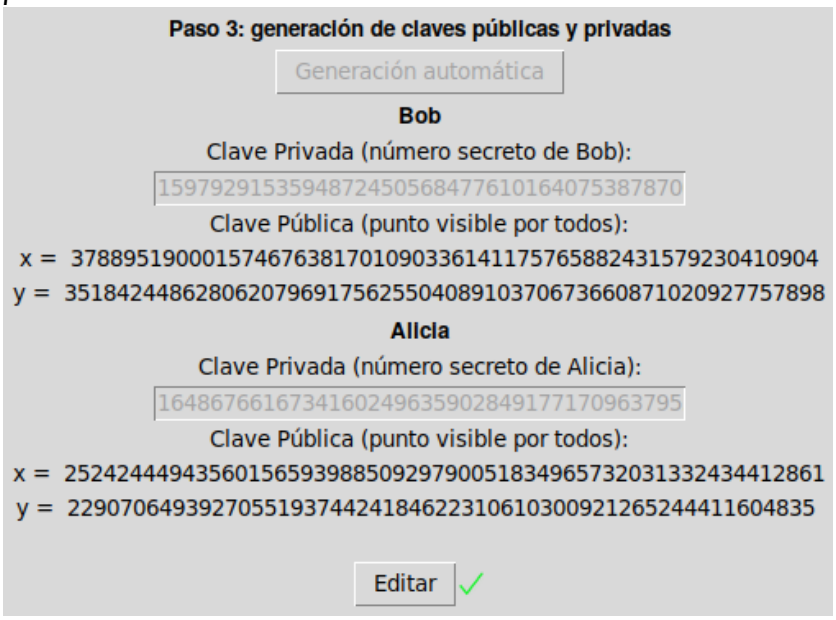
A continuación, es necesario determinar las clave privadas y públicas de tanto Bob y Alice, las cuales surgen de elegir cada uno un número escalar (clave privada) y multiplicarlo por el punto G previamente seleccionado. El punto resultante será la clave pública de cada uno. Si el usuario lo desea, puede generar las claves privadas de forma automática mediante el botón “Generación automática” (ver figura 63).

Figura 63: *Generación automática de claves privadas en interfaz ECDH.*



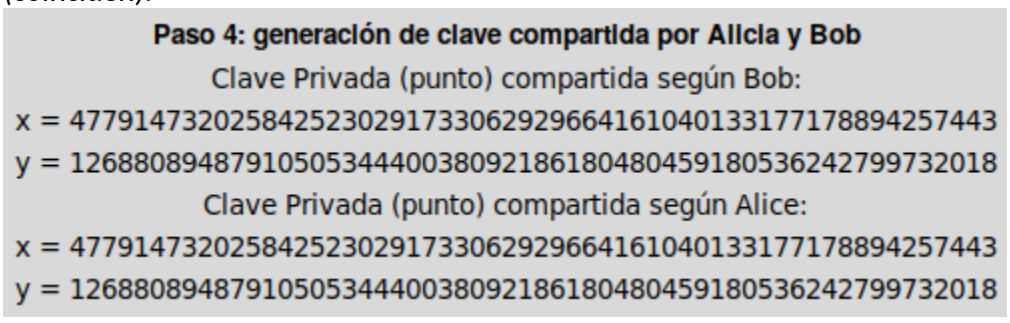
En consecuencia, al aceptar este paso, veremos que se calcularán las coordenadas de los puntos que representan las claves públicas de Bob y Alice (ver figura 64), las cuales serán empleadas para el cálculo restante de la clave secreta compartida entre ambos.

Figura 64: *Resultado de generación de claves públicas y privadas en ECDH.*



Finalmente, en el último paso, vemos que Bob y Alice toman la clave pública del otro, y al multiplicarla por sus propias claves privadas, cada uno arriba a una misma clave secreta (ver figura 65) (otro punto sobre la curva de la cual luego se usará alguna de sus coordenadas como clave en conjunción con otro algoritmo).

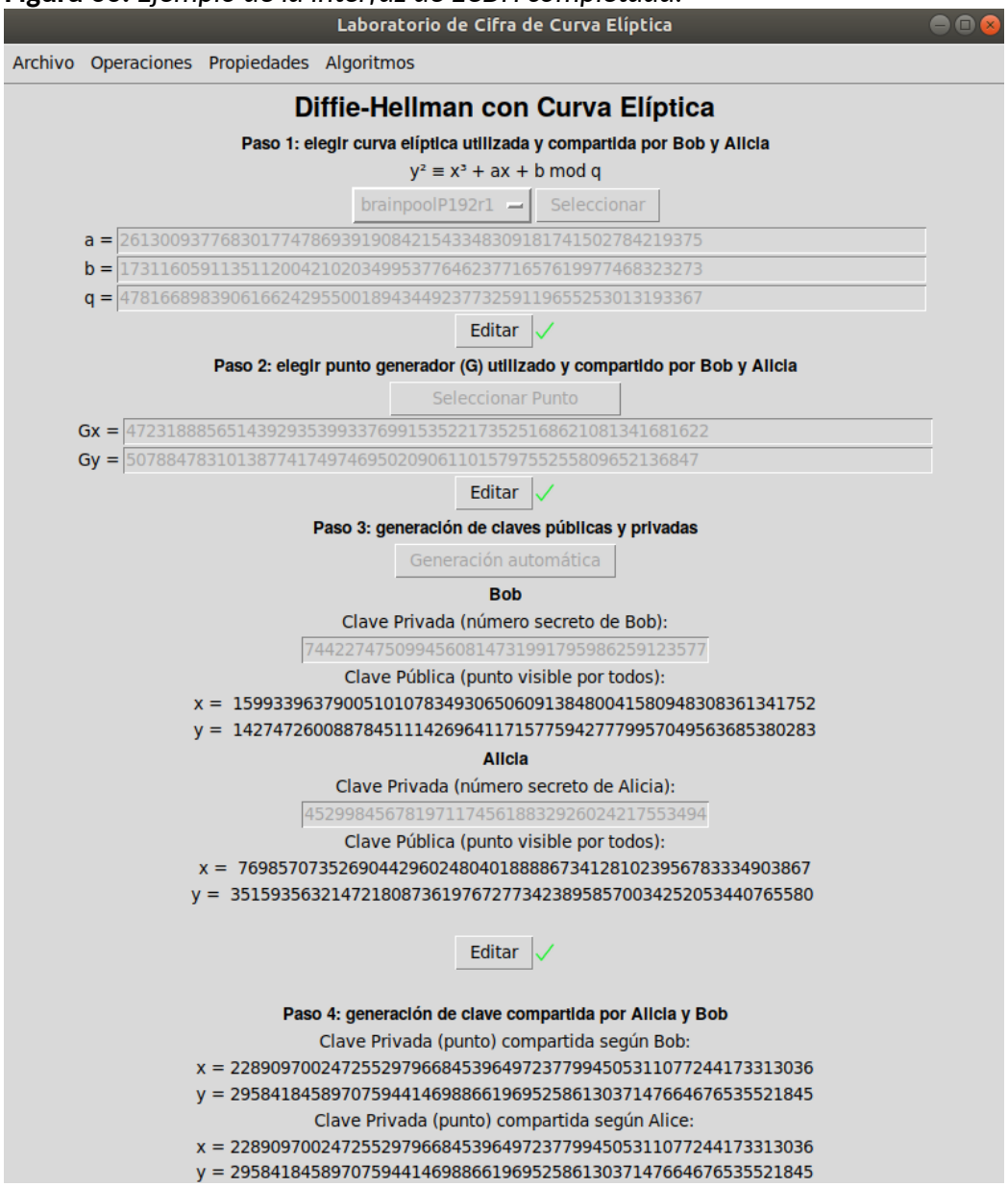
Figura 65: *Cálculo de clave privada compartida por parte de Bob y Alice (coinciden).*



Una vez llegado a este punto, es posible retroceder a alguno de los pasos anteriores para modificar su valor, lo cual también conllevará que, todos los pasos posteriores al que se

quiera modificar sean limpiados y deshabilitados, a la espera de realizar nuevos cálculos con los nuevos datos ingresados por el usuario.

Figura 66: *Ejemplo de la interfaz de ECDH completada.*



6. Conclusiones

En base a los objetivos específicos que fueran listados en el capítulo 3, concluimos:

- **Objetivo**: Estudiar la teoría de curva elíptica, logrando que queden armados los cimientos básicos para la creación de una plataforma/laboratorio de aprendizaje exclusivo.

  Resultado: se ha alcanzado el objetivo de construir el marco general de una aplicación enfocado completamente al estudio de las curvas elípticas y la criptografía de curva elíptica (ECC), abarcando interfaces dedicadas a sus dos operaciones fundamentales de suma y multiplicación, una a la propiedad abeliana de conmutatividad y finalmente otra para el estudio del algoritmo de intercambio de claves de Diffie-Hellman con curva elíptica (ECDH). De esta forma, la aplicación queda preparada y organizada para, de cara futuras líneas de trabajo, poder incorporar nuevas funcionalidades y enriquecerse, para así poder convertirse en una plataforma más completa dedicada al estudio de las curvas elípticas y sus propiedades.

- **Objetivo**: Lograr que sea comprensible para estudiantes de ingeniería.

  Resultado: el diseño y construcción de las distintas interfaces se hizo teniendo en mente que el público objetivo del mismo son estudiantes de ingeniería que se encuentren realizando un curso de criptografía. Desde el ordenamiento de los tópicos en el menú horizontal superior que despliega las interfaces secuencialmente de más elemental y sencillo a complejo, hasta las terminologías empleadas en distintas etiquetas a lo largo de las interfaces para fomentar una asimilación más sencilla de los conceptos allí plasmados.

- **Objetivo**: Construir la aplicación de tal forma que sea interactiva.

  Resultado: en todas las interfaces de la aplicación se hace énfasis en elementos interactivos que van desde cajas de texto y botones, hasta gráficos sobre los cuales el usuario debe hacer clic para seleccionar o quitar puntos dependiendo de la situación puntual. Además, todas las interfaces están pensadas como un paso a paso sobre el

cual el usuario puede avanzar y retroceder, aplicando cambios en el camino, y la interfaz respondiendo acordemente. Por otro lado, en múltiples ubicaciones se asiste al usuario con funciones de autocompletado, no solamente para avanzar más rápido, sino también por si no está seguro de qué debería completar en dichos pasos.

- **Objetivo: Comprender y probar la operación de suma de puntos.**

  Resultado: se ha logrado elaborar una interfaz en la cual es posible practicar y probar distintas combinaciones de curvas elípticas, seleccionando distintos puntos sobre ellas. En caso de ingresar parámetros de dominio inválidos, la aplicación desafía al usuario a corregirlos, e incluso ofrecer autocompletar los datos iniciales. Luego, se procede a una gráfico interactivo sobre el cual el usuario puede seleccionar los puntos en cuestión, y proceder a calcular el resultado, no sin antes mostrar explícitamente cuáles son las ecuaciones que se están empleando en cada caso. Están contempladas las situaciones en que ambos puntos sean distintos, iguales, o incluso que su suma arroje el punto en el infinito (O). Todo esto facilita al estudiante la comprensión de la que es la operación más fundamental de toda la ECC.

- **Objetivo**: Comprender y probar la multiplicación de puntos.

  Resultado: se pudo construir una interfaz en la cual se elija una curva, un punto y un escalar para realizar la multiplicación del mismo por sobre el punto. Además, se visualiza concretamente el tiempo tomado entre la realización del mismo por el algoritmo de multiplicación directo y por *double-and-add*, evidenciando que el segundo tiene mayor rendimiento que el primero. Incluso se permite ingresar un valor escalar lo suficientemente grande, como para que el usuario note la lentitud que le toma a la aplicación realizar dicho cálculo por el método directo cuando se encuentre en dicha interfaz.

- **Objetivo**: Comprender y probar alguna de las propiedades de grupo abeliano de una curva elíptica.

Resultado: se logró construir una interfaz en la que se estudie la propiedad conmutativa, en la cual se selecciona una curva elíptica, un par de puntos sobre ella, y luego se procede a verificar que $P + Q$ y $Q + P$ arrojan el mismo resultado final.

- **Objetivo**: Comprender y probar el algoritmo de Diffie-Hellman para curva elíptica (ECDH).

  Resultado: se ha construido una interfaz sobre la cual se puede hacer, de comienzo a fin, el recorrido por todas las operaciones que realizan las dos partes comunicantes para arribar, como resultado final, a que cada uno obtenga una clave secreta compartida. Particularmente en esta interfaz se incorporó la posibilidad de poder seleccionar curvas reales de estándares como el NIST o Brainpool, para que el usuario pueda interactuar con ellas y hacer distintas pruebas. Esto incluye utilizar el punto generador G que viene provisto en el estándar, o cambiarlo por otro si así lo desea, e incluso asistirlo en la generación automática de las claves secretas y públicas de ambos. Al final se comparan los resultados obtenidos, y el usuario puede corroborar que efectivamente ambos lograron, por separado, llegar al mismo punto que funciona como clave pública.

6.1. Líneas de trabajo futuros

Las principales líneas de trabajo a futuro son:

- Incluir interfaces donde se estudien más algoritmos como la firma digital con curva elíptica (ECDSA) o el algoritmo de ElGamal con curva elíptica. Esto conllevaría a que el valor de la presente aplicación aumente, al ir incorporando más elementos propios de la ECC, haciéndola más atractiva para más estudiantes y docentes de criptografía.

- Considerar agregar una sección enfocada en ataques a curvas elípticas en donde se puedan hacer ejemplos explícitos en donde se intente resolver el logaritmo discreto para curva elíptica (ECDLP) (Musson, s. f.), o en donde se pueda evaluar otras debilidades matemáticas que posean ciertas curvas elípticas. Por ejemplo, se podría estudiar puntualmente el algoritmo de Pollard's Rho (Silverman, 2009).

- Completar el estudio de las propiedades abelianas de curva elíptica más allá de la conmutatividad realizada, incluyendo asociatividad, existencia de elemento inverso, existencia de elemento identidad, y cierre.

- Incorporar alguna referencia práctica que le permita al estudiante estudiar el Teorema de Helmut-Hasse para estimar el número de puntos que tiene una curva elíptica *E* definida sobre un campo F_q (Silverman, 2009).

- Enriquecer la aplicación incorporando interfaces prácticas y mejorando los algoritmos ya desarrollados, para que se pueda estudiar de forma más detallada todos los parámetros de dominio de una curva elíptica, particularmente su orden (*n*), puntos generadores (*G*) y cofactor (*h*). Esto es importante porque le permite al estudiante comprender mejor por qué algunas curvas son consideradas más seguras que otras en lo que respecta los subgrupos cíclico que se forman, cómo hallar dichos subgrupos (multiplicando hasta llegar al punto infinito), y por qué en algunas curvas de estándares como el NIST se decide tomar alguno de esos valores como mejores que otros.

- En el presente trabajo se explicó y abordó la temática de las curvas elípticas desde el punto de vista de los números reales y los cuerpos finitos. Sin embargo, también existen los campos binarios que quedaron fuera del alcance de lo aquí desarrollado, y que también podría incorporarse en un futuro a la presente aplicación (Reyes et al., 2013).

Referencias Bibliográficas

Adamovic, S., Sarac, M., Stamenkovic, D., & Radovanovic, D. (2018). The Importance of the Using Software Tools for Learning Modern Cryptography. *International Journal of Engineering Education*, *34*(256-262), 8.

Aglawe, D., & Gajbhiye, S. (2012). Software Implementation of Cyclic Abelian Elliptic Curve using Matlab. *International Journal of Computer Applications*, *42*(6), 43-48. https://doi.org/10.5120/5700-7754

Aryanti, A., & Mekongga, I. (2018). Implementation of Rivest Shamir Adleman Algorithm (RSA) and Vigenere Cipher In Web Based Information System. *E3S Web of Conferences*, *31*, 10007. https://doi.org/10.1051/e3sconf/20183110007

Barker, E. (2016). *Recommendation for Key Management Part 1: General* (NIST SP 800-57pt1r4; p. NIST SP 800-57pt1r4). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-57pt1r4

Bitcoin explainer. (s. f.). https://github.com/nlitsme/bitcoinexplainer

Blake, I. F., Seroussi, G., & Smart, N. P. (Eds.). (2005). *Advances in elliptic curve cryptography*. Cambridge University Press.

Brown, D. R. L. (2010). *SEC 2: Recommended Elliptic Curve Domain Parameters*. Certicom Corp.

Brown, E., & Myers, B. T. (2002). Elliptic Curves from Mordell to Diophantus and Back. *The American Mathematical Monthly*, *109*(7), 639. https://doi.org/10.2307/3072428

Cryptool. (s. f.). https://www.cryptool.org/en/

Diagrams.net. (2022). https://app.diagrams.net/

Díaz Arroyo, R. (2018). *GenRSA: Software para la Generación de Claves RSA, Cifra, Firma y Ataques* (2.1) [Computer software]. Escuela Técnica Superior de Ingeniería de

Sistemas Informáticos Universidad Politécnica de Madrid - España. http://www.criptored.upm.es/software/sw_m001d.htm

Dooley, J. F. (2013). *A Brief History of Cryptology and Cryptographic Algorithms*. Springer International Publishing. https://doi.org/10.1007/978-3-319-01628-3

Dowling, T. (2006). The design and evaluation of a cryptography teaching strategy for software engineering students. *European Journal of Engineering Education*, *31*(5), 593-606. https://doi.org/10.1080/03043790600797434

Elliptic Curve Cryptography (ECC). (2022, febrero). The History and Benefits of ECC Certificates. https://www.digicert.com/faq/ecc.htm

Elliptic Curve Cryptography vs RSA Certificates: What's the Difference? (2022, febrero). Elliptic Curve Cryptography vs RSA Certificates: What's the Difference? https://sectigostore.com/page/elliptic-curve-cryptography-vs-rsa-certificates-whats-the-difference/

Gençoğlu, M. T. (2019). Importance of Cryptography in Information Security. *IOSR Journal of Computer Engineering (IOSR-JCE)*, *21*(1), 65-68. https://doi.org/10.9790/0661-2101026568

Gennaro, R., Goldfeder, S., & Narayanan, A. (2016). Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. En M. Manulis, A.-R. Sadeghi, & S. Schneider (Eds.), *Applied Cryptography and Network Security* (Vol. 9696, pp. 156-174). Springer International Publishing. https://doi.org/10.1007/978-3-319-39555-5_9

Giry, D. (2020, mayo 24). *Cryptographyic Key Length Recommendation*. https://www.keylength.com/en/compare/

Grau, S. (2011). *Elliptic Curves over Finite Fields*. https://graui.de/code/elliptic2/

Hankerson, D. R., Vanstone, S. A., & Menezes, A. J. (2003). *Guide to elliptic curve cryptography*. Springer.

Harkanson, R., & Kim, Y. (2017). Applications of elliptic curve cryptography: A light introduction to elliptic curves and a survey of their applications. *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, 1-7. https://doi.org/10.1145/3064814.3064818

Higham, N. J. (2002). *Accuracy and stability of numerical algorithms* (2nd ed). Society for Industrial and Applied Mathematics.

Holmes, D. (2016). *TLS Telemetry Report—Tracking Global Internet Encryption Trends* (p. 23). F5 Labs. https://www.f5.com/content/dam/f5-labs-v2/article/articles/threats/7--2017-jan-mar/20170119_2016_tls_report/REPORT-The-2016-TLS-Telemetry-Report.pdf

IdenTrust Global Common Certificate Policy. (2021, marzo). IdenTrust Global COmmon Certificate Policy. https://www.identrust.com/sites/default/files/resources/IGC-CP-v1.5.3_03012021.pdf

JCryptool. (s. f.). https://www.cryptool.org/en/jct/

Koblitz, A. H., Koblitz, N., & Menezes, A. (2011). Elliptic curve cryptography: The serpentine course of a paradigm shift. *Journal of Number Theory*, *131*(5), 781-814. https://doi.org/10.1016/j.jnt.2009.01.006

Koblitz, N. (s. f.). *Elliptic Curve Cryptosystems*. 7. https://doi.org/10.1090/S0025-5718-1987-0866109-5

Kultinov, K. (2019). Software Implementations and Applications of Elliptic Curve Cryptography. *Wright State University*, 89.

Lara-Nino, C. A., Diaz-Perez, A., & Morales-Sandoval, M. (2018). Elliptic Curve Lightweight Cryptography: A Survey. *IEEE Access*, *6*, 72514-72550. https://doi.org/10.1109/ACCESS.2018.2881444

Lenstra, H. W. (1987). Factoring Integers with Elliptic Curves. *The Annals of Mathematics*, *126*(3), 649. https://doi.org/10.2307/1971363

Musson, M. (s. f.). *Attacking the Elliptic Curve Discrete Logarithm Problem*. 154.

Online Gantt. (2022). https://www.onlinegantt.com/#/gantt

Reyes, A. C., Castillo, A. K. V., Morales-Sandoval, M., & Diaz-Perez, A. (2013). A performance comparison of elliptic curve scalar multiplication algorithms on smartphones. *CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing*, 114-119. https://doi.org/10.1109/CONIELECOMP.2013.6525770

SageMath. (s. f.). https://www.sagemath.org/

Shaikh, J. R., Nenova, M., Iliev, G., & Valkova-Jarvis, Z. (2017). Analysis of standard elliptic curves for the implementation of elliptic curve cryptography in resource-constrained E-commerce applications. *2017 IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS)*, 1-4. https://doi.org/10.1109/COMCAS.2017.8244805

Silverman, J. H. (2009). *The Arithmetic of Elliptic Curves* (Vol. 106). Springer New York. https://doi.org/10.1007/978-0-387-09494-6

Sonmez Turan, M., McKay, K., Chang, D., Calik, C., Bassham, L., Kang, J., & Kelsey, J. (2021). *Status Report on the Second Round of the NIST Lightweight Cryptography Standardization Process*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8369

Usage statistics of SSL certificate authorities for websites. (2021, noviembre). Usage Statistics of SSL Certificate Authorities for Websites. https://w3techs.com/technologies/overview/ssl_certificate

Valenta, L., Sullivan, N., Sanso, A., & Heninger, N. (2018). In Search of CurveSwap: Measuring Elliptic Curve Implementations in the Wild. *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 384-398. https://doi.org/10.1109/EuroSP.2018.00034

Whitfield, D., & Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, *IT-22*(6), 644-654.

Wiles, A. (1995). Modular Elliptic Curves and Fermat's Last Theorem. *The Annals of Mathematics*, *141*(3), 443. https://doi.org/10.2307/2118559

Yildirim, E. (2016). The Importance of Information Security Awareness for the Success of Business Enterprises. En D. Nicholson (Ed.), *Advances in Human Factors in Cybersecurity* (Vol. 501, pp. 211-222). Springer International Publishing. https://doi.org/10.1007/978-3-319-41932-9_17

Anexo A – Guía de instalación

Los siguientes pasos corresponden a una instalación sobre la plataforma Linux (Ubuntu 20.04), aunque el procedimiento a seguir para otros sistemas operativos es muy similar (Windows: https://docs.python.org/es/3/using/windows.html, y para Mac: https://docs.python.org/es/3/using/mac.html):

1. Descargar el código:
    1. Abrir un navegador y descargar el archivo zip (https://github.com/omarsoftware/ecc/archive/refs/heads/master.zip) y descomprimirlo en el directorio de su preferencia.
    2. Si posee git, puede clonar el repositorio abriendo una terminal y ejecutando el siguiente comando parado en el directorio de su preferencia:

       git clone https://github.com/omarsoftware/ecc.git
2. Asegurarnos de tener instalado Python 3.8+:
    1. En la terminal, por línea de comando ejecutar:
        1. *sudo apt update*
        2. *sudo apt -y upgrade*
        3. *python3 -V*
    2. El resultado por pantalla debería ser similar a: *Python 3.8.x*
    3. En caso de no haber obtenido este resultado, consultar este link (https://docs.python.org/es/3/using/unix.html) para instalar Python o una versión más actualizada del mismo en la plataforma correspondiente.
3. Instalar el gestionador de paquetes pip:
    1. *sudo apt install -y python3-pip*
    2. *sudo apt install -y build-essential libssl-dev libffi-dev python3-dev*
4. Creación de entorno virtual:
    1. *sudo apt install -y python3-venv*
    2. Pararse en el directorio raíz del proyecto descargado en el paso 1, y crear un entorno virtual *my_env* con el comando:

       python3 -m venv my_env
    3. Para activar al entorno virtual, ejecute:

       source my_env/bin/activate
    4. Notará que el nombre de su terminal cambiará por el de su entorno virtual (“my_env”).

5. Instalación de dependencias
    1. Estando dentro del entorno virtual, ejecute:

       pip install -r requirements.txt
6. Dentro del entorno virtual, ejecute:

   python main.py
7. La aplicación se abrira y podrá interactuar con ella