Universidad de Buenos Aires

Facultades de Ciencias Económicas,

Cs. Exactas y Naturales e Ingeniería

Maestría en Seguridad Informática

Tema:

Criptografía

Título:

Implementación de OTP usando el protocolo Diffie Hellman basado en curvas elípticas (ECDH)

Autor: Ing. Rodrigo Horacio Zapata Valdez

Tutor: Mg. Ing. Juan Alejandro Devincenzi

Año 2014

Cohorte 2012

Declaración Jurada de origen de los contenidos

“Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual”

Resumen

Desde tiempos inmemoriales la comunicación entre los seres humanos es una necesidad que fue logrando distintas alternativas de solución a medida del transcurso de los años y la evolución del conocimiento.

La problemática de que los mensajes emitidos solo fueran comprendidos por los destinatarios, impulsó la creación y desarrollo de la criptografía. Ciencia que nació y se fortaleció en el ámbito militar, pero que desde hace unas décadas juega un papel importante en la vida cotidiana, en especial con el auge de Internet.

A lo largo del trabajo se desarrollarán los conceptos necesarios para poder construir un sistema semánticamente seguro. Sabiendo que el sistema de cifrado denominado One Time Pad ofrece la propiedad de secreto perfecto, y que su implementación es algo compleja en términos prácticos, se construirá un sistema con ciertas características del OTP que ofrezca la seguridad deseada. Básicamente la gran diferencia se encuentra en el proceso de generación de claves aleatorias, que serán reemplazadas por claves pseudoaleatorias obtenidas a través de un proceso Diffie-Hellman basado en curvas elípticas.

Palabras claves: *criptografía, one time pad, Diffie Hellman, curvas elípticas, seguridad semántica*

TABLA DE CONTENIDO

1 Presentación de la tematica 1
1.1 Formulación de la problemática 1
1.2 Objetivos: 2
1.2.1 Generales 2
1.2.2 Específicos 2
1.3 Hipótesis 3
1.4 Alcance 3
1.5 Estructura del trabajo 3
2 Criptografía 5
2.1 Criptografía asimétrica 6
2.2 Criptografía simétrica 6
2.2.1 One-Time Pad o cifrador de Vernam 7
3 Conceptos de Seguridad 9
3.1 Definiciones y conceptos de seguridad 9
3.1.1 Seguridad Perfecta 9
3.1.2 Conceptos asociados a la Seguridad Perfecta 10
3.1.3 Seguridad computacional 12
3.2 Conceptos básicos de la seguridad 16
3.2.1 Seguridad semántica 18
3.2.2 Pseudoaleatoriedad 19
4 Introducción a la criptografía de curvas elípticas 20
4.1 Problema del logaritmo discreto de curvas elípticas (PLDCE) 22
4.1.1 Longitud de la clave 22
4.2 Elección del campo finito 23
4.3 Estándares relacionados 23
4.4 Dominio de parámetros 24
4.5 Requerimientos de la curva elíptica 25
4.6 Ataques conocidos contra las curvas elípticas 27
4.6.1 Representación de los puntos de una curva elíptica 29
5 Establecimiento de claves compartidas 31
5.1 Intercambio de claves Diffie-Hellman 31
5.2 Algoritmo de establecimiento de clave Diffie-Hellman 31
5.3 Protocolo Diffie-Hellman de Curva Elíptica 32

5.3.1 Validación de claves públicas 33
6 Desarrollo del sistema 34
6.1 Introducción 34
6.2 One-Time Pad modificado 34
6.2.1 Funcionamiento del sistema 35
6.3 Análisis de indistinguibilidad del sistema 36
6.3.1 Construcción criptográfica con un generador pseudoaleatorio 36
6.3.2 Teorema 37
6.3.3 Generación de claves pseudoaleatorias en ECDH 39
6.3.4 Análisis de indistinguibilidad del OTP modificado 39
7 El lenguaje Java 41
7.1 Justificación de elección del Lenguaje Java 41
7.2 Características criptográficas de Java 43
7.2.1 Proveedores Criptográficos 45
7.2.2 Proveedores de terceros 48
7.2.3 Justificación elección de Proveedor Criptográfico 54
8 Implementación del sistema 56
8.1 Instalación de los archivos de políticas de JCE 56
8.2 Instalación y configuración del proveedor Criptográfico FlexiProvider 57
8.3 Descripción del programa 58
8.3.1 Pseudoaleatoriedad y generación de claves en Java 62
9 Conclusiones 66
10 Bibliografía 69
Anexo A: Aritmética Modular 75
A.1 Estructuras algebraicas 76
A.1.2 Grupos 76
A.2 Campos Finitos 77
A.2.1 Campo primo $\mathbb{F}\boldsymbol{p}$ 78
A.2.2 Campos binarios $\mathbb{F}\boldsymbol{2m}$ 79
A.2.3 Subcampo 79
Anexo B: Curvas elípticas 80
B.1 Estructura de grupo (caso general) 82
B.1.1 Curvas elípticas sobre campos $\mathbb{F}\boldsymbol{p}$ 83
B.1.2 Curvas elípticas sobre el campo $\mathbb{F}\boldsymbol{2m}$ 87
B.2 Algunas propiedades básicas 88

Anexo C: Parámetros recomendados para la curva secp521r1 89
Anexo D: Codigos de porgramas de prueba 90
D.1 Librería Bouncy Castle 90
D.2 Librería Flexi Provider 91
D.3 Librería SunEC 92

Lista de figuras

Figura 1: Prueba de seguridad por reducción 16
Figura 2: Diffie Hellman de Curvas Elípticas..... 32
Figura 3: Esquema propuesto 34
Figura 4: Construcción con generador pseudoaleatorio 37
Figura 5: Proveedores criptográficos en Java 45
Figura 6: Archivos de extensión de políticas..... 56
Figura 7: Proveedores criptográficos 57
Figura 8: Interfaz gráfica Cliente-Servidor 59
Figura 9: Diagrama de Flujo General..... 61
Figura 10: Diagramas de flujos varios 61
Figura 11: Adición de puntos 82
Figura 12: Perspectiva geométrica de la suma de dos puntos distintos 84
Figura 13: Perspectiva geométrica de la suma de dos puntos iguales 84
Figura 14: Perspectiva geométrica del doble de un punto: P+P=R, si $\boldsymbol{yp} \neq 0$ 85
Figura 15: Perspectiva geométrica de la suma de dos punto iguales $\boldsymbol{P} + (-\boldsymbol{P}) = \boldsymbol{\mathcal{O}}$ 86

Lista de Tablas

Tabla 1: Tamaños de claves recomendados por NIST 20
Tabla 2: Nivel de Seguridad de ECC..... 26
Tabla 3: Curvas elípticas disponibles en SunEC - Campos primos Fp 47
Tabla 4: Curvas elípticas disponibles en SunEC - Campos Binarios 48
Tabla 5: Curvas elípticas disponibles en Bouncy Castle - Campos primos Fp..... 51
Tabla 6: Curvas elípticas disponibles en Bouncy Castle - Campos binarios..... 52
Tabla 7: Tabla 4: Curvas elípticas disponibles en FlexiProvider - Campos primos Fp 53
Tabla 8: Curvas elípticas disponibles en FlexiProvider - Campos Binarios 54
Tabla 9: Tiempo de ejecución Bouncy vs Flexi Provider 55

ABREVIATURAS

AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Program Interface
ASN	Abstract Syntax Notation One
BSGS	Baby Step, Giant Step
CA	Certificate Authority
CBC	MAC chaining message authentication Code
CRL	Certificate Revocation List
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
ECMQV	Elliptic Curve Menezes-Qu-Vanstone
ECNR	Elliptic Curve Nyberg Rueppel Signature Scheme
J2SE	Java 2 Platform, Standard Edition
JCA	Java Cryptography Architecture
JCE	Java Cryptograhpy Extension
FIPS	Federal Information Processing Standard
FNW	Forma Normal de Weierstrass
GNU	GPL General Public License
HMAC	Hash Message Authentication Code
IEEE	Institute of Electrical and Electronics Engineers
ITAR	International Traffic in Arms Regulations
JDK	Java Development Kit
JVM	Java Virtual Machine
MAC	Message Authentication Code

MD5	Message-Digest Algorithm 5
MOV	Menezes-Okamoto-Vanstone
MQV	Menezes–Qu–Vanstone
NIST	National Institute of Standards and Technology
OTP	One-Time Pad
PGP	Pretty Good Privacy
PKCS	Public-Key Cryptography Standards
PKI	Public Key Infrastructure
PLD	Problema del Logaritmo Discreto
PLDCE	Problema del Logaritmo Discreto de Curvas Elípticas
PPT	Probabilistic Polynomial Time
RSA	Rivest, Shamir y Adleman
SEC	Standards for Efficient Cryptography
SHA	Secure Hash Algorithm
TLS	Transport Layer Security

1 PRESENTACIÓN DE LA TEMATICA

Criptografía (del griego, kryptos y graphein, literalmente “escritura oculta o secreta”) en un principio se refería al arte de cifrar un mensaje, es decir, escribirlo de una manera no entendible para alguien que no conozca ese proceso de cifrado. Un mensaje transformado por un proceso de cifrado se llama criptograma o texto cifrado, en contraste con un mensaje no-cifrado llamado texto en plano. La operación que recupera el texto en plano de un texto cifrado se denomina proceso de descifrado.

Actualmente la criptografía es una disciplina amplia cuyo propósito es proveer características de seguridad para las comunicaciones. El criptoanálisis es el enfoque opuesto a la criptografía, ya que intenta romper los procesos de cifrado, es decir, recuperar texto plano de un texto cifrado sin un conocimiento a priori del proceso de descifrado. El estudio de la criptografía va acompañado con un estudio del criptoanálisis correspondiente, juntos forman parte de la criptología, la ciencia de lo secreto.

1.1 Formulación de la problemática

Los seres humanos siempre han sentido la necesidad de ocultar información, mucho antes de que los avances tecnológicos dieran lugar a los primeros equipos informáticos.

La protección de la información sensible contra el acceso no autorizado o cambios fraudulentos ha sido una preocupación fundamental a través de la historia. La criptografía moderna surgió como una solución a la imperiosa necesidad de poder transmitir de manera segura códigos militares durante la Segunda Guerra Mundial. Desde entonces, esta área de la criptología ha experimentado un crecimiento muy importante, aun cuando la mayor parte de estos avances se mantienen en secreto.

Cabe destacar que hasta el año 1996, la exportación de software criptográfico se encontraba bajo la regulación del tráfico internacional de armas (ITAR, por sus siglas en inglés). A fines de ese año, se transfirió al departamento de comercio bajo las regulaciones de la administración de las exportaciones.

En los últimos años, importantes investigaciones en numerosas universidades de todo el mundo han logrado que la criptografía sea una ciencia al alcance de todos, y que se haya convertido en la piedra angular de áreas tan importantes como el comercio electrónico, la telefonía móvil, las redes de computadoras y la banca electrónica, entre otras. La seguridad de la información en estos casos es fundamental, sobre todo para una sociedad que tiende al uso cada vez más frecuente de medios electrónicos para el procesamiento, almacenamiento y transporte de datos, información y conocimientos.

El creciente uso de redes públicas, como Internet, para el envío y recepción de información desde y hacia cualquier lugar del mundo, impone también fuertes requerimientos de seguridad para el transporte de dicha información. Debido a la creciente necesidad de nuevas soluciones prácticas, se propone la implementación de un sistema de intercambios de mensajes con elevados niveles de seguridad, utilizando el sistema de cifrado de flujo de Vernam o llamado OTP por sus siglas en inglés (One-Time Pad), y utilizando como generador de claves de cifrado el protocolo ECDH (Elliptic Curve Diffie-Hellman) en cada intercambio de mensajes.

Dicha propuesta se basa en aprovechar la imposibilidad de quebrar el método OTP, fortaleciendo aún más la seguridad, con la complejidad que le otorga el uso de curvas elípticas al protocolo de intercambio de mensajes Diffie Hellman, teniendo en cuenta que en la bibliografía pública no existen una implementación similar con su correspondiente análisis teórico.

1.2 Objetivos:

1.2.1 Generales

- Desarrollar e implementar un software que permita el intercambio de mensajes entre dos computadoras usando los principios de One-Time Pad y de intercambios de claves Diffie Hellman basado en Curvas Elípticas

1.2.2 Específicos

- Realizar una investigación sobre los fundamentos en los que basan su funcionamiento los algoritmos que emplean curvas elípticas.

- Analizar y describir los principios que demuestran la seguridad del protocolo One-Time Pad.
- Investigar sobre las bibliotecas y APIS criptográficas del lenguaje de programación Java.

1.3 Hipótesis

El desarrollo e implementación que utilice generación e intercambio de claves Diffie Hellman basado en ECDH en conjunto con OTP en cada transmisión de mensaje permite lograr un sistema de comunicación con seguridad semántica.

1.4 Alcance

La principal finalidad de la investigación es el desarrollo e implementación de un mecanismo de intercambios de mensajes para ser utilizado en entornos que requieran altos niveles de seguridad. Se enfocará principalmente en asegurar la confidencialidad de los mensajes transmitidos, quedando fuera del alcance la integridad y autenticación.

1.5 Estructura del trabajo

El trabajo presenta una estructura modular dividida en capítulos, a través de los cuales se van introduciendo los conceptos fundamentales que serán necesarios para entender la construcción del algoritmo One-Time Pad con intercambios de claves usando Diffie-Hellman basado en curvas elípticas.

En el capítulo 2 se dan las nociones básicas para entender un sistema criptográfico y poder diferenciar los dos tipos de enfoque: simétrico y asimétrico. Además se da una idea general del algoritmo base del proyecto: el One-Time Pad, haciendo referencia a su funcionamiento y a las desventajas que posee.

En el capítulo 3 comienza a hablarse de la noción de seguridad introduciéndose distintos conceptos (seguridad perfecta, computacional) hasta llegar al concepto de seguridad semántica, que es el tipo de seguridad que se quiere probar que posee el sistema criptográfico objetivo del trabajo.

En el capítulo 4 tiene su argumento en las curvas elípticas, justificando su uso en la criptografía y se toma como referencia teórica los Anexos A: Aritmética modular y Anexos B: Curvas elípticas. Se analiza qué parámetros son necesarios para definir una curva y los ataques a los cuales puede verse comprometida.

En el capítulo 5 se trata el tema del intercambio de claves en los sistemas criptográficos y cómo puede aplicarse la fortaleza de las curvas elípticas en su implementación.

En el capítulo 6 se presenta formalmente el protocolo objetivo del trabajo, se dan las pautas para su implementación y se prueba usando todos los conceptos desarrollados en los capítulos previos que tiene seguridad semántica (usando el teorema de equivalencia con la indistinguibilidad).

En el capítulo 7 se describe el lenguaje Java y se justifica su elección como plataforma para implementar en forma práctica el protocolo OTP con Diffie Hellman. Se muestra la estructura criptográfica de Java y el uso de los llamados proveedores o librerías.

En el capítulo 8 tiene lugar la implementación del sistema en el lenguaje Java, mostrando requisitos previos, configuración del entorno y características propias del sistema implementado.

Por último se presentan las conclusiones que se derivan de la realización del trabajo.

2 CRIPTOGRAFÍA [1]

La criptografía se ocupa del estudio de técnicas que permitan brindar algunos servicios de seguridad, en especial la confidencialidad. Esto se logra a través del diseño y uso de criptosistemas.

Puede definirse un criptosistema como la quíntupla (M, C, K, E, D), compuesta de 3 espacios o conjuntos finitos y de dos conjuntos de funciones:

- M representa el conjunto de todos los posibles mensajes sin cifrar llamados textos en claro.
- C representa el conjunto de todos los mensajes cifrados o criptogramas.
- K representa el conjunto de claves que se pueden emplear en el criptosistema
- E es el conjunto de transformaciones de cifrado o familia de funciones que se aplica a cada elemento de M para obtener un elemento de C. Existe una transformación diferente E_k para cada valor posible de la clave k.
- D es el conjunto de transformaciones de descifrado, similar a E.

El texto en claro se refiere al mensaje que el emisor desea enviar y el texto cifrado es el mensaje resultante del proceso de cifrado que es recibido por el destinatario. Se debe cumplir la siguiente condición:

Para cualquier texto plano $m \in M$ y cualquier función de cifrado $Enc \in E$, donde $Enc(ek, m) = c$ con $c \in C$, debe existir una función de descifrado $Dec \in D$ tal que $Dec\big(dk, Enc(ek, m)\big) = m$ y $Dec(dk,\cdot) = Enc^{-1}(ek,\cdot)$, donde ek, $dk \in k$ son llaves que indexan o definen las funciones de cifrado y de descifrado.

Si en conjunto con el par de funciones $(Enc(ek,\cdot), Dec(dk,\cdot))$ para valores de $ek\ y\ dk \in K$, se toma también el método para obtener ambas claves, se obtiene lo que se denomina un esquema de cifrado. Todo esquema de cifrado se compone entonces de tres algoritmos: un generador de llaves (Gen), un cifrador $Enc(ek,\cdot)$ y un descifrador $Dec(dk,\cdot)$. Según el principio de Kerckoffs[1], todo excepto las claves dentro del sistema es públicamente conocido.

[1] Auguste Kerckhoffs (1835 - 1903) fue el primero en publicar ensayos sobre criptografía militar en la Revista de Ciencias Militares francesa. Se concretaron en los seis principios básicos para el correcto diseño de sistemas criptográficos [73].

Existen dos ramas de la criptografía: la simétrica (o de clave privada) y la asimétrica (o de llave pública).

2.1 Criptografía asimétrica

Es aquella que utiliza criptosistemas en los cuales se tienen un par de claves distintas ek y dk tal que para cualquier valor $m \in M$ se cumple que $Dec(dk, \cdot) = Enc^{-1}(ek, \cdot)$. Este par de claves se denominan pública y privada respectivamente y como su nombre lo indica la llave pública ek debe ser accesible para cualquier entidad que quiera establecer una comunicación segura con el dueño de dicha clave, mientras que el acceso a la clave privada se restringe únicamente al propietario. Ofrecen un abanico superior de posibilidades, pudiendo emplearse para establecer comunicaciones seguras por canales inseguros para llevar adelante autenticaciones. En esta categoría se encuentra por ejemplo el acuerdo de claves Diffie-Hellman que será tratado con detalle en el Capítulo 5.

2.2 Criptografía simétrica

Es aquella que utiliza criptosistemas en los que la clave de cifrado ek y la clave de descifrado dk son iguales, por lo que cuando se habla de cifrado simétrico solamente se hace referencia al cifrado k. Presentan el problema de que para ser empleados en comunicaciones la clave k debe estar tanto en el emisor como en el receptor, lo que deja abierto el interrogante de cómo transmitir la clave en forma segura.

Dentro del cifrado simétrico existen dos categorías, que se dividen de acuerdo a la forma en la que se leen el texto plano para convertirlo en texto cifrado. En primer lugar se tiene el cifrado por flujo donde el mensaje de entrada leído y cifrado es un símbolo (bit o byte). La salida del cifrado depende de la llave, del texto plano y de un estado que define qué transformación se aplica. Son los más eficientes ya que las operaciones que utilizan sus transformaciones son simples y no requieren grandes cantidades de memoria y de espacio. En esta categoría se encuentra el sistema One-Time Pad que se describirá en a continuación.

La segunda categoría es el cifrado por bloques donde el cifrado lee consecutivamente bloques de símbolos (bits) de una longitud fija n para cifrarlos.

Este tipo de cifradores requieren más recursos computacionales pues necesitan hacer uso de memoria y las transformaciones suelen ser menos sencillas.

Un esquema de cifrado por bloques puede verse como una función que mapea elementos de $m \in M$ de longitud n a elementos $c \in C$, también de longitud n.

2.2.1 One-Time Pad o cifrador de Vernam

En 1917, Gilbert Sandford Vernam patentó un cifrador que consiguió alcanzar la propiedad de secreto perfecto. No existieron pruebas para demostrar este hecho en aquel tiempo, pues no estaba definida la noción de secreto perfecto. Aproximadamente 25 años después, Claude Shannon, introdujo el concepto y demostró que el One-Time Pad (también llamado cifrador de Vernam) alcanzaba este nivel de seguridad.

Sea $a \oplus b$ la operación bit a bit or-exclusiva (XOR) de dos cadenas binarias a y b (por ejemplo si $a = a_1, \dots, a_\ell$ y $b = b_1, \dots, b_\ell$, entonces $a \oplus b = a_1 \oplus b_l, \dots, a_\ell \oplus b_\ell$). Entonces el esquema de cifrado es definido como:

- Dado un entero $\ell > 0$. Luego el espacio de mensaje M, el espacio de claves K y el espacio de textos cifrados C, son todos iguales a $\{0,1\}^\ell$ (por ejemplo todo el conjunto de todas las cadenas de tamaño ℓ).
- El algoritmo de generación de claves Gen trabaja eligiendo una cadena de $K = \{0,1\}^\ell$ de acuerdo a una distribución normal (por ejemplo, cada una de las 2^ℓ cadenas en el espacio son elegidas como claves con la probabilidad exactamente igual a $2^{-\ell}$).
- El algoritmo Enc opera de la siguiente manera: dada una clave $k \in \{0,1\}^\ell$ y un mensaje $m \in \{0,1\}^\ell$, el valor de la salida es $c \coloneqq k \oplus m$ (esta notación significa que el algoritmo es determinista, para distinguirlo del caso probabilístico).
- El proceso Dec trabaja como se sigue: dada una clave $k \in \{0,1\}^\ell$ y un texto cifrado $c \in \{0,1\}^\ell$, el valor de la salida es $m \coloneqq k \oplus c$.

El One-Time Pad constituye un esquema de cifrado ya que $\forall\, k, m$ se tiene $Dec_k\big(Enc_k(m)\big) = k \oplus k \oplus m$.

Intuitivamente, el One-Time Pad cumple las condiciones del secreto perfecto (concepto que se verá más adelante), ya que dado un texto cifrado c, no existe manera que un adversario pueda conocer cuál es el texto plano m que lo originó.

Para verificar que esta afirmación es verdadera, se debe notar que para cada posible m existe una clave k tal que $c = Enc(k, m)$, es decir, $k = m \oplus c$. Además, cada clave es elegida con una probabilidad uniforme y no existe clave que tenga mayor probabilidad que otra. Combinando lo anterior, se afirma que c no revela nada acerca de cuál texto plano m fue cifrado, porque cada texto plano es igualmente probable de haber sido cifrado (teniendo en cuenta que el largo de k es información que el adversario no posee).

2.2.1.1 Desventajas del One-Time Pad

La clave en el esquema de cifrado One-Time Pad debe ser tan larga como la extensión del mensaje. Esto limita la aplicabilidad del sistema si se quiere enviar mensajes extensos (resulta dificultoso almacenar una clave de gran tamaño) o bien no se conoce el límite superior del tamaño del mensaje.

Por otro lado, se tiene que One-Time Pad, como su nombre lo indica es solamente seguro si es usada la clave una sola vez. Se debe tener en cuenta que al cifrar más de un mensaje con la misma clave, se está ofreciendo algo de información al adversario. En particular, sean dos mensajes m, m' son cifrados usando la clave k, un adversario que obtiene $c = m \oplus k$ y $c' = m' \oplus k$ puede computar:

$$c \oplus c' = m \oplus m'$$

Y puede aprender algo acerca de la función or-exclusiva de los dos mensajes.

Otro inconveniente que presenta la clave debe ser totalmente aleatoria elegida mediante un proceso de selección que lo garantice.

Además se tiene que One-Time Pad es un esquema seguro contra sólo ataques de texto cifrado. Por ejemplo el esquema es inseguro contra ataques de texto conocido. Un adversario quien obtuvo el cifrado c de un mensaje conocido, puede calcular la clave k=$c \oplus m$ y descifrar cualquier mensaje cifrado usando esa clave.

3 CONCEPTOS DE SEGURIDAD

El término seguridad proviene de latín “securitas”. Según el diccionario de la Real Academia Española, se define como “libre y exento de todo peligro, daño o riesgo”. Se puede decir que el término tiene diversos sentidos dependiendo del campo en el que se esté tratando.

3.1 Definiciones y conceptos de seguridad

Se definirán algunos modelos de seguridad que pueden ser provistos por los esquemas de cifrado. Existen una vasta variedad de tipos, algunos de ellos similares entre sí que llegan incluso a ser equivalentes.

3.1.1 Seguridad Perfecta [2]

Según se vio en el Capítulo 2, un esquema de cifrado está definido por tres algoritmos (Gen, Enc, Dec) que operan sobre los conjuntos de $K, M\ y\ C$. La distribución de probabilidad sobre K es simplemente la definida ejecutando la función Gen y tomando la salida. Para $k \in K$ se tiene que $Pr[K = k]$ denota la probabilidad que la clave generada por Gen sea k. Del mismo modo para $m \in$ M, se tiene $Pr[M = m]$ denota la probabilidad que un mensaje enviado sea igual a m.

Se debe asumir que las probabilidades sobre K y M son independientes, es decir, que las claves y los mensajes son independientes.

Para $c \in E$ se tiene que $Pr[\ C = c]$ indica la probabilidad de que un texto cifrado sea igual a c.

Para definir intuitivamente la noción de secreto perfecto, considerar que un adversario conoce la probabilidad de distribución sobre m, esto es, conoce la probabilidad que diferentes mensajes sean enviados. Luego el adversario observa algunos textos cifrados que están siendo enviados de una parte a otra. Idealmente la observación de estos textos cifrados no tendría efecto sobre el conocimiento del adversario, es decir, la probabilidad a posteriori que algunos mensajes m están siendo enviados no debería ser diferente de la probabilidad a priori que un mensaje m sería enviado. Además esto se acompaña con el hecho de que el adversario no cuenta con poder computacional ilimitado. Esto significa que un texto cifrado no

revela nada acerca del texto plano y un adversario que interceptare un texto cifrado no sabría absolutamente nada acerca de ese texto plano.

3.1.1.1 Teorema de Shannon [2] [3]

Sean (Gen, Enc, Dec) un esquema de cifrado sobre un espacio de mensajes M tal que $|M| = |K| = |C|$. Este esquema cumple con la propiedad de secreto perfecto si y solo si:

I. Cada clave $k \in K$ debe ser elegida con la misma probabilidad $1/|K|$ por el algoritmo generador de llaves Gen.

II. Para cada mensaje $m \in M$ y cada texto cifrado $c \in C$, existe una sola clave $k \in K$, tal que $Enc_k(m) = c$.

Otra definición tratada en [2]**:**

Un esquema de cifrado (Gen, Enc, Dec) sobre un espacio de mensaje M es perfectamente secreto si para toda distribución de probabilidades sobre M, todo mensaje $m \in M$ y todo texto cifrado $c \in C$ para la cual la $Pr[C = c] > 0$:

$$Pr[M = m \mid C = c] = Pr[M = m]$$

El esquema de cifrado One-Time Pad cumple con la propiedad de secreto perfecto, puede verse la demostración en [2].

3.1.2 Conceptos asociados a la Seguridad Perfecta [2]

3.1.2.1 Indistinguibilidad perfecta

Consiste en una formulación equivalente a la de secreto perfecto. Se basa en el hecho que la distribución de probabilidad a través de C es independiente del texto plano. Sea $C(m)$ la distribución del texto cifrado cuando el mensaje que está siendo cifrado es $m \in M$ (esta distribución depende de la elección de k). Entonces para todo m_0, $m_1 \in M$, la distribución $C(m_0)$ y $C(m_1)$ son idénticas. Esta simplemente es otra forma de afirmar que el texto cifrado no contiene información sobre el texto plano. El concepto de indistinguibilidad perfecta implica que es imposible distinguir el cifrado m_0 del cifrado de m_1. Formalmente se define:

[2] Claude Elwood Shannon (1916 - 2001) es considerado como el padre fundador de la era de las comunicaciones electrónicas. Fue ingeniero matemático estadounidense, cuyo trabajo sobre los problemas técnicos y de ingeniería dentro de la industria de las comunicaciones, fijaron las bases tanto para la industria de la informática como de las telecomunicaciones [72].

Un esquema de cifrado (Gen, Enc, Dec) definido sobre un espacio de mensajes M es perfectamente secreto si y sólo si para toda distribución de probabilidad sobre M, todo m_0 y $m_1 \in M$ y todo texto cifrado $c \in C$:

$$Pr[C = c \mid M = m_0] = Pr[C = c \mid M = m_1]$$

3.1.2.2 Indistinguibilidad del adversario

Es otra definición equivalente a la de secreto perfecto. Está basada en un experimento que involucra a un adversario A y se tiene que A es incapaz de distinguir el cifrado de un texto plano del cifrado de otro, esto se llama indistinguibilidad del adversario. A lo largo del trabajo se usarán los llamados experimentos para definir el concepto de seguridad. Un experimento es un "juego" en el que intervienen un adversario (espía) que intenta romper el esquema de cifrado y un probador o tester que observa si el adversario tiene éxito. La finalidad de definir experimentos es para establecer qué probabilidad tiene el adversario de vencer la seguridad del esquema, a esta probabilidad se le llama ventaja (Adv).

Se define el experimento $PrivK^{esp}$, considerando un esquema de cifrado de clave privada y un adversario espía (es espía o fisgón[3]) que sólo recibe un texto cifrado c y luego intenta determinar algo acerca del texto plano. El experimento es definido sobre cualquier esquema de cifrado $\Pi = (Gen, Enc, Dec)$ a través de un espacio de mensaje M y para cualquier adversario A. Se define $PrivK^{esp}_{A,\Pi}$ la ejecución del experimento dado Π y A.

I. El adversario A observa un par de mensajes $m_0, m_1 \in M$
II. Una clave k es generada ejecutando Gen, y un bit aleatorio $b \leftarrow \{0,1\}$ es elegido por una entidad imaginaria que está ejecutando el experimento junto con A. Luego, un texto cifrado $c \leftarrow Enc_k(m_b)$ es calculado y dado a A.
III. A tiene como salida el b'.
IV. La salida del experimento es definida 1 si $b' = b$, y 0 de otra manera. Se tendrá $PrivK^{esp}_{A,\Pi} = 1$ si la salida es 1 y en este caso se dice que A tuvo éxito.

[3] Fisgón: según la RAE, aficionado a husmear

Se puede observar que A tiene éxito cuando su b' es correcto, tiene una probabilidad de ½ si adivina b´ al azar. Se debe destacar que no existe limitación en cuanto al poder de cómputo de A. Entonces puede definirse:

Un esquema de cifrado $\Pi = (Gen, Enc, Dec)$ sobre un espacio de mensaje M es perfectamente secreto por cada adversario A, si se sostiene que:

$$Pr\left[PrivK_{A,\ \Pi}^{esp} = 1\right] = 1/2$$

3.1.3 Seguridad computacional [1] [2]

El concepto surgió para resolver algunos inconvenientes relativos e inherentes a la seguridad perfecta en cuestiones prácticas, los métodos de cifrado de acuerdo a este término, permiten cifrar mensajes de gran tamaño (mega o gigabytes) con claves pequeñas (cientos de bits).

El atributo de computacional se debe a que se establecen determinados límites en cuanto al poder computacional del adversario (principalmente el atributo de tiempo). Lo primordial en esta idea de seguridad es que un esquema de cifrado puede ser quebrado, pero el tiempo que se requiere para conseguirlo es demasiado grande, en términos de muchos años equivalentes a varios ciclos de vida humana.

Este tipo de seguridad podría considerarse como más débil que la anterior, debido a que los esquemas de cifrado pueden ser vulnerados, pero en la actualidad es suficiente para propósitos prácticos.

En el capítulo 2 se hizo mención a Kerckhoffs, en este caso se hace referencia a otro de los principios expuestos por el criptógrafo: Un cifrador debe ser prácticamente si no es matemáticamente indescifrable. El concepto de un cifrador computacionalmente seguro (o prácticamente indescifrable), tiene su fundamento en que se obtiene un nivel de seguridad suficiente si el método no puede ser quebrado en un tiempo y con una probabilidad razonable.

Para poder llegar de la idea de seguridad perfecta a la seguridad práctica es necesario suavizar dos requerimientos. En primer lugar se debe considerar que el esquema será seguro sólo en el caso contra adversarios “eficientes” durante una cantidad de tiempo razonable. En segundo término, se debe tomar en cuenta que los adversarios pueden romper potencialmente la seguridad con una probabilidad muy pequeña (tan insignificante que no influye que ocurra).

La seguridad computacional puede ser definida considerando dos puntos de vista diferentes el enfoque concreto y asintótico.

- **Enfoque concreto:** especifica límites exactos sobre la máxima probabilidad de éxito y el tiempo máximo de ejecución del adversario. En particular, se definen esquemas de seguridad (t, ε), donde un adversario ejecutando a lo máximo un tiempo t, tiene como máximo una probabilidad ε. Estos parámetros de seguridad se dan generalmente en potencias de 2, haciendo referencia a los ciclos de procesador. El inconveniente que presenta este enfoque es que es demasiado específico, no se conoce con certeza información sobre qué pasaría si el adversario ejecutara durante el doble de tiempo o la mitad del tiempo especificado. Así mismo no se conoce sobre qué tipo de hardware se considera seguro (personal, supercomputadora, clúster). Lo correcto sería definir cuál es el poder de cómputo del adversario y dar un rango de valores para t y ε, que permitan definir cuando el esquema será seguro.
- **Enfoque asintótico:** tiene mayor flexibilidad. Tanto el tiempo como la probabilidad de éxito son funciones de un parámetro de seguridad, un entero n[4], establecido por los participantes y que se asume que es conocido por el adversario. En este enfoque se equipara la noción de algoritmos eficientes con algoritmos probabilísticos en tiempo polinomial (PPT por sus siglas en inglés). Dadas un par de constantes a y c, el algoritmo se ejecuta en tiempo $a * n^c$ (sobre el parámetro de seguridad n), se considera que los adversarios tienen mucho más poder de cómputo que el resto de los participantes. Al hablar de probabilidad de éxito "insignificante", se hace mención a que la probabilidad es menor a cualquier inverso polinomial en n, es decir, dada una constante c de probabilidad de éxito del adversario es, menor que n^{-c} para grandes valores de n. Una función que crece tan lentamente como cualquier inversa polinomial es llamada despreciable (negligible en inglés). Una función f es "despreciable" si para todo

[4] Para ser consistentes con la convención estándar en teoría de complejidad, donde el tiempo de ejecución de un algoritmo es medido como función del tamaño de su entrada, se utilizará para los participantes de los experimentos el parámetro de seguridad en forma unaria: 1^n.

polinomio $p(\cdot)$ existe un N tal que para todos los enteros $n > N$, se tiene que $f(n) < \frac{1}{p(n)}$.

Se debe destacar que en la mayoría de los casos el parámetro de seguridad n es el que determina el tamaño de la clave empleada y es por ello que en esos casos a mayor tamaño de clave mayor seguridad.

3.1.3.1 Seguridad demostrable

Un esquema de cifrado cuenta con este tipo de seguridad si es posible mostrar de alguna manera que quebrarlo es tan difícil como resolver un problema conocido que se supone difícil (no existe un método de resolverlo en tiempo razonable y no existe prueba de que en realidad lo sea). Por ejemplo se encuentran dentro de esta categoría a la factorización de números enteros grandes y la resolución de logaritmos discretos

Cuando se habla de demostrable, se refiere a igualar con alguna suposición. Este tipo de seguridad puede ser considerada como parte de la definida anteriormente ya que muchos de los métodos de cifrado definidos bajo la seguridad práctica pueden equipararse o igualarse a resolver problemas considerados difíciles, pero sin embargo lo que lo diferencia de la seguridad demostrable es que no existe prueba de dicha equivalencia. Además debe agregarse que los enfoques concreto y asintótico son aplicables a la seguridad demostrable, ya que se prueba que el esquema de cifrado es seguro computacionalmente.

Las pruebas realizadas para demostrar la seguridad son conocidas como reducciones, porque se reduce el problema de analizar un esquema de cifrado completo a analizar una primitiva criptográfica básica. Las pruebas por reducción suponen que una primitiva básica es difícil de resolver y prueban en base a esa suposición que el esquema es seguro, es decir, no se demuestra que un esquema es seguro, sino que se lo reduce a uno que se considera seguro. Estas pruebas muestran como convertir un adversario eficiente A, que posee una probabilidad no despreciable de éxito de quebrar el cifrador, en un algoritmo eficiente A', que tiene éxito en resolver el problema que se consideraba difícil.

Se comienza con la suposición de que algún problema X no puede ser resuelto por algún algoritmo de tiempo polinomial excepto con una probabilidad despreciable.

Los pasos que se siguen para realizar una prueba por reducción son:

I. Establecer un adversario eficiente A (en tiempo polinomial probabilístico) que ataca al cifrador (Π), con una probabilidad $\varepsilon(n)$.

II. Construir un algoritmo eficiente A' que trata de resolver el problema X usando a A como una subrutina. A' no sabe cómo funciona A, lo único que sabe es que A quiere atacar a Π. Entonces dada una instancia x del problema X, el algoritmo A' simula para A una ejecución de Π tal que:

   a. La vista de A cuando actúa como sub rutina de A' es lo más cercana posible a la vista de A cuando interactúa con el cifrador por sí mismo.

   b. Si A tiene éxito en quebrar la instancia del cifrador simulado por A', esto le permitirá a A' resolver la instancia x del problema con probabilidad no despreciable $1/p(n)$.

III. Tomar las dos consideraciones del punto anterior implica que si $\varepsilon\,(n)$ no es despreciable, entonces A' resuelve el problema X con una probabilidad no despreciable $\varepsilon(n)/p(n)$. Ya que A' es eficiente y ejecuta al adversario A de tiempo polinomial como una subrutina, esto implica que un algoritmo eficiente resuelve X con una probabilidad no despreciable, por lo que contradice la suposición inicial sobre X.

IV. Se concluye que dada la suposición sobre el problema X, ningún adversario eficiente puede romper el cifrador con una probabilidad ε que no es despreciable, por lo que Π es computacionalmente seguro.

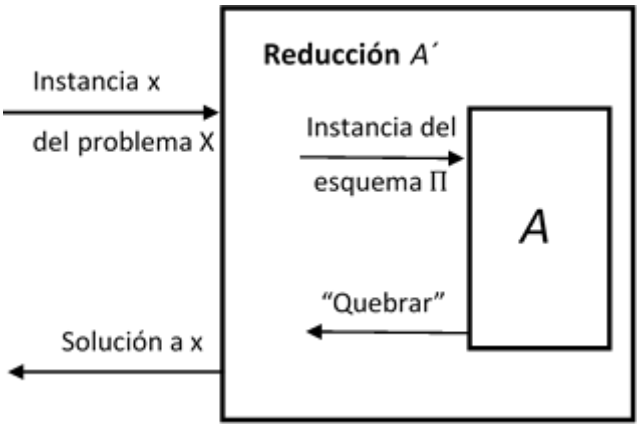

Figura 1: Prueba de seguridad por reducción [2]

Se debe tener en cuenta que las pruebas pueden diseñarse bajo distintos modelos: el modelo estándar se limita al adversario en la cantidad de tiempo y poder computacional (basa su seguridad en alguna suposición difícil de resolver en tiempo polinomial). También existen otros modelos como el oráculo aleatorio y el de conocimiento cero.

Las pruebas dentro del modelo de oráculo aleatorio consideran que los participantes y adversarios tienen acceso a una función aleatoria que actúa como oráculo: responde a cada consulta con una respuesta tomada de forma uniformemente aleatoria dentro del espacio posible de valores.

3.2 Conceptos básicos de la seguridad

Antes de que se inicie la era digital el hombre ya buscaba el concepto de seguridad, pero para fines de este trabajo se enfocará en la seguridad del ámbito computacional. En este ámbito, se consideraba que un criptosistema podía alcanzar dos niveles de seguridad: seguridad semántica y maleabilidad en sentido criptográfico.

La seguridad semántica fue la primera definición formal de seguridad para esquemas de cifrado, los que establecieron esta idea fueron S. Goldwasser y S. Micali en 1982 [4]. Estos autores demostraron [5] que la seguridad semántica es equivalente a atribuir la propiedad de indistinguible, y esta es una definición más sencilla y fácil de probar.

Cuando se habla que un cifrador tiene la característica de ser indistinguible se hace referencia que un adversario que obtiene un texto cifrado $c \in C$ es incapaz de distinguir cual texto plano $m \in M$ lo produjo.

La maleabilidad es una característica que indica que un esquema de cifrado permite que el adversario sea capaz de, dado el texto cifrado c de un mensaje m, transformar c en otro texto cifrado c' cuyo descifrado corresponda al texto plano $f(m)$ para algún tipo de función f conocida, sin tener ningún conocimiento acerca de m.

Para mostrar propiedad de indistinguible pude emplearse el experimento usado en párrafos anteriores con unas ligeras modificaciones:

I. Se debe considerar adversarios ejecutando en tiempos polinomiales.
II. Se concede al adversario que pueda determinar el mensaje cifrado con una probabilidad "insignificantemente" mejor que ½.

Además se deben considerar una serie de diferencias:

I. Se parametriza el experimento con un valor de seguridad n. Se denota $Pr\left[PrivK_{A,\Pi}^{esp}(n) = 1\right]$
II. Se requiere que el adversario tenga como salida dos mensajes m_0, m_1 del mismo tamaño.

Entonces el experimento consta de lo siguiente:

I. El adversario A tiene como entrada 1^n y salida un par de mensajes m_0 y m_1 del mismo tamaño.
II. Una clave k es generada ejecutando $Gen(1^n)$ y un bit aleatorio bit $b \leftarrow \{0,1\}$ es elegido. El texto cifrado $c \leftarrow Enc_k(m_b)$ es calculado y dado a A.
III. A tiene como salida el bit b'.
IV. La salida del experimento es 1 si $b' = b$ y 0 de otro modo. Si $PrivK_{A,\Pi}^{esp}(n) = 1$, se dice que A tuvo éxito.

Si Π es un esquema con un tamaño fijo de valor ℓ el experimento anterior es modificado exigiendo que $m_0, m_1 \in \{0,1\}^{\ell(n)}$. Se define entonces:

Un esquema de cifrado de clave privada $\Pi = (Gen, Enc, Dec)$ tiene un cifrado indistinguible en la presencia de un espía si para todos los adversarios A en tiempo probabilístico polinomial existe una función insignificante (negl) tal que:

$$Pr\left[PrivK_{A,\Pi}^{esp}(n) = 1\right] \leq \frac{1}{2} + negl(n)$$

Donde la probabilidad es tomada, sobre valores aleatorios usados por A así como también los valores aleatorios usados en el experimento (para la elección de la clave, el bit aleatorio b y cualquier valor aleatorio usado en el proceso de cifrado).

3.2.1 Seguridad semántica

Un esquema de cifrado de clave secreta (Gen, Enc, Dec) es semánticamente seguro frente a la presencia de un espía si para todo algoritmo probabilístico en tiempo polinomial A existe un algoritmo probabilístico en tiempo polinomial A' tal que para toda distribución susceptible de muestreo efectivo [5] $X = (X_1, X_2 \dots)$ y para toda función computable de tiempo polinomial f y h, existe una función insignificante negl tal que:

$$|Pr[A(1^n, Enc_k(m), h(m)) = f(m)] - Pr[A'(1^n, h(m) = f(m)]| \leq negl(n)$$

Donde m es elegido de acuerdo a la distribución X_n, y las probabilidades son tomadas sobre la elección de m y la clave k y cualquier valor al azar usado por A, A' y el proceso de cifrado.

Notas complementarias:

Al algoritmo A (representa el adversario real) se le da el texto cifrado $Enc_k(m)$, así como también la función histórica $h(m)$, que representa el conocimiento externo sobre el texto plano m que el adversario puede tener. Luego el adversario A intenta adivinar el valor de $f(m)$ [6].El algoritmo A' también intenta adivinar el valor de $f(m)$, pero solo teniendo $h(m)$. Los requerimientos de seguridad se fundamentan en el éxito de A en adivinar $f(m)$ cuando dado el texto cifrado, puede ser emparejado con algún algoritmo A' que no tiene el texto cifrado. Esto es, el texto cifrado $Enc_k(m)$ no revela nada nuevo sobre el valor de $f(m)$.

Esta definición es una formulación robusta sobre la garantía de seguridad de un esquema de cifrado y más convincente que la de indistinguibilidad (que sólo considera dos textos planos y no hace mención a conocimiento externo o distribuciones arbitrarias). Sin embargo es más simple trabajar con la definición de indistinguibilidad y a través del siguiente teorema se pueden ver que son equivalentes:

[5] En ingles efficiently-sampleable

[6] $f(m)$ representa la función del texto plano dado el texto cifrado.

Teorema: Un esquema de cifrado de clave privada tiene la calidad de ser indistinguible en la presencia de un espía si y sólo si es semánticamente seguro en la presencia de un espía.

3.2.2 Pseudoaleatoriedad [1] [2]

Este concepto juega un papel fundamental en la definición del esquema de cifrado propuesto en esta tesis. En términos generales una cadena pseudoaleatoria es aquella que aparenta tener una distribución uniforme siempre y cuando la entidad que lo está analizando corra en tiempos polinomiales. Del mismo modo que el concepto de indistinguibilidad puede ser visto como un concepto computacional (relajado) de la seguridad perfecta, el término de pseudoaleatoriedad es un concepto computacional de aquello que es verdaderamente aleatorio.

La pseudoaleatoriedad refiere a una distribución sobre cadenas, y cuando se dice que una distribución D sobre cadenas de tamaño ℓ es pseudoaleatoria, significa que D es indistinguible de una distribución uniforme sobre cadenas de tamaño ℓ .

Más precisamente es imposible para algún algoritmo de tiempo polinómico decidir si una cadena es muestreada de acuerdo a una distribución D o una cadena de ℓ bits elegidos al azar.

Definición:

Sea $\ell(\cdot)$ polinomial y sea G un algoritmo determinístico de tiempo polinomial tal que para cualquier entrada $s \in \{0,1\}^n$, la salida del algoritmo G es una cadena de longitud $\ell(n)$. Se dice que G es un generador pseudoaleatorio si se cumplen las siguientes dos condiciones:

I. (Expansión) Para cada n se tiene que $\ell(n) > n$
II. (Pseudoaleatoriedad) Para todo observador D que opera en tiempo tiempo polinomial, existe una función negl tal que.

$$\left|Pr[D(r) = 1] - \Pr(D\big(G(s)\big) = 1\right| \leq negl(n)$$

Donde r es elegido uniformemente en forma aleatoria del conjunto $\{0,1\}^{l(n)}$, la semilla s es elegida uniformemente en forma aleatoria del conjunto $\{0,1\}^n$, y las probabilidades son tomadas sobre elementos aleatorios usados por D y la eleción de r y s.

4 INTRODUCCIÓN A LA CRIPTOGRAFÍA DE CURVAS ELÍPTICAS

Dos sistemas criptográficos pueden ser comparados en cuanto a su eficiencia y rapidez, teniendo como referencia la dificultad que conlleva realizar dos operaciones sobre un algoritmo: se tiene una operación “hacia adelante” que debe ser tratable operacionalmente y su operación inversa, que en cuestiones prácticas debería ser intratable.

Uno de los factores que dificultan estas operaciones es la longitud de las claves usadas, ya que mientras mayor sea, se torna más complicado efectuar las operaciones: para la operación inversa este crecimiento es sub-exponencial, y para la operación directa se tiene un crecimiento lineal.

A partir de lo expresado anteriormente, un sistema criptográfico será más seguro que otro si posee una clave de mayor longitud. Se debe tener en cuenta que al incrementar el tamaño de las claves, se requieren mayores recursos (poder de cómputo, consumo de energía, espacio de memoria, etc). [6]

Para el caso de las implementaciones de ECC, se pueden considerar claves con una longitud más pequeña, ofreciendo el mismo nivel de seguridad que otros sistemas criptográficos ofrecen pero estos lo hacen empleando longitudes de mayor tamaño, esto queda de manifiesto en la comparación efectuada en la Tabla 1.

Tamaño de clave simétrica (bits)	Tamaño de clave RSA y Diffie-Hellman (bits)	Tamaño de clave Curvas Elípticas (bits)
80	1024	160
112	2048	224
128	3072	256
92	7680	384
256	15360	521

Tabla 1: Tamaños de claves recomendados por NIST [7]

Algunos sistemas criptográficos (Diffie-Hellman, RSA, MQV, ECC) realizan operaciones sobre Grupos, confiando su seguridad en la resolución del logaritmo discreto. Lo que los hace diferente entre sí, es la manera en que los elementos del Grupo son definidos y el modo en que las operaciones son llevadas a cabo.

Puede decirse entonces que la seguridad que ofrece ECC con tamaños de claves más pequeñas, tiene su origen en la definición de las operaciones en la curva elíptica: se usa la curva elíptica para definir miembros del conjunto sobre el cual el grupo es calculado y las operaciones matemáticas que se realizarán sobre el grupo.

Para entender el concepto anterior es pertinente afirmar que mientras Diffie-Hellman usa campos finitos, cuyos elementos son números enteros primos, ECC integra al conjunto todos los puntos (x,y) que se encuentran sobre la curva y además con esos elementos realiza las operaciones fundamentales (suma, resta, multiplicación y división).

Una ventaja adicional de usar los grupos de puntos sobre la curva elíptica, consiste en que pueden representarse sobre campos característico dos (los elementos del grupo pueden ser ceros y unos), permitiendo que las implementaciones sean más compactas y las operaciones hacia adelante se hagan con menor dificultad, pero incrementándola notablemente en la operación inversa del algoritmo. [6]

La teoría de curvas elípticas es un tópico matemático estudiado y conocido desde el siglo XVIII, siendo hasta no hace mucho tiempo un campo puramente teórico sin alguna utilización práctica. Sin embargo, en los últimos 30 años, la criptografía tuvo una expansión muy importante que impulsó la investigación de más teorías matemáticas que le den soporte. Entre estas teorías se encuentra la relacionada con curvas elípticas definidas en un campo finito.

Las primeras propuestas de uso de las curvas elípticas fueron hechas en forma independiente por Neal Koblitz [8] y Víctor Miller [9] en 1986. Así fue posible definir esquemas como el de intercambio de claves Diffie-Hellman y el de firma digital utilizando esta teoría.

En la actualidad la tecnología que usa curvas elípticas en la criptografía es una de las más sobresalientes. Su uso en el área comercial se ha incrementado, sin dejar de mencionar que las más importantes y grandes agencias de seguridad, la han tomado como la mejor opción.

Las bases matemáticas necesarias para entender la teoría de curvas elípticas parten del Álgebra Abstracta, comenzando por teoría de números, pasando

por funciones, polinomios, grupos, etc, hasta campos, siguiendo con Geometría Analítica y Álgebra Lineal.

Es pertinente estudiar estas temáticas para comprender el desarrollo matemático de las curvas elípticas. Las teorías referidas a la aritmética modular y en forma particular a las curvas elípticas se encuentran en los anexos A y B, para una mayor comprensión y organización.

4.1 Problema del logaritmo discreto de curvas elípticas (PLDCE) [6]

Sea G un grupo cíclico finito de orden n. Sea α un generador de G y sea $\beta \in$ G. El logaritmo discreto de β en la base α, denotado $\log_\alpha \beta$es el único entero x, $0 \leq x \leq n-1$, tal que $\beta = \alpha^x$.

El problema del logaritmo discreto (PLD) es, dado un grupo cíclico finito G de orden n, un generador $\alpha \in G$, encontrar el entero x, $0 \leq x \leq n-1$, tal que $\alpha^x = \beta$.

Este problema puede también definirse para el grupo de puntos de una curva elíptica sobre un campo finito. Sea E una curva elíptica definida sobre un campo finito $\mathbb{F}_q$ y sea $P \in E(\mathbb{F}_q)$ y $Q \in E(\mathbb{F}_q)$ dos puntos sobre E de orden n, el Problema del Logaritmo Discreto de curvas elípticas (PLDCE) consiste en encontrar un entero k, $0 \leq k \leq n-1$, suponiendo que existe, tal que $Q = kP$.

4.1.1 Longitud de la clave [10]

En los cuerpos finitos primos, los elementos del cuerpo $\mathbb{F}_p$, se representan mediante los números enteros $\{0,1,2,\dots,p-1\}$. En este caso, el elemento neutro de la operación suma es el entero 0, el elemento unitario de la operación producto es el entero 1, la suma de elementos se realiza a partir de la suma de enteros módulo p y la multiplicación de elementos del cuerpo se efectúa igualmente mediante la multiplicación de números enteros módulo p.

Por otro lado se tiene a los cuerpos finitos de tipo $\mathbb{F}_{2^m}$, donde los elementos del cuerpo se representan mediante cadena de bits de longitud m. Si el polinomio $f(x)$ es irreducible de grado m con coeficientes en $\mathbb{F}_2$, entonces el

cuerpo $\mathbb{F}_{2^m}$ puede interpretarse como el conjunto de polinomios con coeficientes en $\mathbb{F}_2$ de grado estrictamente menor que el grado $f(x)$, es decir se cumple:

$$\mathbb{F}_{2^m} = \mathbb{F}_2[X]/(f(x))$$

Los parámetros que definen el tamaño del cuerpo finito son fundamentales para poder calcular la longitud de las curvas elípticas y de las claves asociadas a dichas curvas. Esta longitud tiene que ser entendida como el número de bits necesarios para representar el entero p en curvas sobre cuerpos primos (es decir, el valor $\lceil log_2 p \rceil$, mientras que en el caso de curvas sobre cuerpos binarios la longitud coincide con el elemento m.

4.2 Elección del campo finito

Como se vio anteriormente, las dos elecciones que existen en aplicaciones prácticas del campo finito subyacente son $\mathbb{F}_{2^m}$ y $\mathbb{F}_p$, siendo p, un primo diferente de 2 y 3. El problema del logaritmo discreto es igualmente difícil en curvas sobre $\mathbb{F}_{2^m}$, como en curvas definidas sobre $\mathbb{F}_p$, suponiendo que 2^m y p son aproximadamente iguales. Hasta la fecha no hubo ningún descubrimiento matemático que permita suponer que el PLDCE sobre $\mathbb{F}_{2^m}$ tenga mayor facilidad de resolución sobre $\mathbb{F}_p$ o viceversa. [11]

Debido a que los campos finitos de característica 2 están formados por 2^m elementos para algún entero positivo m, poseen la ventaja de poder representarse como m-tuplas, cadenas de m elementos de 0 y 1. Esto favorece las implementaciones de hardware, ya que las computadoras trabajan en forma natural con este tipo de elementos.

Los campos finitos de característica impar a veces proporcionan resultados más óptimos en implementaciones de software [12] [11]. Para el presente trabajo se decidió la utilización de campos finitos sobre$\mathbb{F}_p$ en la implementación en software, a pesar de que en la mayoría de los fundamentos teóricos se incluyan tanto campos sobre $\mathbb{F}_p$como sobre $\mathbb{F}_{2^m}$.

4.3 Estándares relacionados

La información precisa que permite realizar implementaciones prácticas se detalla en los estándares desarrollados por diferentes organizaciones, tanto

internacionales como consorcios industriales. A continuación se nombran los principales estándares y las organizaciones que elaboran algún tipo de norma relacionada con curvas elípticas y su aplicación en la criptografía.

ANSI X9.62 y X9.63: Estos dos estándares son referidos exclusivamente a curvas elípticas, mientras que X9.62 trata sobre ECDSA, el X9.63 lo hace sobre ECDH, ECMQV y ECIES. Ambas especificaciones indican un formato de mensajes a ser usados y una lista de curvas recomendadas.

FIPS 186.2: este estándar sobre firmas digitales es una actualización del FIPS 186, que detalla solamente algoritmos de DSA. En cambio el FIPS 186.2 especifica tanto DSA como ECDSA y brinda una lista de curvas recomendadas, las cuales son mandatarias para instalaciones en el gobierno de Estados Unidos.

SEC 1 y 2: Fue escrito por un consorcio internacional de empresas lideradas por Consortium. SEC 1, se encarga de describir en forma detallada los fundamentos matemáticos de las curvas elípticas junto con su aplicación en diversos algoritmos. Por otro lado, SEC 2 es un complemento que actualiza a SEC 1 y enuncia parámetros de dominio de curvas elípticas para alcanzar distintos niveles de seguridad.

4.4 Dominio de parámetros [6]

La definición del dominio de parámetros consiste en una elección adecuada elección de la curva elíptica E, definida sobre un campo finito $\mathbb{F}_p$ de característica p, y un punto de base P que pertenezca a $E(\mathbb{F}_p)$. Cabe destacarse que estos parámetros deben ser compartidos por las partes que quieren comunicarse, de modo que en general se trata de usar siempre los mismos parámetros que los recomendados por organizaciones productoras de estándares.

Los parámetros de dominio de curva elíptica sobre campos $\mathbb{F}_p$ pueden ser de dos tipos: uno asociado con las curvas de Koblitz y otros son aquellos elegidos en forma pseudoaleatoria. Se puede decir que las primeras son elegidas en algunos casos por admitir implementaciones eficientes. Sin embargo para este trabajo sólo se considerarán el segundo tipo.

Los parámetros elegidos en forma pseudoaleatoria tienen algunas ventajas adicionales. Son elegidos a partir de una semilla s usando el método de hash SHA-1 tal como se especifica en el estándar ANSI X9.62. Este proceso asegura que los parámetros no sean predeterminados y que no sean susceptibles a futuros ataques de propósito general.

Cuando los parámetros se eligen de este modo, la semilla s utilizada como entrada del Hash puede ser almacenada junto con el resto de los parámetros, de modo que los usuarios puedan verificar que los mismos fueron efectivamente elegidos al azar.

A continuación se definen cuáles son estos parámetros y un algoritmo para generarlos, solo a modo informativo, pues se utilizarán las curvas propuestas en el estándar SEC 2. [13].

4.5 Requerimientos de la curva elíptica [14]

Los parámetros que definen el dominio de las curvas elípticas sobre $\mathbb{F}_p$ son una sextupla:

$$T = (p, a, b, G, n, h)$$

Donde cada uno de los elementos representa a:

- Un número entero primo p que especifica el campo finito $\mathbb{F}_p$
- Dos elementos $a, b \epsilon \mathbb{F}_p$ especificando una curva elíptica $E(\mathbb{F}_p)$definida por la ecuación:

$$E: y^2 \equiv x^3 + ax + b \ (mod\ p)$$

Ecuación 1

- Un punto base $G = (x_G, y_G) \in E(\mathbb{F}_p)$.
- Un número primo n el cual es el orden de G, el cual debe cumplir que $n \geq 2^{160}$ y $n \geq 4\sqrt{q}$.
- Un número entero h que es el cofactor $h = \#E(\mathbb{F}_p)/n$

El punto G es un punto de referencia que se proporciona para poder hacer criptografía. Es el elemento análogo al generador en el DLP. El número n es el orden de G, es decir, $\mathrm{nG} = \mathcal{O}$.

Se debe tener en cuenta que n es un número primo grande. El orden de G debe ser lo suficientemente grande como para que sea impráctico buscar todos los múltiplos $G, 2G, \dots, (n-1)G$ de G.

Si los parámetros del dominio de la curva elíptica T son verificablemente aleatorios (no es tema de estudio del presente proyecto), debe incluirse el valor de la semilla s de los cuales son derivados.

Algoritmo para la generación de parámetros de dominio de curva elíptica [14]

El proceso para generar la séxtupla $T = (p, a, b, G, n, h)$, es el siguiente:

I. Elegir el nivel de seguridad aproximado para los parámetros de dominio de curva elíptica. Este debe ser un entero $t \in \{80, 112, 128, 192, 256\}$, opcionalmente el valor de la semilla s. La Tabla 2 muestra información adicional para ayudar a determinar el nivel de seguridad deseado.

Nivel de Seguridad	Alg. Simétrico	ECC	DSA/RSA	Protegido hasta año
112	112	224	1024	2030
128	128	256	2048	2040
192	192	384	3072	2080
256	256	512	15360	2120

Tabla 2: Nivel de Seguridad de ECC

Una vez elegido esto, generar los parámetros de dominio de la curva elíptica como sigue:

II. Seleccionar un número primo p, tal que $\lceil \log_2 p \rceil = 2t$ si $80 < t < 256$, tal que $\lceil \log_2 p \rceil = 521$ si $t = 256$, tal que $\lceil \log_2 p \rceil = 192$ si $t = 80$, para determinar el campo finito $\mathbb{F}_p$.

III. Seleccionar los elementos $a, b \in \mathbb{F}_p$ para determinar la curva $E(\mathbb{F}_p)$, definida por la Ecuación 1, un punto base $G = (x_G, y_G)$ sobre $E(\mathbb{F}_p)$, un número primo n el cual es el orden de G, y un cofactor entero $h = \#E(\mathbb{F}_p)/n$, que tenga las siguientes restricciones:

a. $4a^3 + 27b^2 \not\equiv 0 \; mod(p)$.

b. $\#E(\mathbb{F}_p) \neq p$

c. $p^{\beta} \not\equiv 1\ (mod\ n)$ para $1 \leq \beta \leq 100$.

d. $h \leq 2^{t/8}$

e. $n-1$ y $n+1$ deben tener un factor primo grande r, de modo que $log_n(r) > {}^{19}/_{20}$

IV. Si es provista la semilla s, el par de coeficientes (a, b) o el punto $G = (x_G, y_G)$ o ambos serán derivados de la misma.

V. Retornar (p, a, b, G, n, h)

Observaciones:

Dependiendo del nivel de seguridad que se busca, el número primo p de $\mathbb{Z}_p$ se debe elegir de manera que:

$$\lceil \log_2 p \rceil \in \{112, 128, 160, 192, 224, 256, 384, 521\}$$

Esto representa el tamaño en bits de p. Se debe tener en cuenta la aparición de 521 en lugar de 512, esto se debe a un requerimiento de seguridad del gobierno de los Estados Unidos.

Respecto de las restricciones que deben cumplir los demás parámetros:

a. Se requiere para que cumpla con la definición de curvas elípticas.

b. Se pide para evitar curvas anómalas.

c. Para evitar curvas supersingulares (condición MOV).

d. Se pide que $h \leq 4$, para que $\#E(\mathbb{F}_p)$ sea un número cercano a un primo grande.

4.6 Ataques conocidos contra las curvas elípticas [15] [16]

A continuación se presentan una serie de ataques conocidos sobre el PLDCE y en algunos casos la forma de definir las curvas elípticas para evitarlos.

Ataque por fuerza bruta: consiste en la búsqueda exhaustiva de k a partir de P hasta encontrar $Q = kP$. La complejidad es exponencial, con un número de iteraciones esperado de $n/2$.

El algoritmo de Pohlig-Hellman: su funcionamiento se basa en la factorización de n, el orden del punto P, entonces la búsqueda de k se reduce a

hallar k módulo los factores primos de n. Luego se recupera k usando el teorema chino del resto.

Una forma de evitarlo es elegir el orden de la curva un número primo o casi primo (contener al menos un factor primo grande).

Algoritmo BSGS (Baby Step, Giant Step), Algoritmo de paso de bebé-paso de gigante: consiste en calcular puntos partiendo de P, en un primer momento dando saltos pequeños (diferencia de un punto) y luego saltos grandes (diferencia $\sqrt{n}$. Al obtener coincidencia se consigue k.

Tiene una complejidad exponencial, $O(\sqrt{n})$, tanto en tiempo como en espacio.

Algoritmo de Pollard-ρ: basado en el BSGS pero los saltos los hace en forma aleatoria. Tiene una complejidad temporal similar al BSGS $O(\sqrt{\pi n/2})$, aunque casi sin requerimientos de almacenamiento. Existen mejoras reduciendo la complejidad con el uso de procesadores en paralelo.

Algoritmo de Pollard-λ: tiene un funcionamiento similar al anterior, puede ser paralelizado eficientemente aunque más lento.

Múltiples instancias: el algoritmo de Pollard-ρ se encarga de almacenar algunos puntos determinados al hacer la búsqueda de k. Una vez resuelta la instancia del problema, se puede utilizar la información almacenada por el algoritmo para acelerar el cálculo de otra instancia del mismo problema, esto se hace sin considerar el espacio de almacenamiento requerido.

Ataque MOV (Menezes-Okamoto-Vanstone): En algunos casos, el PLDCE de una curva $E(\mathbb{F}_q)$ puede reducirse el PLD en el grupo multiplicativo del campo de extensión $\mathbb{F}_{q^r}$ para algún $r \geq 1$, problema más simple de resolver si r no es grande. Para evitar que este algoritmo sea aplicable, se debe comprobar que n no divida a $q^r - 1$ para todo r pequeño para el que se pueda solucionar el PLD bajo $\mathbb{F}_{q^r}$. Para valores comunes de $n \approx 2^{160}$, es suficiente verificar los valores de r, con $r \leq 20$.

Para la mayoría de las curvas, el r resultante será grande, entonces no podrá aplicarse esta reducción. Existen las curvas supersingulares que cumplen con $r \leq 6$, con lo que se podría reducir el cálculo a tiempo subexponencial. Por ello estas curvas son excluidas de los estándares.

Curvas anómalas de cuerpo primo: Una curva $E(\mathbb{F}_p)$ es anómala de cuerpo primo si $\#E(\mathbb{F}_p) = p$. El PLDCE en estas curvas puede ser resuelto en forma eficiente. Por ello se debe evitar que el cardinal del grupo de puntos racionales coincida con el cardinal del cuerpo sobre el que se defina la curva. Estas curvas también se excluyeron de los estándares.

Algoritmo Index-calculus: Es el mejor algoritmo que se conoce capaz de atacar al DLP, sin embargo aún no se pudo aplicar al ECDLP. Existen algunos trabajos de investigación acerca de los mismos [17].

4.6.1 Representación de los puntos de una curva elíptica [10]

Dada una curva elíptica E definida sobre un cuerpo finito $\mathbb{F}_q$ de característica p, un punto P de la curva puede representarse de forma comprimida o descomprimida, como se muestra a continuación: (se tendrá en cuenta solo para campos primos).

- Forma descomprimida

La forma descomprimida de un punto $P = (x, y)$ es la cadena $0x04||X||Y$, donde X e Y son las representaciones hexadecimales de la primera y segunda coordenadas del punto, teniendo ambas cadenas hexadecimales una longitud de $\lceil (log_2 q)/8 \rceil$ bytes.

- Forma comprimida

La forma comprimida de un punto $P = (x, y)$ es la cadena $C||X$, donde X es lar representación hexadecimal de la primera coordenada del punto, y el elemento C puede tener los valores $0x02$ o $0x03$ como lo muestra el algoritmo, dependiéndose del punto.

Compresión de un punto $P = (x, y)$ de la curva

I. Obtener la representación binaria hexadecimal X de la coordenada x que deberá tener una longitud de $\lceil (log_2 q)/8 \rceil$ bytes

   Calcular $\tilde{y} = y \ (mod\ 2)$.

II. Si $\tilde{y} = 0$, asignar al byte C el valor $0x02$. En caso contrario asignarle el valor $0x03$.

III. Generar como la representación comprimida de P la cadena $C||X$.

Descompresión de un punto $P = (x, y)$ de la curva:

I. Interpretar la cadena hexadecimal de longitud $\lceil(log_2 q)/8\rceil$+1 bytes, como los elementos concatenados $C||X$, donde la longitud de C es de un byte.

II. Convertir la cadena X en el elemento x del cuerpo $\mathbb{F}_q$.

III. Si el valor del byte C es $0x02$, realizar la asignación $\tilde{y} = 0$. Si por el contrario C es $0x03$ se debe hacer $\tilde{y} = 1$.

IV. Derivar el valor de y a partir de los elementos x e $\tilde{y}$ de la siguiente manera: calcular el elemento del cuerpo $\mathbb{F}_q$ $\alpha \equiv x^3 + ax + b\ (modp)$, y calcular una raíz cuadrada β de α módulo p. Si $\beta \equiv \tilde{y}\ (\mathrm{mod}\ 2)$, hacer $y = \beta$. Si no se cumple la equivalencia indicada tomar $y = p - \beta$.

V. Obtener como salida del procesos el punto $P = (x, y)$

5 ESTABLECIMIENTO DE CLAVES COMPARTIDAS

El establecimiento de una clave compartida es el proceso mediante el cual dos o más entidades comparten de manera secreta un código. Esta clave posteriormente será utilizada para alcanzar algún objetivo criptográfico, tal como la integridad o confidencialidad de los datos.

5.1 Intercambio de claves Diffie-Hellman [10]

En 1976, Whitfield Diffie y Martin Hellman [18], presentaron una manera de resolver las necesidades de seguridad de aquel momento a través del concepto de criptografía de clave pública. La propuesta se conoció como Protocolo de establecimiento o acuerdo de clave Diffie-Hellman, consiste en un mecanismo por el que dos entidades intercambian pequeñas cantidades de información a través de un canal inseguro, de modo que al terminar el proceso únicamente ellos conozcan la clave secreta compartida.

El protocolo de acuerdo de clave Diffie-Hellman originalmente no fue planteado como un criptosistema, ya que no realiza el cifrado y descifrado de información, sino que sólo permite el intercambio de información con el objetivo de conseguir una clave secreta.

A continuación se presenta de manera ordenada los pasos que los usuarios participantes del intercambio deben dar para poder obtener una clave secreta común con la que pueden cifrar toda la comunicación posterior.

Después de seguir los pasos, U y V conocen y comparten un elemento común del grupo $\mathbb{Z}_p^*$ y que es desconocido para cualquier otro usuario. Este elemento es:

$$(\alpha^v)^u = (\alpha^u)^v = \alpha^{uv} \in \mathbb{Z}_p^*$$

5.2 Algoritmo de establecimiento de clave Diffie-Hellman

I. Si U y V son los dos usuarios que participan en el protocolo, en primer lugar deben seleccionar y hacer público un número primo p y un generador $\alpha \in \mathbb{Z}_p^*$ donde $\mathbb{Z}_p^* = \{\alpha^i : i = 1, \dots, p-1\}$ y $1 < \alpha < p - 1$.

II. A continuación U debe generar un número entero aleatorio u, con $1 < u < p - 1$, calcular $\alpha^u \in \mathbb{Z}_p^*$, y transmitir este elemento a V.

III. De forma análoga, V tiene que generar un número aleatorio v, $1 < v < p - 1$, calcular $\alpha^v \in \mathbb{Z}_p^*$, y transmitir este elemento a U.

IV. Tras recibir α^v, U debe calcular el valor de $(\alpha^v)^u \in \mathbb{Z}_p^*$.

V. De manera equivalente, una vez recibido el valor α^u, V debe proceder a calcular el valor de $(\alpha^u)^v \in \mathbb{Z}_p^*$.

5.3 Protocolo Diffie-Hellman de Curva Elíptica [19]

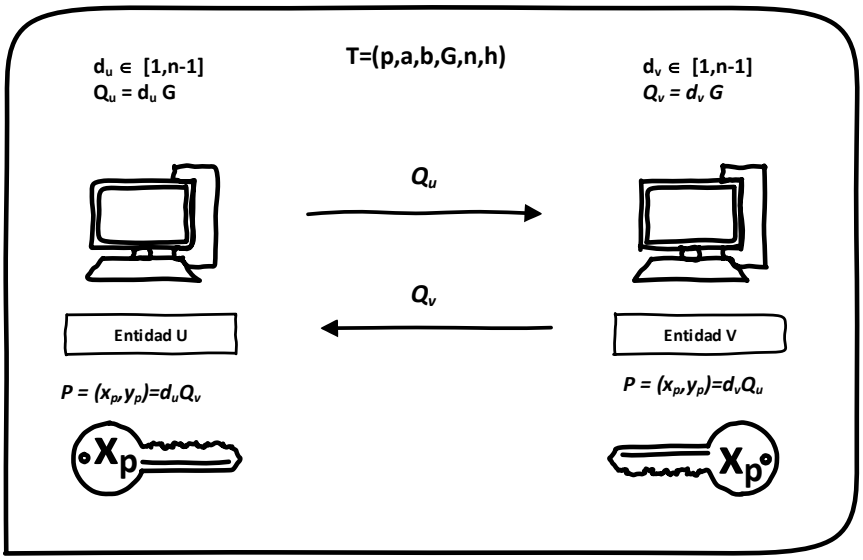

Figura 2: Diffie Hellman de Curvas Elípticas

Para que una entidad U pueda compartir un valor secreto con la entidad V usando el protocolo Diffie-Hellman de curva elíptica, se debe ejecutar el proceso siguiente como se observa en la Figura 2, que toma como entrada los parámetros validados de curva elíptica $T = (p, a, b, G, n, h)$, la clave privada d_U asociada con T perteneciente a U y una clave pública Q_V (al menos parcialmente validada) asociada con T perteneciente a V.

I. Calcular el punto $P = (x_p, y_p) = d_u Q_V$ de la curva elíptica

II. Chequear que $P \neq \mathcal{O}$. Si $P = \mathcal{O}$, retornar “inválido” y parar

III. Retornar $z = x_p$, el elemento del campo finito que es secreto y compartido.

Donde se tiene que:

- Para la entidad U:

La clave privada d_u es un entero elegido en forma aleatoria o pseudoaleatoria en el intervalo $[1, n-1]$.

La clave pública Q_u es el punto $Q_u = d_u G$

- Para la entidad V:

La clave privada d_v es un entero elegido en forma aleatoria o pseudoaleatoria en el intervalo $[1, n-1]$.

La clave pública Q_v es el punto $Q_v = d_v G$

5.3.1 Validación de claves públicas [19]

Para comprobar la validez de una clave pública que una entidad quiere utilizar, es necesario ejecutar una serie de pruebas sobre la misma (usando los parámetros de dominio) para verificar que la clave efectivamente fue derivada en forma correcta.

Generalmente sobre Internet, la clave pública estará embebida dentro de un certificado PKI (o X.509) emitido por un CA confiable, autenticado en forma apropiada, por lo que la entidad a través de una verificación de firma comprueba que la clave en el certificado sea auténtica.

En algunos casos, para una entidad es suficiente que una clave pública sea parcialmente validada. Una clave pública de curva elíptica Q se dice que es parcialmente válida si Q es un punto sobre la curva elíptica asociada pero no necesariamente es cierto que $Q = dG$ para algún entero d.

6 DESARROLLO DEL SISTEMA

6.1 Introducción

A partir de los conceptos teóricos desarrollados, se está en condiciones de construir un sistema basado en el One-Time Pad, que tenga la propiedad de tener un cifrado indistinguible ante la presencia de un espía. La gran diferencia con el protocolo original, es que se usará como "pad" una cadena pseudoaleatoria obtenida de un intercambio ECDH, en lugar de una cadena totalmente aleatoria. Esto se ilustra en la Figura 3 .

Figura 3: Esquema propuesto

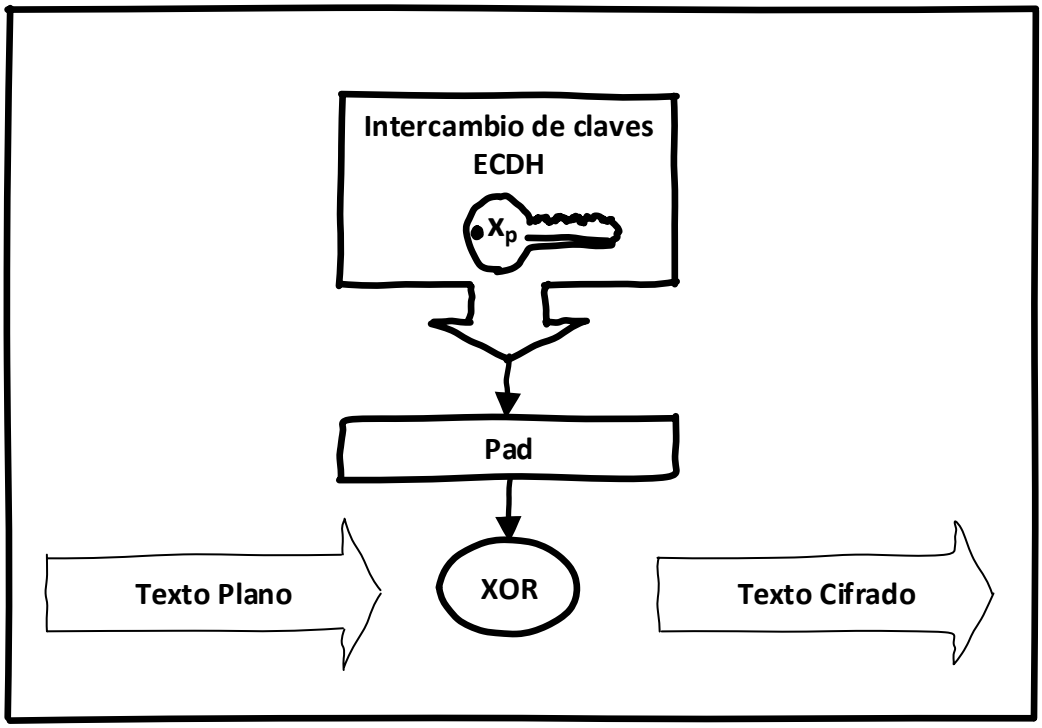

Figura 3: Esquema propuesto

6.2 One-Time Pad modificado

Sea $a \oplus b$ la operación bit a bit or-exclusiva (XOR) de dos cadenas binarias a y b, entonces $a \oplus b = a_1 \oplus b_l, \dots, a_\ell \oplus b_\ell$. El esquema de cifrado es definido como:

- Dado un entero $\ell > 0$. Luego el espacio de mensaje M, el espacio de claves K y el espacio de textos cifrados C, son todos iguales a $\{0,1\}^{\ell}$. ℓ está delimitado por el tamaño de clave compartida que puede generarse con el protocolo ECDH.
- El algoritmo de generación de claves Gen elige las llaves pseudoleatorias utilizando el protocolo ECDH.
- El algoritmo Enc opera de la siguiente manera: dada una clave $k \in \{0,1\}^{\ell}$ y un mensaje $m \in \{0,1\}^{\ell}$, el valor de la salida es $c \coloneqq k \oplus m$.
- El proceso Dec trabaja como se sigue: dada una clave $k \in \{0,1\}^{\ell}$ y un texto cifrado $c \in \{0,1\}^{\ell}$, el valor de la salida es $m \coloneqq k \oplus c$.

El One-Time Pad modificado constituye un esquema de cifrado ya que $\forall\, k, m$ se tiene $Dec_k\big(Enc_k(m)\big) = k \oplus k \oplus m$.

6.2.1 Funcionamiento del sistema

La comunicación entre las partes se produce de manera sincrónica. Una vez que los usuarios se encuentran conectados entre sí (no se contempla la autenticación dentro del alcance del trabajo), se produce un intercambio de claves Diffie Hellman basado en curvas elípticas.

Luego de este intercambio, los usuarios cuentan con una clave compartida cuya longitud depende de la curva elegida, y que a su vez es recortada para que sea múltiplo del tamaño que ocupa almacenar un valor char. Se verá que para el caso de Java es de 2 bytes (16 bits).

El tamaño máximo del mensaje es igual al tamaño de la clave definida anteriormente. En caso de que el mensaje tenga un tamaño menor, se elige un carácter denominado de relleno (por ejemplo #) que completa los caracteres hasta alcanzar el máximo permitido. Este carácter no debe usarse por ningún usuario.

Una vez que se cuenta con la clave compartida existen dos roles de los usuarios, uno de ellos tiene la posibilidad de enviar el mensaje y el otro está a la espera de recibir el mensaje. Los usuarios se nombrarán como A o B.

Se elige en forma arbitraria el usuario que tiene derecho a enviar el primer mensaje.

El usuario A, una vez que cuenta con el mensaje utiliza el One-Time Pad modificado para cifrar el mensaje., donde el Xor se establece a nivel de bits, es decir depende de la codificación de cada carácter (ASCII, UTF-8, UTF-16). Se procede al envío del mensaje.

El usuario B recibe el mensaje y lo decodifica utilizando One-Time Pad modificado (Xor a nivel de bits).

Se produce nuevamente un intercambio de claves Diffie Hellman y se genera una nueva clave compartida. Se invierten los roles y es ahora el usuario B el que tiene derecho a enviar y el usuario A el que recibe.

Este proceso de generación e intercambio de claves, envío y recepción de mensajes se repite hasta que uno de los dos usuarios envía una frase que cierra la comunicación entre ambos.

6.3 Análisis de indistinguibilidad del sistema

A continuación se describirá una construcción criptográfica muy similar al OTP modificado, en la que en la bibliografía [2] se realiza una demostración de su carácter de indistinguible frente a la presencia de un espía, lo que permite decir que es seguro semánticamente.

6.3.1 Construcción criptográfica con un generador pseudoaleatorio

Sea G un generador pseudoaleatorio de factor de expansión ℓ. Se define un esquema de cifrado de clave privada para mensajes de tamaño ℓ como sigue:

- Gen: entrada 1^n, se eligen se elige $k \leftarrow \{0,1\}^n$ uniformemente en forma aleatoria y la salida es la clave.
- Enc: entrada la clave $k \in \{0,1\}^n$ y un mensaje $m \in \{0,1\}^{\ell\{n\}}$, y produce como salida el texto cifrado:

$$c := G(k) \oplus m.$$

- Dec: entrada la clave $k \in \{0,1\}^n$ y el texto cifrado $c \in \{0,1\}^{\ell(n)}$, y produce como salida el mensaje en texto plano

$$m := G(x) \oplus c$$

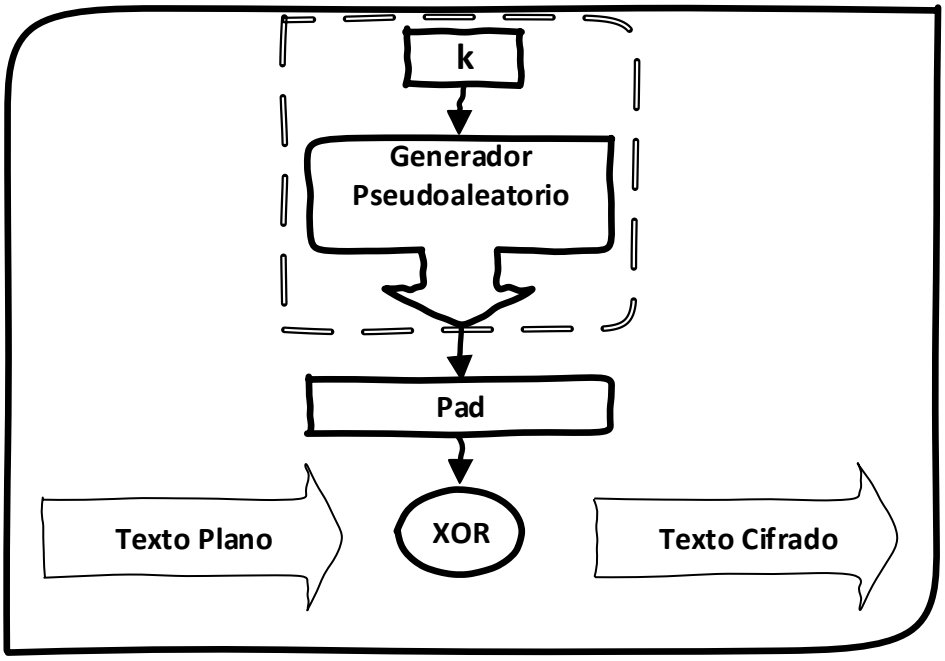

Figura 4: Construcción con generador pseudoaleatorio

6.3.2 Teorema

Si G es un generador pseudoaleatorio, entonces la construcción del punto 6.3.1 es un esquema de cifrado del clave de tamaño fijo que tiene la propiedad de tener seguridad indistinguible frente a la presencia de un espía.

6.3.2.1 Demostración

Sea Π, la construcción 6.3.1. Se probará que si existe un adversario $\mathcal{A}$ de tiempo polinomial probabilístico que no cumple con la característica de seguridad indistinguible, se puede construir un algoritmo de tiempo polinomial probabilístico que distinga la salida de G de una cadena verdaderamente aleatoria. Si Π usa una cadena verdaderamente aleatoria en lugar de una cadena pseudoaleatoria $G(k)$, entonces el esquema resultante sería idéntico al esquema OTP y $\mathcal{A}$ sería capaz de adivinar correctamente cual mensaje fue encriptado con una probabilidad mejor que ½.

Sea $\mathcal{A}$ un adversario de tiempo polinomial probabilístico, y se define ε como:

$$\varepsilon(n) \stackrel{\text{def}}{=} \Pr\left[PrivK_{\mathcal{A},\Pi}^{esp}(n) = 1\right] - \frac{1}{2}$$

Ecuación 2

Se usa $\mathcal{A}$ para construir un observador $\mathcal{D}$ para el generador pseudoaleatorio G, de modo que $\mathcal{D}$ tiene éxito con una probabilidad $\mathcal{E}(n)$. El observador $\mathcal{D}$ tiene como entrada la cadena w y su objetivo es determinar si w fue elegido en forma aleatoria o si fue generado eligiendo un valor aleatorio k y computando $w := G(k)$. $\mathcal{D}$ emula el experimento del espía para $\mathcal{A}$ como se muestra a continuación, y observa si $\mathcal{A}$ tiene éxito o no. Si $\mathcal{A}$ tiene éxito entonces $\mathcal{D}$ adivina que w debe ser una cadena pseudoleatoria, mientras que si $\mathcal{A}$ no tiene éxito, entonces $\mathcal{D}$ adivina que w es una cadena aleatoria. Se tiene:

Observador $\mathcal{D}$:

$\mathcal{D}$ tiene como entrada una cadena $w \in \{0,1\}^{\ell(n)}$.

I. Ejecutar $\mathcal{A}\ (1^n)$ para obtener un par de mensajes $m_0, m_1 \in \{0,1\}^{\ell(n)}$.
II. Elegir un bit aleatorio $b \leftarrow \{0,1\}$. Asignar $c \coloneqq w \oplus m_0$.
III. Dar c a $\mathcal{A}$ y obtener la salida b'. Dar como salida 1 si $b' = b$, y dar salida 0 de otro modo.

Se define el esquema de cifrado $\widetilde{\Pi} = (\widetilde{Gen}, \widetilde{Enc}, \widetilde{Dec})$ que es exactamente igual al esquema OTP, salvo que se incorpora el parámetro de seguridad que determina el tamaño de los mensajes a cifrar. Se tiene que $\widetilde{Gen}(1^n)$ genera como salida una clave completamente aleatoria k de tamaño $\ell(n)$, y el cifrado del mensaje $m \in \ell(n)$ que usa la clave $k \in \{0,1\}^{\ell(n)}$ es $c \coloneqq w \oplus m$. Por la propiedad de secreto perfecto del OTP:

$$\Pr\left[PrivK_{A,\Pi}^{esp}(n) = 1\right] = \frac{1}{2}$$

Ecuación 3

Se puede detallar algunas observaciones sobre el comportamiento de $\mathcal{D}$:

I. Si w es elegido en forma uniformemente aleatoria de $\{0,1\}^{\ell(n)}$, entonces el punto de vista de A cuando se ejecuta como subrutina por $\mathcal{D}$ es idéntico al punto de vista de A en el experimento $PrivK_{A,\widetilde{\Pi}}^{esp}(n)$. Esto es porque A recibe un texto cifrado $c \coloneqq w \oplus m_b$, donde $w \in \{0,1\}^{\ell(n)}$ es completamente aleatorio, se tiene:

$$\Pr[\mathcal{D}(w) = 1] = \Pr\,[PrivK_{A,\Pi}^{esp}(n) = 1] = \frac{1}{2}$$ (Por la Ecuación 3)

II. Si w es igual a $G(k)$ para $k \leftarrow \{0,1\}^n$ elegido uniformemente en forma aleatoria, entonces el punto de vista de A cuando se ejecuta como una subrutina por $\mathcal{D}$ es idéntico al punto de vista de A en el experimento $PrivK_{A,\Pi}^{esp}(n)$. Esto es porque A recibe un texto cifrado $c := w \oplus m_b$, donde $w = G(k)$ para un valor con una distribución uniforme $k \longleftarrow \{0,1\}^n$. Entonces se tiene:

$$\Pr[\mathcal{D}(w) = 1] = \Pr[\mathcal{D}(G(k) = 1] = \Pr\left[PrivK_{A,\Pi}^{esp}(n) = 1\right] = \frac{1}{2} + \mathcal{E}(n)$$

Donde el segundo término viene de la definición de $\mathcal{E}$

Entonces

$$|\Pr[D(w) = 1] - Pr[D(G(k)) = 1]| = \mathcal{E}(n)$$

Donde w es elegido uniformemente de $\{0,1\}^{\ell(n)}$ y k es elegido en forma uniforme de $\{0,1\}^n$. Por la suposición de que G es un generador pseudoaleatorio, se debe cumplir que $\mathcal{E}$ es despreciable. Por la definición de $\mathcal{E}$ (ver Ecuación 2) implica que Π tiene un cifrado indistinguible frente a la presencia de un espía.

6.3.3 Generación de claves pseudoaleatorias en ECDH

Es importante tener en cuenta la pseudoaleatoriedad de las claves compartidas generadas por el intercambio ECDH. Esta característica está garantizada si dadas dos entidades U y V, un dominio de parámetros conocido $T = (p, a, b, G, n, h)$ y las claves privadas d_u y d_v son enteros elegidos en forma pseudoleatoria en el intervalo $[1, n-1]$. De modo que la clave compartida x_p es también un entero pseudoaletorio, donde $\mathrm{P} = (\mathrm{x}_p, \mathrm{y}_p) = d_u Q_{v=} d_v Q_u = d_u d_v G$. [14].

Una deficiente generación de números pseudoaleatorios es quizás la causa más común de ataques contra los sistemas criptográficos. [14]

6.3.4 Análisis de indistinguibilidad del OTP modificado

Se tomará como conjetura que el generador pseudoaleatorio de la Figura 4, es una caja negra, que tiene como salida justamente una cadena pseudoaleatoria,

por lo tanto puede ser reemplazada por otro sistema que tenga la misa salida, en este caso el OTP modificado. Como se describió en el apartado anterior, el mismo tiene como salida una clave de sesión compartida de longitud fija que posee la característica de ser una cadena pseudoaleatoria.

Se debe tener en cuenta que un generador pseudoaleatorio genera su secuencia de salida a partir de una semilla s elegida e forma aleatoria o pseudoaleatoria, mientras que a través del intercambio ECDH se generan una clave de sesión pseudoaleatoria a través de aplicar operaciones matemáticas sobre dos claves privadas elegidas en forma aleatoria.

Teniendo en cuenta la fortaleza de las curvas elípticas frente a los ataques contra el logaritmo discreto, y haciendo las salvedades citadas en los párrafos anteriores, se puede afirmar que el OTP modificado, presenta la característica de seguridad indistinguible, por lo tanto es seguro semánticamente.

7 EL LENGUAJE JAVA

Sun Microsystems desarrolló, en 1991, el lenguaje de programación orientado a objetos que se conoce como Java. El objetivo era utilizarlo en pequeños dispositivos electrónicos. El primer nombre del lenguaje fue Oak, luego se conoció como Green y finamente adoptó su denominación actual de Java.

La intención de la empresa Sun era crear un lenguaje similar a C y C++, tanto en su estructura y sintaxis, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel.

Lo habitual es que las aplicaciones Java se encuentren compiladas en un bytecode (un fichero binario que tiene un programa ejecutable), aunque también pueden estar compiladas en código máquina nativo. Sun controla las especificaciones y el desarrollo del lenguaje, los compiladores, las máquinas virtuales y las bibliotecas de clases a través del Java Community Process[7]. En los últimos años, la empresa (que fue adquirida por Oracle) ha liberado gran parte de las tecnologías Java bajo la licencia GNU GPL.

7.1 Justificación de elección del Lenguaje Java

Razones para escoger Java por sobre otros lenguajes [20] [21]:

- **Es orientado a objetos:** un objeto es una instanciación o concreción de una clase. Una clase es el molde o prototipo que especifica variables y métodos comunes de un determinado tipo de elemento. Entre las ventajas más evidentes que ofrece se encuentra un gran control sobre el código y una mejor organización, dado que basta con escribir una vez los métodos y las propiedades de un objeto, independientemente de la cantidad de veces que se utilicen.
- **Es muy flexible:** Java es un lenguaje especialmente preparado para la reutilización del código; permite a sus usuarios tomar un programa que hayan desarrollado tiempo atrás y actualizarlo con mucha facilidad, sea que necesiten agregar funciones o adaptarlo a un nuevo entorno. Esto se

[7] Java Community Proccess: proceso de la comunidad Java.

consigue utilizando el mismo código original, heredando funciones y agregando nuevas características, sin alterar el original.

- **Es multiplataforma:** a diferencia de los programas escritos en otros lenguajes, que requieren de versiones específicas para cada sistema operativo, las aplicaciones desarrolladas en Java funcionan en cualquier entorno, dado que no es el sistema quien las ejecuta, sino la máquina virtual (conocida como Java Virtual Machine o JVM).
- **Su uso no implica realizar inversiones económicas:** programar en lenguaje Java es totalmente gratis, solo es necesario descargar el kit de desarrollo (Java Development Kit[8] o JDK) y comenzar el proyecto de realización del software.
- **Es de fuente abierta:** Java ofrece el código de la mayoría de sus librerías nativas para que los desarrolladores puedan conocerlas y estudiarlas en profundidad para ampliar su funcionalidad, beneficiándose toda la comunidad de programadores.
- **Es un lenguaje expandible:** se relaciona con la característica anterior, cada programador puede revisar y mejorar el código de Java, convirtiéndose su trabajo en solución a problemas de muchas personas.
- **Es muy potente:** en la actualidad con Java se puede desarrollar un centenar de aplicaciones de cualquier tamaño y fin. Se puede programar desde aplicaciones simples como calculadoras, hasta procesadores de texto, por ejemplo.
- **Gran número de librerías disponibles:** a las librerías nativas provistas por Oracle, se le suman las generadas por programadores independientes y empresas que sirven para fines diversos.
- **Soporte y documentación:** existen numerosas comunidades que centran su actividad en otorgar soporte al lenguaje Java y se encuentran distribuidas en diversos países con soporte para diversos idiomas. Además se destaca la documentación completa, que pone a disposición Oracle sobre cada una de las clases, métodos y componentes.

[8] Java Development kit: Kit de desarrollo de Java.

- **Lenguaje seguro:** uno de los objetivos de los creadores de Java fue crear un lenguaje que no pudiera acceder a los recursos del sistema de manera incontrolada. Es así que se eliminó la posibilidad de manipular la memoria a través del uso de punteros y la capacidad de transformación de números en direcciones de memoria (como se puede hacer en C) con lo que se evita todo acceso ilegal a la memoria. Esto se logra gracias a que el compilador de Java efectúa una verificación sistemática de las conversiones.
- **Soporte de diversos algoritmos criptográficos:** el lenguaje Java presenta dentro de su arquitectura componentes que facilitan el uso de algoritmos criptográficos de forma simple e intuitiva, esta característica por su gran importancia se describe en el título siguiente.

7.2 Características criptográficas de Java [10]

El lenguaje Java posee dentro de su arquitectura la API de Seguridad (Security API) que constituye una de sus interfaces más destacables, construida en torno al paquete *java.security.* La versión del JDK 1.1 incluyó entre sus características el esquema denominado JCA (Java Cryptography Architecture[9]), que entre algunas de sus funciones permitía la generación de firmas digitales y hashes de los mensajes[10] [22].

En versiones siguientes de la plataforma Java con la denominación J2SE (Java 2 Platform , Standard Edition[11]), extendió la funcionalidad del JCA basándolo en una arquitectura de tipo “proveedor” que permitiera la utilización de implementaciones de múltiples algoritmos criptográficos que puedan ser interoperables entre sí. Como resultado se lo introdujo como componente adicional al JCE[12] (Java Cryptography Extension).

JCE proporciona implementaciones para cifrado y descifrado de datos, generación y acuerdo de claves y algoritmos MAC (Message Authentication Code[13]). En un primer momento, JCE formó parte de un paquete opcional del software

[9] Java Cryptography Architecture: Arquitectura Criptográfica de Java
[10] Originalmente JCA solo contaba con algoritmos de Autenticación, sin algoritmos que se relacionen con cifrado y descifrado de datos.
[11] Standard Edition: Edición estándar
[12] Java Cryptography Extension: Extensión Criptográfica de Java
[13] Message Authentication Code: Código de autenticación de mensaje.

J2SE[14], pero desde la versión 1.4 se incluyó en el núcleo de la distribución. Es por ello que puede considerarse que los esquemas de JCA y JCE forman un único bloque, siendo referido este esquema de seguridad como el modelo JCA/JCE [15]. [22]

La arquitectura JCA/JCE fue diseñada con dos objetivos [23]:

- Independencia e interoperabilidad de implementación
- Independencia y extensibilidad algorítmica

La independencia de implementación y algorítmica son objetivos complementarios, se logran a través de la definición de motores o proveedores criptográficos. La Figura 5 muestra la utilización de los proveedores a través de un ejemplo en el que cierta aplicación necesita emplear el algoritmo AES: la aplicación no codifica su lógica, sino que solicita a uno de los proveedores criptográficos el uso de la implementación de dicho algoritmo.

Este esquema permite que cualquier aplicación que necesite usar un algoritmo, siempre utilizará las mismas clases y nombres identificativos, independientemente de quien haya implementado el proveedor criptográfico que esté siendo usado. Algunos ejemplos de las clases incluidas en dichos motores son MessageDigest[16] (generación de hashes), Signature[17] (creación y verificación de firmas digitales), KeyFactory[18] (generación de claves), Cipher [19](cifrado y descifrado de datos), MAC (cálculos de códigos MAC), entre otras.

La interoperabilidad de implementación significa que por ejemplo que para un mismo algoritmo, una clave generada por un proveedor puede ser usada por otro, del mismo modo que una firma generada por cierto proveedor puede ser verificada por cualquier otro.

Por último, extensibilidad algorítmica significa que se pueden agregar fácilmente nuevos algoritmos dentro de las clases soportadas por cada proveedor.

[14] Debido a las leyes de Estados Unidos de control de exportación de algoritmos criptográficos.
[15] Se flexibilizaron las leyes de control de exportación y SUN produjo cambios en el esquema de JCE: se utilizaron las llamadas firmas de proveedores y el uso de archivo de políticas restrictivas.
[16] MessageDigest: Resumen de mensajes.
[17] Signature: Firma
[18] Key Factory: generador de claves.
[19] Cipher: Cifrador

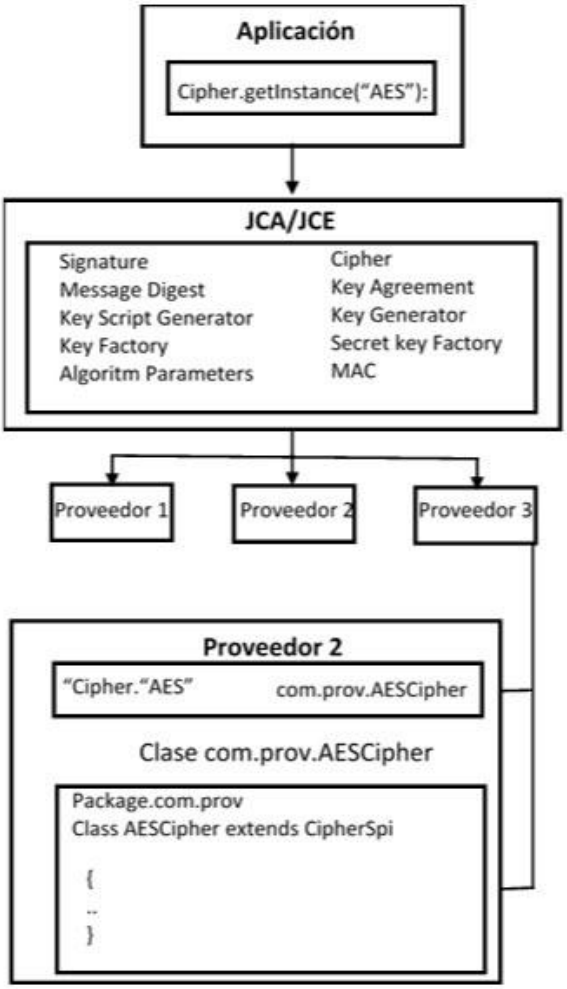

Figura 5: Proveedores criptográficos en Java [23]

7.2.1 Proveedores Criptográficos

La versión JAVA SE 7 incluye los siguientes proveedores criptográficos [24]:

- SunPKCS11
- SUN
- SunRsaSign
- SunJSSE
- SunJCE
- SunJGSS
- SunSASL
- XMLDSig
- SunPCSC
- SunMSCAPI

- SunEC

El motivo por el que se incluye un gran número de proveedores se debe a que inicialmente los motores criptográficos se crearon en forma independiente teniendo en cuenta necesidades muy específicas. Es por ello que el grado de solapamiento de los proveedores en relación a los algoritmos soportados es mínimo. Por ejemplo, el paquete SUN incluye algunos algoritmos de Hash como MD5, SHA-1, SHA-256, etc.; el paquete SunRsaSign se encarga de las implementaciones de firma digital RSA que emplean distintas funciones hash (por ejemplo MD5withRSA, SHA1withRSA, etc.), el paquete SUNJCE permite acceder a funciones de generación de códigos MAC (HmacSHA1, HmacSHA256, etc.).

Antes de la versión J2SE 5.0, la arquitectura JCA/JCE no incluía clases específicas para Curvas Elípticas, por lo que los desarrolladores se vieron obligados a adquirir software de terceros. Esta situación se revirtió a partir de la versión J2SE 5.0 se incluyeron algunas clases e interfaces para facilitar a los proveedores soporte para ECC.

A partir de la versión JSE 7, Oracle incluyó un proveedor criptográfico llamado SunEC. A continuación se dará una breve descripción de este proveedor junto con otros desarrollados por compañías de terceros, para luego culminar con la elección de uno de ellos para codificar el programa.

7.2.1.1 SunEC

Permite tamaños de claves de los 112 a los 571 bits. Implementa protocolos de acuerdo de claves ECDH y procedimientos de firma digital ECDSA: A través del protocolo ECDH produce un valor secreto que corresponde a la primera coordenada del punto de la curva elíptica calculado como producto de la llave privada de un usuario y la pública de otro. Mientras que la implementación de ECDSA permite las siguientes funciones hash: SHA-1, SHA-256, SHA-384 y SHA-512.

En las siguientes tablas se muestran los identificadores de curva elíptica de Java implementado a través de SunEC tanto de campos binarios como primos. Estas curvas están definidas en la primera versión de SECG SEC 2 [25] y ANSI X9.62 [26], y la segunda versión de NIST FIPS 186. [27].

En lo que se refiere a documentación, el proveedor de Sun solamente pudo encontrase un paper [28] de Diciembre de 2013 donde existe una descripción de las principales características y cuenta con tres programas de ejemplo de uso de la librería. Es decir carece de documentación oficial.

Sin embargo, en ese paper se concluye que tiene que actualizarse a los estándares más recientes, y que la principal falencia es que no implementa ECIES (aunque para fines de este proyecto no es necesario).

SECG SEC 2	ANSI X9.62 NIST	NIST FIPS 186-2
secp112r1		
secp112r2		
secp128r1		
secp128r2		
secp160k1		
secp160r1		
secp160r2		
secp192k1		
secp192r1	X9.62 prime192v1	NIST P-192
	X9.62 prime192v2	
	X9.62 prime192v3	
secp224k1		
secp224r1		NIST P-224
	X9.62 prime239v1	
	X9.62 prime239v2	
	X9.62 prime239v3	
secp256k1		
secp256r1	X9.62 prime256v1	NIST P-256
secp384r1		NIST P-384
secp521r1		NIST P-521

Tabla 3: Curvas elípticas disponibles en SunEC - Campos primos Fp [28]

SECG SEC 2	ANSI X9.62	NIST FIPS 186-2
sect113r1		
sect113r2		
sect131r1		
sect131r2		
sect163k1		NIST K-163
sect163r1		
sect163r2		NIST B-163
	X9.62 c2tnb191v1	

	X9.62 c2tnb191v2	
	X9.62 c2tnb191v3	
sect193r1		
sect193r2		
sect233k1		NIST K-233
sect233r1		NIST B-233
sect239k1		
	X9.62 c2tnb239v1	
	X9.62 c2tnb239v2	
	X9.62 c2tnb239v3	
sect283k1		NIST K-283
sect283r1		NIST B-283
	X9.62 c2tnb359v1	
sect409k1		NIST K-409
sect409r1		NIST B-409
	X9.62 c2tnb431r1	
sect571k1		NIST K-571
sect571r1		NIST B-571

Tabla 4: Curvas elípticas disponibles en SunEC - Campos Binarios [28]

7.2.2 Proveedores de terceros

Existen una gran diversidad de bibliotecas o librerías y módulos criptográficos disponibles en Internet creados con el objetivo de desarrollar programas de seguridad, por ejemplo:

- Bouncy Castle (www.boncycastle.org)
- Cryptix (www.cryptix.org)
- FlexiProvider (www.flexiprovider.de)

7.2.2.1 Cryptix [10]

Cryptix [29] se destaca por ser la primera biblioteca criptográfica de código libre para el desarrollo de aplicaciones Java. Logró implementar un proveedor JCE para Java compatible hasta la versión JDK 1.4 (Cryptix JCE), el estándar OpenPGP (Cryptix OpenPGP) y un paquete creado para la utilización de criptografía de curva elíptica (Cryptix Elliptix).

El proyecto fue cerrado en 2005 como lo indica su página web. Con el objetivo poder compararlo con otras librerías resulta interesante conocer cuál fue su grado de desarrollo. Su objetivo fue la implementación de los estándares IEEE 1363 [30], ANSI X9.62 [26] y ANSI X9.63 [31]. La versión final se incluía soporte para aritmética sobre $\mathbb{F}_p$ (con coordenadas projectivas) como sobre $\mathbb{F}_{2^m}$ (con coordenadas afines). Tenía el inconveniente de que no daba soporte para la generación de los parámetros de la curva y que no contaba con operaciones a alto nivel como la firma digital.

7.2.2.2 Boncy Castle [10]

Legion of the Bouncy Castle [32] está formado por un grupo de voluntarios que desarrolló implementaciones de código abierto de API criptográficas en los lenguajes Java y C#. Su última versión (la 1.50, publicada en diciembre de 2013), está organizada de manera de contener dos conjuntos de API:

- Una API independiente (llamada light-weight[20]) de bajo nivel que puede ser usada en cualquier entorno JAVA, sin necesidad de compatibilidad con la arquitectura JCE.
- Un proveedor criptográfico que se ajusta a la arquitectura JCA/JCE.

  Además la librería proporciona los siguientes servicios (se destacan solo algunos):
  - Una implementación de la arquitectura JCE 1.2.1.
  - Una biblioteca para la lectura y escritura de objetos ASN.1
  - Una API TLS Y DTLS para el modo cliente.
  - Generadores de certificados X.509 en sus versiones 1,2,3, generadores de listas de revocación de certificados (CRL) y archivos PKCS#12 (son utilizados para almacenar claves privadas con su certificado de clave pública).

En lo relacionado con criptografía de curvas elípticas, incluye implementaciones de ECDH, EDCSA y ECIES.

[20] Light-weight: peso ligero

Dentro del paquete org.bouncycastle.math se encuentran las siguientes clases importantes:

- ECCurve: representa una curva elíptica mediante su ecuación de Weirstrass.
- ECFieldElements: representa un elemento del cuerpo finito utilizado.
- ECPoint: representa los puntos de una curva elíptica e implementa la aritmética de puntos.

Todas las clases abstractas (ECCurve, ECFieldElements y ECPoint) son implementadas por dos subclases derivadas $\mathbb{F}_p$ y $\mathbb{F}_{2m}$ representando a las curvas definidas por cuerpos finitos y binarios respectivamente.

El sitio presenta información detallada sobre la versión actual del producto, además cuenta con notas de implementación que permiten saber cómo se utiliza la librería creada y algunos consejos de implementación. Existen en internet gran cantidad de páginas que contienen códigos de ejemplo usando la librería de Bouncy. Así mismo la página ofrece una suscripción a listas de correo especializadas, donde los usuarios exponen sus problemas frente a la comunidad. Además existe un libro [22], donde plantean casos de las implementaciones de sistemas criptográficos usando las API de Bouncy,

En las siguientes tablas se muestran los identificadores de curva elíptica de Java implementado a través de Bouncy Provider tanto de campos binarios como primos. Estas curvas están definidas en la primera versión de SECG SEC 2 [25] y ANSI X9.62 [26], la segunda versión de NIST FIPS 186. [27] y en el documento propuesto por Brain Pool [33].

SECG SEC 2	ANSI X9.62 NIST	ECC Brainpool	NIST 186-2
secp112r1			
secp112r2			
secp128r1			
secp128r2			
secp160r1			
secp160r2			
		brainpoolP160r1	
		brainpoolP160t1	
secp192r1			
	X9.62 prime192v1		
	X9.62 prime192v2		

	X9.62 prime192v3		
		brainpoolP192r1	
		brainpoolP192t1	
secp224r1			NIST P-224
		brainpoolP224r1	
		brainpoolP224t1	
	X9.62 prime239v1		
	X9.62 prime239v2		
	X9.62 prime239v3		
	X9.62 prime256v1		
secp256r1			NIST P-256
		brainpoolP256r1	
		brainpoolP256t1	
		brainpoolP320r1	
		brainpoolP320t1	
		brainpoolP384r1	
		brainpoolP384t1	
secp384r1			NIST P-384
		brainpoolP512r1	
		brainpoolP512t1	
secp521r1			NIST P-521

Tabla 5: Curvas elípticas disponibles en Bouncy Castle - Campos primos Fp

SECG SEC 2	ANSI X9.62	NIST FIPS 186-2
sect113r1		
sect113r2		
sect131r1		
sect131r2		
sect163k1		
sect163r1		
sect163r2		NIST B-163
	X9.62 c2pnb163v1	
	X9.62 c2pnb163v2	
	X9.62 c2pnb163v3	
	X9.62 c2pnb176w1	
	X9.62 c2tnb191v1	
	X9.62 c2tnb191v2	
	X9.62 c2tnb191v3	
sect193r1		
sect193r2		
	X9.62 c2pnb208w1	
sect233k1		
sect233r1		NIST B-233

sect239k1		
	X9.62 c2tnb239v1	
	X9.62 c2tnb239v2	
	X9.62 c2tnb239v3	
	X9.62 c2pnb272w1	
sect283k1		
sect283r1		NIST B-283
	X9.62 c2pnb304w1	
	X9.62 c2tnb359v1	
	X9.62 c2pnb368w1	
sect409k1		
sect409r1		NIST B-409
	X9.62 c2tnb431r1	
sect571k1		
sect571r1		NIST B-571

Tabla 6: Curvas elípticas disponibles en Bouncy Castle - Campos binarios

7.2.2.3 FlexiProvider [34]

Es un potente conjunto de herramientas de código libre creado para dar soporte a la arquitectura JCA/JCE de Java. Fue desarrollado por el Grupo de Investigación en Ciencias de la Computación de la Universidad Técnica de Darmstadt, Alemania.

Su objetivo es facilitar implementaciones rápidas y seguras de algoritmos criptográficos fáciles de usar, incluso para desarrolladores no expertos en criptografía [35].

Entre los paquetes provistos por el proyecto se destacan:

- CoreProvider

Incluye algoritmos de cifrado asimétrico (por ejemplo RSA), simétrico (AES, 3DES), funciones de Hash (MD5, SHA1) y de generación de códigos MAC (CBC-MAC, CMAC, HMAC).

- ECProvider

Es el que contiene los algoritmos criptográficos basados en curvas elípticas. Incluye esquema de firma digital (ECDSA y ECNR), acuerdo de claves Diffie-Hellman (ECDH) y el esquema de cifrado (ECIES). Estos algoritmos son implementados de acuerdo a los siguientes estándares.

  - ANSI X9.62 [26]: ECDSA

- IEEE 1363 [30] and 1363a [36]: ECDSA, ECNR, ECDH, ECIES
- Certicom SEC 1 [14], SEC 2 [13]: ECDSA, ECDH, ECIES

En su página web, además del código binario y documentación, incluye ejemplos y tutoriales sobre como emplear los paquetes nombrados anteriormente. Actualmente se encuentra disponible la versión FlexiProvider 1.7p7, lanzada el 20 de Enero de 2014.

En las siguientes tablas se muestran los identificadores de curva elíptica de Java implementado a través de FlexiProvider tanto de campos binarios como primos. Estas curvas están definidas en la primera versión de SECG SEC 2 [25] y ANSI X9.62 [26] y en el documento propuesto por Brain Pool [33].

SEC 2	ANSI X9.62	Brain Pool
secp112r1		
secp112r2		
secp128r1		
secp128r2		
secp160k1		
secp160r1		
secp160r2		
		brainpoolP160r1
		brainpoolP192r1
	X9.62 prime192v1	
	X9.62 prime192v2	
	X9.62 prime192v3	
secp192k1		
		brainpoolP224r1
secp224k1		
secp224r1		
	X9.62 prime239v1	
	X9.62 prime239v2	
	X9.62 prime239v3	
		brainpoolP256r1
	X9.62 prime256v1	
secp256k1		
		brainpoolP320r1
		brainpoolP384r1
secp384r1		
		brainpoolP512r1
secp521r1		

Tabla 7: Curvas elípticas disponibles en FlexiProvider - Campos primos Fp

ANSI X9.62
X9.62 c2pnb163v1
X9.62 c2pnb163v2
X9.62 c2pnb163v3
X9.62 c2tnb191v1
X9.62 c2tnb191v2
X9.62 c2tnb191v3
X9.62 c2pnb208w1
X9.62 c2tnb239v1
X9.62 c2tnb239v2
X9.62 c2tnb239v3
X9.62 c2pnb272w1
X9.62 c2tnb359v1
X9.62 c2pnb368w1
X9.62 c2tnb431r1

Tabla 8: Curvas elípticas disponibles en FlexiProvider - Campos Binarios

7.2.3 Justificación elección de Proveedor Criptográfico

Dentro de la elección de proveedores criptográficos queda de lado la librería Cryptix, ya que como se dijo el proyecto fue cerrado en 2005 y sólo fue incluido en la lista para enunciar que fue el primer proveedor para Java. Por ello, sólo se va a decidir entre los proveedores SunEC, Bouncy Castle y FlexiProvider.

El principal problema SunEC es su falta de documentación oficial, carencia que los restantes proveedores no tienen, ya que presentan una excelente documentación con ejemplos muy útiles de su uso.

Tanto Bouncy Castle como FlexiProvider, son librerías que están en constante actualización teniendo versiones con una antigüedad no superior a seis meses desde que se consultó en la Web.

Sin embargo el factor determinante para la elección de FlexiProvider se encuentra en el tiempo de ejecución. Para ello se codificaron dos programas de prueba, como puede verse en el Anexo D, donde se realiza un proceso de generación de claves basado en Diffie Hellman, utilizando como curva la secp521r1 del estándar SECG SEC 2 soportado por ambas APIs.

En la Tabla 9 se muestran los tiempos en nanosegundo (se utilizó la función System.nanoTime() para tales fines:

TEST	SunEC	Bouncy Castle	FlexiProvider
1	550271037	3521522861	2594965763
2	561870669	3647302243	2478870707
3	581901004	3455424160	2558550019
4	577515590	3393864858	2619573089
5	551617051	3457303235	2603514219
6	576173199	3553206724	2544065877
7	587523285	3327570503	2557586251
8	571463962	3577019214	2768468402
9	565183168	3410030158	2513932285
10	585049553	3640815380	2584571746
Promedio	**570856852**	**3498405934**	**2582409836**

Tabla 9: Tiempo de ejecución Bouncy vs Flexi Provider

Como puede observarse, para una serie de diez pruebas el tiempo de ejecución del programa de prueba usando FlexiProvider tomó 2582409836 nanosegundos, mientras que con Bouncy Castle tomó 3498405934 y para SunEC tomó 570856852 nanosegundos.

8 IMPLEMENTACIÓN DEL SISTEMA

8.1 Instalación de los archivos de políticas de JCE

Debido a restricciones en el control de importaciones de algunos países, la versión de los archivos de política de la extensión JCE, que se encuentran en la JRE 7, contiene limitaciones respecto del tamaño de claves. Es así que solo permite en general 128 bits (se incluyen las curvas elípticas) y para el algoritmo RSA un límite de 2048 bits.

Para poder disponer de los tamaños de claves provistos por los distintos proveedores criptográficos, es necesario copiar dentro de la carpeta donde se instaló el JRE, los siguientes archivos:

- local_policy.jar
- US_export_policy.jar

Los mismos se pueden descargar de:

http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html

Para el caso de Windows, como se ilustra en la Figura 6 el directorio es: java-home\lib\security

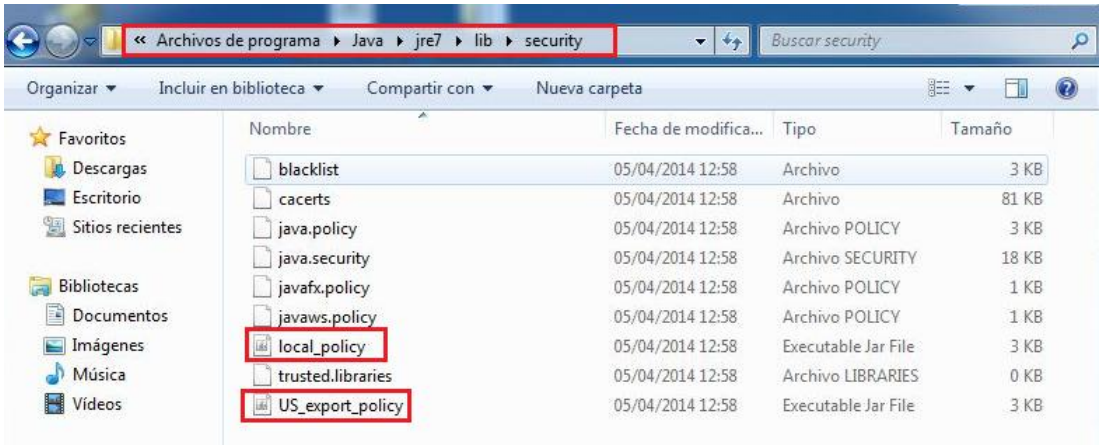

Figura 6: Archivos de extensión de políticas

8.2 Instalación y configuración del proveedor Criptográfico FlexiProvider

Para poder instalar un proveedor criptográfico adicional, se deben descargar los archivos de la página del proveedor y colocarlos dentro de la carpeta de instalación del JRE.

Para el caso del FlexiProvider, descargar de la página: http://www.flexiprovider.de/download.html, los siguientes archivos:

- CoDec-build21-jdk13.jar
- FlexiECProvider-1.7p7.signed.jar
- FlexiProvider-1.7p7.signed.jar

Y copiarlos en java-home\lib\ext como se observa en la Figura 7.

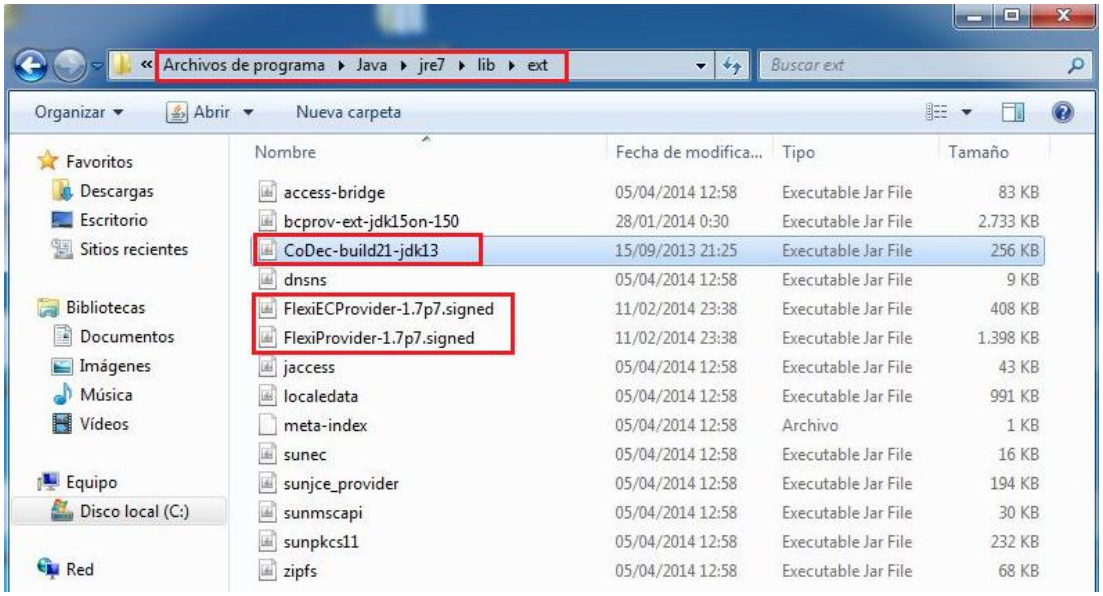

Figura 7: Proveedores criptográficos

Luego se debe agregar las siguientes líneas en el archivo java.security ubicado en java-home\lib\security

security.provider.11=de.flexiprovider.core.FlexiCoreProvider
security.provider.12=de.flexiprovider.ec.FlexiECProvider

Observar que llevan los números 11 y 12, ya que Java por defecto tiene preinstalado otros proveedores criptográficos.

8.3 Descripción del programa

- Elección de los parámetros de curva y tamaño de clave

Se eligió a SEC 2 como estándar para la provisión de los parámetros de curvas elípticas. Teniendo presente el proveedor criptográfico elegido (FlexiProvider), se escogió la curva definida sobre campos primos que brinde la mayor longitud de clave posible. Quedó definida así la curva secp521r1 (donde sec, hace referencia al Estándar SEC 2, 521 es el tamaño en bits de la clave, y r indica que es del tipo de parámetros elegidos en forma aleatoria. Ver Anexo C donde se encuentran los valores de los parámetros.

Como se explicó en la sección anterior, se debe tomar un tamaño de clave múltiplo de 16 bits. Es por ello que se truncó la clave a 512 bits, lo que permite un mensaje 512/16(tamaño de un char)=32 caracteres. Se utilizó como carácter relleno el símbolo #.

- Arquitectura Cliente-Servidor

La implementación en Java se basó en el uso de socket. Los socket no son más que puntos o mecanismos de comunicación entre procesos que permiten que un proceso se comunique (emita o reciba información) con otro proceso, incluso estando estos procesos en distintas computadoras. Un socket queda definido por una dirección IP, un protocolo (se usará TCP) y un número de puerto.

Debido a que la comunicación entre procesos a través de socket se basa en la filosofía Cliente-Servidor, un usuario actuará como servidor creando un socket cuya IP y puerto debe conocer el cliente.

- Inicio y fin de transmisión de mensajes

Una vez que el servidor creó el socket y el cliente se conectó al mismo, se produce en forma inmediata el primer intercambio de claves y generación de llave compartida. Se eligió en forma arbitraria que el servidor será el autorizado a enviar el primer mensaje, por lo que el cliente queda a la espera del mismo.

El mensaje que marca el fin de la conversación es: “FIN!”. Este puede ser enviado tanto por el servidor como por el cliente. Cuando se envía este mensaje, a ambos usuarios se les inhabilita el campo para enviar texto y se le informa que deben presionar el botón Salir para abandonar el programa.

- Estructura del programa

Tanto la implementación para servidor como para cliente manejan dos hilos cada una. Un hilo de ejecución maneja la interfaz gráfica que se encarga de la entrada y salida de datos por teclado y por pantalla, mientras que el otro maneja todos los procesos de conexión, y manipulación de datos (cifrado y descifrado).

- Interfaz gráfica del servidor y el cliente.

En la Figura 8 se ilustran las interfaces respectivas del servidor y cliente, y se indica al pie de ella las funciones de cada elemento.

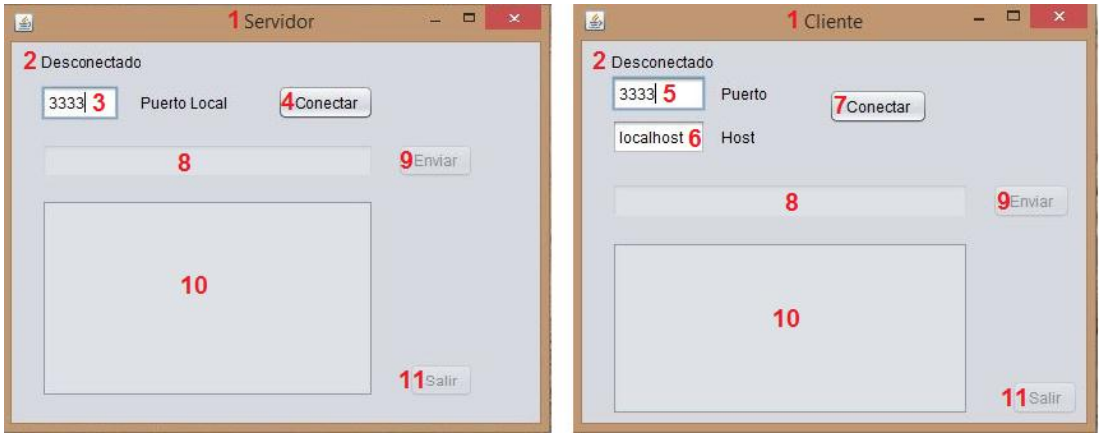

Figura 8: Interfaz gráfica Cliente-Servidor

1. Barra de título: Indica si es el usuario Cliente o Servidor.
2. Etiqueta de estado: muestra información sobre estado de conexión, intercambio de claves y rol.
3. Número de puerto (Servidor): cuadro de texto que permite al usuario ingresar un valor de puerto local que desea abrir para dejarlo a la escucha de algún cliente. Tiene un valor restringido de 3000 a 5000.
4. Botón Conectar (Servidor): una vez ingresado 3 crea el socket.
5. Número de puerto (Cliente): indica el puerto remoto al que se conectará.
6. Dirección de host (Cliente): indica la dirección del host remoto al que se conectará, puede ser por IP o por nombre.
7. Botón Conectar (Cliente): una vez colocado 5 y 6, se conecta con el proceso remoto para crear el socket.
8. Campo de texto: permite el ingreso de un mensaje de 32 caracteres.
9. Botón Enviar: una vez ingresado 8 envía el mensaje.
10. Área de texto: muestra los mensajes enviados y recibidos.

11. Botón Salir: permite abandonar el programa.

Notas:

3 y 4 del lado del servidor y 5,6 y 7 del lado del cliente se desactivan cuando se la conexión se produce exitosamente.

8,9 y 10 se habilitan cuando el usuario tiene el turno de enviar mensaje y se deshabilitan cuando no tiene este turno y cuando se envía o recibe el mensaje de finalización de la conversación.

11 se habilita cuando se envía o recibe el mensaje de finalización de conversación.

- Diagrama de Flujo

En Figura 9 se muestra un esquema del hilo que maneja la conexión, flujos de entrada/salida y manipulación de datos (cifrar y descifrar). En la Figura 10 se desglosan algunas funciones del diagrama general.

- Otras características a destacar

Puede decirse que el OTP modificado implementado en el lenguaje Java, constituye un sistema de cifrado por bloques, ya que en cada intercambio de mensajes se toman elementos de tamaño igual a 32 bytes. Sin embargo este bloque es cifrado a través de la operación de Xor de cada uno de sus bits con la clave generada.

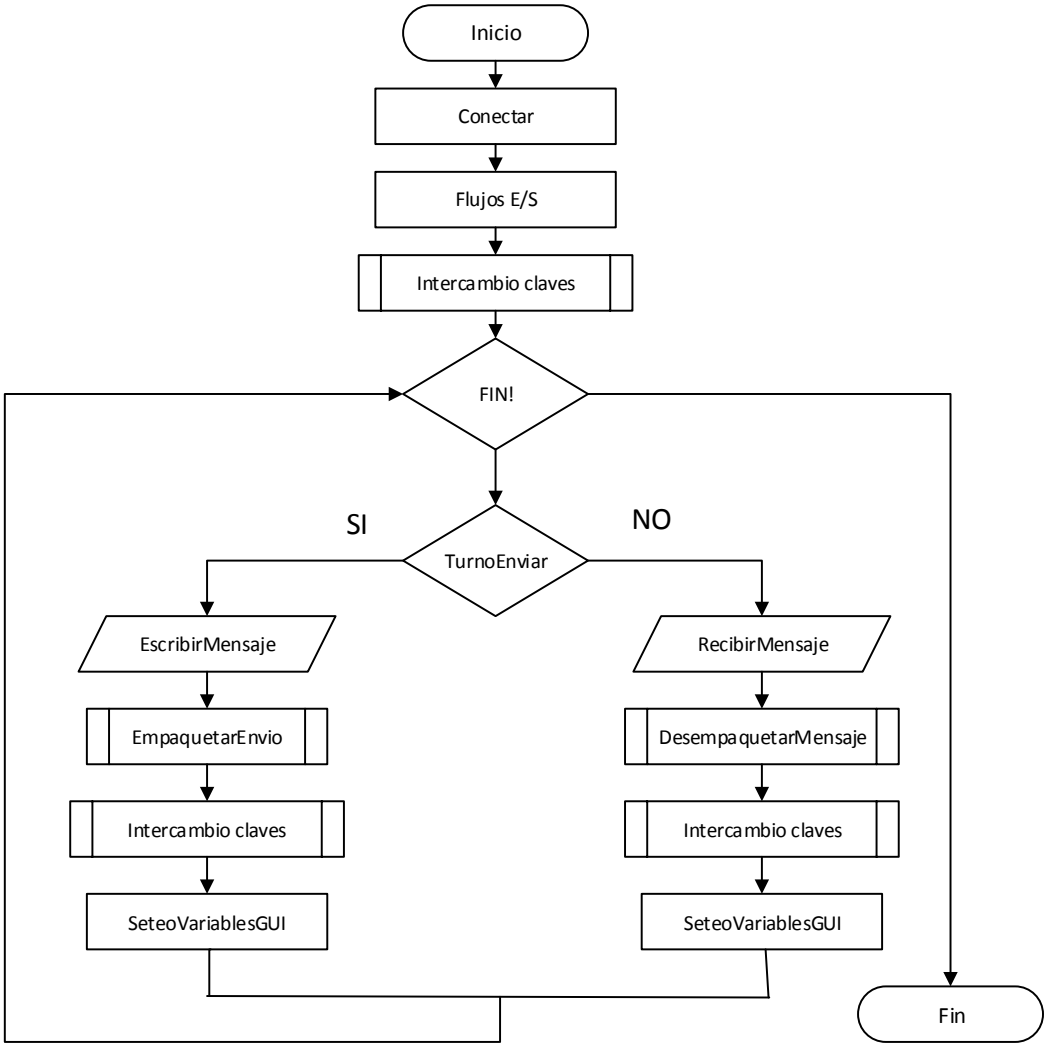

Figura 9: Diagrama de Flujo General

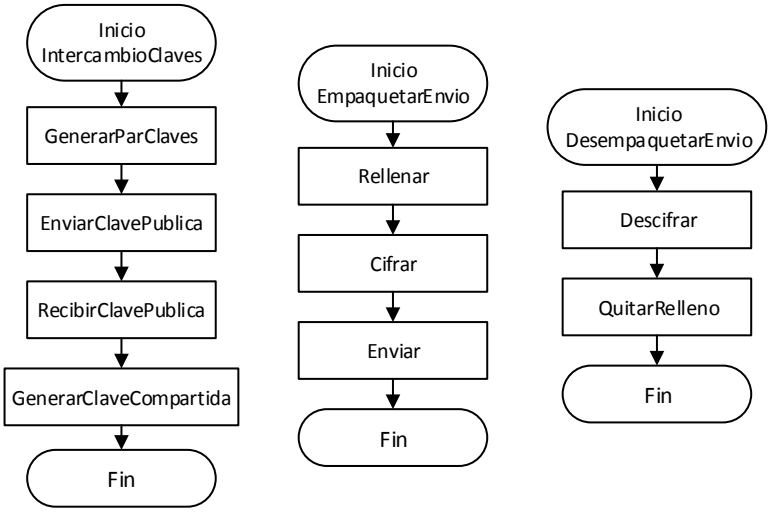

Figura 10: Diagramas de flujos varios

8.3.1 Pseudoaleatoriedad y generación de claves en Java

A continuación se dará una breve reseña del código utilizado para efectuar el intercambio ECDH con la certeza de que se usan números pseudoaleatorios.

```
1 private PublicKey serverPublicKey;
2 private KeyAgreement aKeyAgree;
3 private KeyPair aPair;
4
5 public void generarParClaves (){
6    try {
7       KeyPairGenerator keyGen = KeyPairGenerator.getInstance("ECDH", "FlexiEC");
8       CurveParams ecParams = new CurveRegistry.Secp521r1();
9       keyGen.initialize(ecParams, new SecureRandom());
10
11       aKeyAgree = KeyAgreement.getInstance("ECDH", "FlexiEC");
12       aPair=keyGen.generateKeyPair();
13       serPubkey=aPair.getPublic().getEncoded();
14       }
15    catch (Exception e){
16       System.out.println("Error al generar par de claves");
17       cerrarConexion();
18       }
19    }
20
```

La función generarPardeClaves realiza lo siguiente:

En la línea 7, se llama al método estático getInstance de la clase KeyPairGenerator, con el nombre del algoritmo ("ECDH") y el proveedor criptográfico ("FlexiEC"), para obtener un generador de par de claves pública/privada del algoritmo especificado.

En la línea 8, se obtiene un conjunto de parámetros de dominio de Curvas elípticas usado para el generador de par de claves.

En la línea 9, se inicializa el generador de par de claves con los parámetros de dominios y la fuente de números pseudoaleatorios.

- Clase SecureRandmom

Según la documentación de Java, "esta clase provee un generador de números aleatorios (pseudoaleatorios) criptográficamente fuertes (RNG).

Estos cumplen mínimamente con los test estadísticos de generadores de números aleatorios especificadas en FIPS 140-2, Requerimientos de Seguridad para Módulos Criptográficos, sección 4.9.1. Adicionalmente, SecureRandom produce una salida no determinística. Por lo tanto cualquier semilla pasada al objeto SecureRandmom debe ser impredecible, y todas las secuencias de salida deben ser criptográficamente robustas como se describe en el RFC 1750: Recomendaciones de Aleatoriedad para la Seguridad." [37]

Además afirma que "muchas implementaciones se encuentran en forma de generador de números pseudoaleatorios (PRNG), que significa que utilizan un algoritmo determinístico para producir una salida pseudoaleatoria de una semilla verdaderamente aleatoria" [37]. Este es justamente el caso que se aplica a la implementación mostrada.

En la línea 11, se llama al método estático getInstance de la clase KeyAgreement, con el nombre del algoritmo ("ECDH") y el proveedor criptográfico ("FlexiEC"), para obtener un objeto que maneje el protocolo de intercambio de claves del algoritmo especificado.

En la línea 12, se llama al método generateKeyPair para generar el par de claves pública/privada.

En la línea 13, con la combinación de los métodos getPublic y getEncoded se consigue la llave pública con formato X.509[21] SubjectPublicKeyInfo de un flujo de bytes.

```
21 public void enviarClavePublica() {
22    try{
23       oos.writeInt(serPubkey.length);
24       oos.write(serPubkey);
25       }
26    catch(IOException e){
27       System.out.println("Error en el envio de la clave pública");
28       cerrarConexion();
```

[21] X509: es un estándar UIT-T para infraestructuras de claves públicas

```
29      }
30    }
31
```

La función enviarClavePublica, básicamente lo que realiza es escribir en el buffer de Salida, el tamaño y el valor de la clave pública.

```
32 public void recibirClavePublica(){
33    try{
34       byte[] keyBytes = new byte[ois.readInt()];
35       ois.readFully(keyBytes);
36       KeyFactory kf = KeyFactory.getInstance("ECDH", "FlexiEC");
37       X509EncodedKeySpec x509Spec = new X509EncodedKeySpec(keyBytes);
38       serverPublicKey = kf.generatePublic(x509Spec);
39       }
40    catch(IOException e){
41       System.out.println("Error al recibir la clave publica");
42       cerrarConexion();
43   }
44 }
45
```

La función recibirClavePublica efectúa lo siguiente:

En las líneas 34 y 35 se lee el buffer de entrada y se le asigna a la variable keyBytes el valor que constituye la clave pública.

En la línea 36, se llama al método estático getInstance de la clase KeyFactory, con el nombre del algoritmo ("ECDH") y el proveedor criptográfico ("FlexiEC"), para obtener un objeto que es capaz de convertir claves “opacadas” (con formato X.509 SubjectPublicKeyInfo o PKCS#8) a una representación “transparente” y viceversa.

En las líneas 37 y 38, se obtiene la clave pública de la contraparte, primero construyendo un objeto de tipo X509EncodedKeySpec y luego utilizando el método generatePublic de la clase KeyFactory.

```
46 public void generarClaveCompartida(){
47    try{
48       aKeyAgree.init(aPair.getPrivate());
```

```
49      aKeyAgree.doPhase(serverPublicKey,true);
50      sharedKeybyte=aKeyAgree.generateSecret();
51        }
52    catch(Exception e){
53    System.out.println("Error al generar la clave compartida");
54    cerrarConexion();
55    }
56 }
```

Por último se cuenta con la función generarClaveCompartida, que efectúa lo siguiente:

En la línea 48, se da inicio al proceso de acuerdo de clave, donde se le pasa como parámetro la clave privada que contiene todos los parámetros del algoritmo empleado, en este caso ECDH.

En la línea 49, tomando como parámetro la clave pública de la contraparte, se inicia la siguiente fase del protocolo de acuerdo de claves.

Finalmente en la línea 50, se obtiene en formato de flujo de bytes la clave compartida.

9 CONCLUSIONES

No existe duda que actualmente la humanidad vive en la era de las comunicaciones. Del mismo modo que el hombre utiliza los medios masivos de comunicación para expresar en forma pública información, consiguiendo que llegue a un gran número de personas, también tiene la necesidad de que ciertas comunicaciones sean exclusivamente privadas y los mensajes conserven la confidencialidad siendo solo conocidos por el emisor y receptor.

Esta situación tuvo su máxima expresión a partir de la Segunda Guerra Mundial, donde los países necesitaban enviar códigos secretos. Es a partir de este momento que la criptología comienza a jugar un papel fundamental en la vida de las personas.

En la cima de la evolución de los medios de comunicación se encuentra Internet, que permite la transmisión instantánea de información desde y hacia cualquier parte del planeta. Surgen entonces los requerimientos de lograr que esta transmisión sea segura.

El eje principal del trabajo fue realizar una implementación de un sistema criptográfico que se considere seguro y sea posible su uso a través de la red. Esto confluyó en la necesidad de justificar el concepto de seguridad.

Se comenzó definiendo el algoritmo OTP que es el único que verifica la propiedad de seguridad perfecta dada por Shannon, sin embargo este enfoque es demasiado teórico y muy complejo de llevarlo a la práctica porque exige que el tamaño de la clave sea por lo menos igual al tamaño del mensaje y que las claves usadas una vez sean totalmente aleatorias.

Para poder llegar a una implementación práctica y poder justificar su seguridad, se emplearon, en su lugar, conceptos más "suaves" o "blandos". Es así que se habló de la seguridad computacional y la probabilidad razonable que un algoritmo sea quebrado en un tiempo muy grande no implica que el sistema sea inseguro, sino que es otro punto de vista de análisis.

Luego se abordó el concepto de seguridad semántica, que implica que el conocimiento del texto cifrado y su tamaño no revelan información adicional (que pueda ser extraída en forma sencilla) sobre el mensaje.

Sin embargo este término es complicado de probar en un sistema, por lo que se utilizó la propiedad de indistinguibilidad y el uso de un teorema que establece la relación entre ambas.

Contando con los conceptos de seguridad que pueden aplicarse en la práctica, se procedió a la construcción de un sistema que sea seguro semánticamente. Por ello se propuso el sistema OTP modificado, donde las claves aleatorias están generadas por intercambios Diffie Hellman de curvas elípticas, aprovechando entre otras cosas la fortaleza de las mismas frente a los ataques criptográficos.

Por ello se desarrolló en forma resumida las principales ideas que permiten entender la teoría de las curvas elípticas desde la aritmética modular, campos primos y binarios hasta llegar a definir que es una curva elíptica y la estructura de grupo para poder emplearse en criptografía.

Para la implementación del sistema se empleó el lenguaje de programación Java, que resultó tener una gran potencia y facilidad de uso en cuanto al manejo de algoritmos de seguridad. Esto se puso de manifiesto al explicar la estructura criptográfica de Java denominada JCA/JCE, y los distintos proveedores que suministran la implementación de los algoritmos criptográficos.

Como resultado de la realización del proyecto se lograron profundizar los conceptos anteriormente citados, en especial los asociados a curvas elípticas y se consiguió la implementación simple del sistema criptográfico propuesto codificado en Java.

A pesar del avance vertiginoso de la tecnología en los últimos años, en especial reflejada en el incremento del poder de cómputo, el sistema OTP ideado por Vernam conserva la propiedad de ser el único que posee la seguridad perfecta pero con la limitante de su complicada implementación. Se mostró a lo largo del trabajo que es posible suavizar los conceptos asociados a la seguridad y construir sistemas que sean prácticamente o semánticamente seguros.

Como propuesta a futuro se puede mejorar la implementación del sistema OTP modificado agregando la característica de integridad y autenticación. Así mismo se puede pensar en reemplazar las curvas elípticas por generadores pseudoaleatorios, que tengan como semilla por ejemplo números basados en cronómetros sincronizados.

10 BIBLIOGRAFÍA

[1] J. B. Vidales, Diseño e implementación de un esquema de cifrado híbrido basado en DHIES, 2008.

[2] J. Katz y . Y. Lindell, Introduction to Modern Cryptography, Chapman & Hall/CRC Cryptography and Network Security Series, 2007.

[3] C. Shannon, «Communication Theory of Secrecy Systems,» Bell System Technical Journal 28, p. 656–715, 1949.

[4] S. Goldwasser y . S. Micali, Probabilistic encryption & how to play mental poker keeping secret all partial information, Nueva York: STOC 82, 1982, p. 365–377.

[5] S. Goldwasser y . S. Micali, Probabilistic encryption, 1984, p. 270–299.

[6] E. R. Vera Aguilar, «Autenticación de dispositivos móviles basada en criptografía de curvas elípticas,» 2007. http://bibliotecavirtual.dgb.umich.mx:8083/jspui/handle/123456789/5730. [Último acceso: 29 Octubre 2013].

[7] NSA, «The Case for Elliptic Curve Cryptography,» 15 Enero 2009. http://www.nsa.gov/business/programs/elliptic_curve.shtml. [Último acceso: 1 Julio 2013].

[8] N. Koblitz, «Elliptic curve cryptosystems,» Math. Comp., nº 48, pp. 203-209, 1987.

[9] V. S. Miller, «Use of elliptic curves in cryptography,» Computer Science, vol. 218, pp. 417-426 , 1986.

[10] V. G. Martínez, Implementación en tarjetas inteligentes Java Card de protocolos de cifrado y descifrado basados en curvas elípticas, Madrid, 2010.

[11] J. M. García García, 2003. http://www.repositoriodigital.ipn.mx/handle/123456789/7845. [Último acceso: 2013 Noviembre 07].

[12] D. Neustadter y T. St Denis, Elliptic Curves over Prime and Binary Fields in Cryptography, 2008.

[13] Standards for Efficient Cryptography Group, «Recommended Elliptic Curve Domain Parameters. SECG SEC2 version 2.0,» http://www.secg.org/download/aid-784/sec2-v2.pdf. [Último acceso: 5 Julio 2013].

[14] Standards for Efficient Cryptography Group, «Recommended Elliptic Curve Domain Parametres. SECG SEC 1 version 2.0,» http://www.secg.org/download/aid-780/sec1-v2.pdf. [Último acceso: 5 Julio 2013].

[15] R. T. Trujillo, «Protocolo Criptográfico soportado en la Teoría d Curvas Elípticas,» 2011. http://revistas.mes.edu.cu/greenstone/collect/repo/import/repo/20120830/71208074010.pdf. [Último acceso: 03 Agosto 2013].

[16] F. A. F. Agulló, «Sistemas criptográficos de curva elíptica basados en

matrices,» 2005. http://rua.ua.es/dspace/bitstream/10045/11218/1/Ferr%C3%A1ndez%20Agul%C3%B3,%20Francisco%20Antonio.pdf. [Último acceso: 11 Enero 2014].

[17] J. Silverman y J. Suzuki, Elliptic curve discrete logarithm problem and the index calculus, Lecture Notes in Computer, 1999.

[18] W. Diffie y M. E. Hellman, New directions in cryptography, vol. 22, T. o. I. Theory, Ed., IEEE, 1976, p. 644–654.

[19] G. Belingueres, «Introducción a los criptosistemas de curva eliptica,» 2001. http://www.publispain.com/supertutoriales/matematica/criptografia/cursos/2/curva.pdf. [Último acceso: 12 Enero 2014].

[20] «http://monillo007.blogspot.com/2012/08/10-razones-para-elegir-java-como.html,» [En línea]. [Último acceso: 17 Febrero 2014].

[21] http://definicion.de/java/, «Definicion.de,» [En línea]. [Último acceso: 17 Febrero 2014].

[22] D. Hook, Beginning cryptography with Java, Indianapolis: Wiley Publishing, 2005.

[23] Oracle, «Java ™ Cryptography Architecture (JCA) Reference Guide,» http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html#SimpleEncrEx. [Último acceso: 01 Febrero 2014].

[24] Oracle, «Java Cryptography Architecture Oracle Providers Documentation for Java Platform Standard Edition 7,» http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html. [Último acceso: 22 Febrero 2014].

[25] Standards for Efficient Cryptography Group, «Recommended Elliptic Curve Domain Parameters. SECG SEC 2 version 1.0,» http://www.secg.org/collateral/sec2_final.pdf. [Último acceso: 2013 Julio 2013].

[26] American National Standards Insitute, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorthm (ECDSA). ANSI X9.62, 1998.

[27] National Institute of Standards and Tecnology, Digital Signature Standard (DSS). NIST FIPS 186-2, 2000.

[28] V. G. Martínez, «Implementing ECC with Java Standard Edition,» http://www.academicpub.org/DownLoadPaper.aspx?PaperID=14496. [Último acceso: 15 Diciembre 2013].

[29] Cryptix, «The Cryptix project,» http://www.cryptix.org. [Último acceso: 22 Febrero 2014].

[30] Institute of Electrical and Electronics Engineers (IEEE), «Standard specifications for public key cryptography. IEEE std 1363.,» 2000. http://grouper.ieee.org/groups/1363/P1363/index.html. [Último acceso: 22 Febrero 2014].

[31] American National Standards Insitute (ANSI), Public key cryptography for the financial services industry: Key agreemente and key transport using Elliptic Curve Cryptography. X9.63, 2001.

[32] Bouncy Castle, «The legion of the Bouncy Castle,» https://www.bouncycastle.org. [Último acceso: 23 Febrero 2014].

[33] «ECC Brainpool Standard Curves and Curve Generation v. 1.0,» http://www.ecc-brainpool.org/download/Domain-parameters.pdf. [Último acceso: 23 Febrero 2014].

[34] T. U. Darmstadt, «FlexiProvider, Harnessing the power of the Java Cryptography Architecture,» http://www.flexiprovider.de/. [Último acceso: 01 Febrero 2014].

[35] U. T. d. Darmstadt, «Flexy Provider,» http://www.flexiprovider.de/overview.html. [Último acceso: 23 Febrero 2014].

[36] Institute of Electrical and Electronics Engineers, «IEEE Standard Specifications for Public-Key Cryptography- Amendment 1: Additional Techniques,» http://grouper.ieee.org/groups/1363/P1363a/draft.html. [Último acceso: 23 Febrero 2014].

[37] Oracle, «Java SE Documentation,» http://www.oracle.com/technetwork/java/javase/documentation/index.html. [Último acceso: 19 Abril 2014].

[38] D. Hankerson, A. J. Menezes y S. Vanstone, Guide to Elliptic Curve Cryptography (Springer Professional Computing), Springer, 2004.

[39] J. H. Silverman, The Arithmetic of Elliptic Curves, USA: Editorial Board, 2008.

[40] L. c. Washington, Elliptic Curves Number Theory and Cryptography, College Park, Maryland, U.S.A: University of Maryland, 2008.

[41] M. Vilas, «One-time pad encryption in Python,» 17 Febrero 2010. http://breakingcode.wordpress.com/2010/02/17/one-time-pad-encryption-in-python/. [Último acceso: 1 Julio 2013].

[42] J. Tingley, «One Time Pad Encryption/Decryption GUI Client,» https://code.google.com/p/onetimepadjava/. [Último acceso: 1 Julio 2013].

[43] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C., John Wiley and Sons, Inc., 1996.

[44] Y. Saleem, M. Amjad, M. H. Rahman, F. Hayat, T. Izhar y M. Saleem, «Speech Encryption Implementation of One Time Pad algorithm in Matlab,» 1 Marzo 2013. http://www.paas.com.pk/images/volume/pdf/1546326107-21.pdf.

[45] D. Rijmenants, «Secure Communications with the One Time Pad Cipher,» http://users.telenet.be/d.rijmenants/papers/one_time_pad.pdf. [Último acceso: 30 Junio 2013].

[46] D. Rijmenants. http://users.telenet.be/d.rijmenants/en/onetimepad.htm. [Último acceso: 30 Junio 2013].

[47] D. Rijmenants, «Is One-time Pad History?,» http://users.telenet.be/d.rijmenants/papers/is_one_time_pad_history.pdf. [Último acceso: 30 Junio 2013].

[48] MuhammadYasirMalik, «Efficient Implementation of Elliptic Curve Cryptography Using Low-power Digital Signal Processor,» http://arxiv.org/ftp/arxiv/papers/1109/1109.1877.pdf. [Último acceso: 3 Julio

2013].

[49] A. MS, «Elliptic Curve Cryptography An Implementation Guide,» http://www.infosecwriters.com/text_resources/pdf/Elliptic_Curve_AnnopMS.pdf. [Último acceso: 1 Julio 2013].

[50] S. N. Molotkov, «Quantum cryptography and V A Kotel'nikov's one-time key and sampling theorems,» 2006.

[51] A. J. Menezes, P. C. van Oorschot y S. A. Vanstone, Handbook of Applied Cryptography, Boca Raton, New York, London, Tokyo: CRC Press, 1997.

[52] F. Martignon, «Lecture 3 - One Time Pad,» https://www.lri.fr/~fmartignon/documenti/systemesecurite/3-OneTimePad.pdf. [Último acceso: 2 Julio 2013].

[53] J. Markoff, «Codebook Shows an Encryption Form Dates Back to Telegraphs,» New York Times, 2011.

[54] J. Lopez, «Criptossistemas de Curvas Ellipticas,» http://www.lca.ic.unicamp.br/~jlopez/ecc.pdf. [Último acceso: 3 Julio 2013].

[55] KRISTINLAUTER, «The Advantages of Elliptic Curve Cryptography for Wireless Security,» Febrero 2004. http://research.microsoft.com/en-us/um/people/klauter/ieeefinal.pdf. [Último acceso: 3 Julio 2013].

[56] S. Goldwasser y S. Micali, «Probabilistic encryption & how to play mental poker keeping secret all partial information,» Annual ACM Symposium on Theory of Computing, 1982.

[57] S. D. Deenadayalan, «Vernam One-time Pad Encryption using Diffie Hellman Key Exchange protocol,» http://diamond.boisestate.edu/~liljanab/BOISECRYPTFall09/SriDivya.pdf. [Último acceso: 1 Julio 2013].

[58] S. D. Deenadayalan, «Vernam One-time Pad Encryption using Diffie Hellman Key Exchange protocol,» http://diamond.boisestate.edu/~liljanab/BOISECRYPTFall09/SriDivya.pdf. [Último acceso: 29 Junio 2013].

[59] M. Danter, «Elliptic Curve PHP-OOP DSA and Diffie-Hellman,» Diciembre 2010. http://matejdanter.com/2010/12/elliptic-curve-php-oop-dsa-and-diffie-hellman/. [Último acceso: 1 Julio 2013].

[60] J. Connelly, «A Practical Implementation of a One-time Pad Cryptosystem,» 11 Junio 2008. http://imotp.sourceforge.net/easyotp-report.pdf. [Último acceso: 30 Junio 2013].

[61] E. Barker, D. Johnson y M. Smid, «Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography,» NIST, http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-56Arev1_3-8-07.pdf. [Último acceso: 5 Julio 2013].

[62] N. Altun, «C# Implementation of One-Time Padding(OTP),» http://nazmialtun.blogspot.com.ar/2010/11/c-implementation-of-one-time-paddingotp.html. [Último acceso: 1 Julio 2013].

[63] J. Guajardo, R. Blümel, K. Uwe y C. Paa, «Efficient Implementation of Elliptic

Curve Cryptosystems on the TI MSP430x33x Family of Microcontrollers,» http://link.springer.com/content/pdf/10.1007%2F3-540-44586-2_27.pdf. [Último acceso: 3 Julio 2013].

[64] Certicom Corporation, «The National Security Agency's ECC License Agreement with Certicom Corp.,» http://www.certicom.com/images/pdfs/FAQ-TheNSAECCLicenseAgreement.pdf . [Último acceso: 5 Julio 2013].

[65] National Security Asociation, «Suite B Cryptography / Cryptographic Interoperability,» http://www.nsa.gov/ia/programs/suiteb_cryptography/. [Último acceso: 4 Julio 2013].

[66] NIST, «Recommended Elliptic Curves for Federal Government Use,» Julio 1999. http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf. [Último acceso: 3 Julio 2013].

[67] NSA, «Elliptic Curve Cryptography (ECC) Patent License Agreement (PLA) Information,» http://www.nsa.gov/business/programs/quick_facts.shtml. [Último acceso: 5 Julio 2013].

[68] Certicom, «Elliptic Curve Cryptography (ECC),» http://www.certicom.com/index.php/ecc. [Último acceso: 5 Julio 2013].

[69] «Coding center,» http://pjcodingcenter.blogspot.com.ar/2013/03/implementing-vernam-cipher-using-diffie.html. [Último acceso: 29 Junio 2013].

[70] D. Hook, Beginning Cryptography with Java, 2009.

[71] G. J. C. Apaza, «Seguridad en la autenticación de usuarios mediante firmas digitales basado en criptografía de curvas elípticas.,» http://bibliotecadigital.umsa.bo:8080/rddu/bitstream/123456789/1504/1/T-1938.pdf. [Último acceso: 2013 Agosto 25].

[72] Varios, «Claude Shannon,» http://www.nyu.edu/pages/linguistics/courses/v610003/shan.html. [Último acceso: 15 Abril 2014].

[73] F. Petitcolas, «La Cryptographie Militaire,» http://www.petitcolas.net/fabien/kerckhoffs/#english. [Último acceso: 15 Abril 2014].

[74] J. Ashley, «Elliptic Curve Diffie Hellman Cryptography,» 30 Abril 2007. http://www.codeproject.com/Articles/18610/Elliptic-Curve-Diffie-Hellman-Cryptography. [Último acceso: 1 Julio 2013].

[75] D. Kahn, The Codebreakers, Macmillan, 1967.

[76] M. Y. Malik, «Efficient Implementation of EllipticCurve Cryptograph yUsing Low-power Digital Signal Processor,» 2011. http://arxiv.org/ftp/arxiv/papers/1109/1109.1877.pdf. [Último acceso: 3 Julio

2013].

ANEXO A: ARITMÉTICA MODULAR [15]

La aritmética modular es una rama de las matemáticas que provee numerosas herramientas que son de utilidad para la criptografía. Permite realizar cálculos complejos manteniendo una representación numérica compacta y posibilita el trabajo con un conjunto finito de números enteros. A continuación se ofrecen algunos conceptos.

Definición: El conjunto de los enteros modulo n, denotado por $\mathbb{Z}_n$ es el conjunto formado por los números enteros $\{0,1,2,\ldots.,n-1\}$. La suma, resta y multiplicación en el conjunto son realizadas módulo n.

Definición: Dados los números enteros a, b y n, se dice que a es congruente con b módulo n y se escribe $a \equiv b \; mod \; (n)$, si y sólo si existe algún entero k que divida de forma exacta la diferencia $(a-b)$, o sea $a-b=k*n$.

De la definición se puede decir que por ejemplo $127 \equiv 1 \; mod(2)$. Con $k=64$, se satisface la definición ya que $128-1=64*2$, o sea el resto de la división $128/2=1$. Es importante señalar que $128 \equiv -1 \; mod(2)$, ya que $64-(-1)=65*2$ con $k=65$.

Las operaciones de suma y producto sobre la aritmética modular se definen como:

Suma: $a+b \equiv c \; mod(n)$ si y sólo si $a+b=c+kn$ con $k \in \mathbb{Z}$.

Producto: $ab \equiv c \; mod(n)$, si y sólo si $ab=c+kn$ con $k \in \mathbb{Z}$.

La operación suma cumple las propiedades asociativa y conmutativa y posee elementos neutro y simétrico. Por su parte el producto cumple con las propiedades asociativa y conmutativa y tiene elemento neutro, pero no necesariamente simétrico.

Otras operaciones importantes que son necesarias conocer son las operaciones de inverso multiplicativo y exponenciación modular.

Definición: Sea $a \in \mathbb{Z}_n$. El inverso multiplicativo de a módulo n es el entero $x \in \mathbb{Z}_n$ tal que $ax \equiv 1 \; mod(n)$. Si x existe es único y se dice que a es invertible. El inverso de a se denota a^{-1}.

Definición: Sea $a, b, c \in \mathbb{Z}$. La exponenciación modular de a en b modulo n se define como $a^b \equiv c \; mod \; n$.

A.1 Estructuras algebraicas [15]

Las estructuras algebraicas tienen mucha importancia en la definición de criptosistemas asimétricos. A través de su uso, es posible otorgar a un conjunto de elementos de algunas propiedades necesarias que garantizan el proceso de cifrado/descifrado y elementos matemáticos que sustentan su seguridad.

A.1.2 Grupos

Un grupo $(\mathbb{G},*)$ consiste de un conjunto $\mathbb{G}$ con la operación binaria *. $\mathbb{G}x\mathbb{G} \rightarrow \mathbb{G}$, satisfaciendo las siguientes propiedades.

- Cerradura: para todo $a, b \in \mathbb{G}$, el elemento $a * b$ es un elemento de $\mathbb{G}$.
- Asociatividad: Para todo $a, b, c \in \mathbb{G}$ se tiene

$$a * (b * c) = (a * b) * c$$

- Identidad: Existe un elemento $e \in \mathbb{G}$, tal que

$$a * e = e * a = a$$

  para toda $a \in \mathbb{G}$.
- Inversa: Para cada $a \in \mathbb{G}$, existe un elemento $a^{-1} \in \mathbb{G}$, llamado la inversa de a, tal que:

$$a * a^{-1} * a = e$$

  Si además cumple con la propiedad:
- Conmutatividad: Para todo $a, b \in G$ se tiene

$$a * b = b * a$$

  Se tiene que el grupo se denomina abeliano

Son dos las operaciones que se realizan en los grupos: la suma (+) o la multiplicación (·). En la primera operación, el grupo es llamado un grupo aditivo. El elemento identidad de la adición, es usualmente denotado por 0 y la inversa (aditiva) de a es denotado por $-a$. Para la segunda operación, el grupo es llamado un grupo multiplicativo. El elemento identidad de la multiplicación es usualmente denotado por 1 y la inversa (multiplicativa) de a es denotada por a^{-1}

Un grupo $\mathbb{G}$ se dice ser un grupo finito si el conjunto $\mathbb{G}$ tiene un número finito de elementos. En ese caso, el número finito de elementos es llamado el orden de G, denotado como $|\mathbb{G}|$.

Algún grupo de orden primo es cíclico. Sea $\mathbb{G}$ un grupo de orden n, y sea g algún elemento de $\mathbb{G}$. El conjunto $\langle g \rangle = \{a \in \mathbb{G}: a = g^i \; para \; 0 \leq i \leq n\}$, es llamado un subgrupo cíclico de $\mathbb{G}$ generado por g. El grupo $\mathbb{G}$ es llamado grupo cíclico si existe un elemento $g \in \mathbb{G}$, tal que $\mathbb{G} = \langle g \rangle$. Para este caso g se llama generador de $\mathbb{G}$.

A.2 Campos Finitos [6]

Un campo es un conjunto de elementos que satisfacen las condiciones de asociatividad, conmutatividad, identidad, inversa y propiedad distributiva. Además consisten de un conjunto $\mathbb{F}$ junto con dos operaciones. La suma (denotada por +) y la multiplicación (denotada por $\cdot$), que satisfacen las siguientes propiedades aritméticas:

1. $(\mathbb{F}, +)$ es un grupo abeliano con la identidad aditiva denotada por 0.
2. $(\mathbb{F}^* = \mathbb{F} - 0)$ es un grupo abeliano con la identidad multiplicativa denotada por 1.
3. Por la ley distributiva:
   $(a + b) \cdot c = a \cdot c + b \cdot c$
   Para todo $a, b, c \in \mathbb{F}$

Si el conjunto $\mathbb{F}$ es finito, es decir, si contiene un número finito de elementos, entonces se dice que es un campo finito o también llamado campo de Galois.

Un campo finito $\mathbb{F}$ no solo cuenta con las operaciones de la suma y multiplicación, sino también con sus inversas. La resta de los elementos del campo es definida en términos de la suma: para $a, b \in \mathbb{F}, a - b = a + (-b)$ donde $-b$ es el único elemento en $\mathbb{F}$ tal que $b + (-b) = 0$, por lo tanto $a - b$ se le llama el negativo de b. La división de elementos del campo, está dada en términos de la multiplicación: para $a, b \in \mathbb{F}$ con $b \neq 0$, se tiene $a/b = a \cdot b^{-1}$, donde b^{-1} es el único elemento en $\mathbb{F}$ tal que $b \cdot b^{-1} = 1$, así, a b^{-1}se le conoce como la inversa de b.

El orden de un campo finito, es el número de elementos que tiene el campo, más precisamente, el orden del campo es un número primo o la potencia de un primo (p^m).

De este modo, existe un campo finito $\mathbb{F}$ de orden q si y sólo si q es una potencia de un primo; $q = p^m$ donde p es un número primo llamado la característica de $\mathbb{F}$, y m es un entero positivo. Si $m = 1$, entonces $\mathbb{F}$ es llamado un campo primo. Si $m \geq 2$, entonces $\mathbb{F}$ es llamado campo de extensión.

Si q es la potencia de un primo, existe exactamente un campo de orden q, esto significa que cualquier par de campos finitos de orden q son estructuralmente iguales excepto por la forma usada para representar los elementos que puede ser diferente. Estos campos tienen la característica de ser isomorfos [22]y se denotan como $\mathbb{F}_p$ o también como $GF(p)$ (la notación GF son las siglas de Galois Field, en honor al matemático que fue el primero en estudiar los campos finitos).

Los elementos que no son iguales a cero en un campo $\mathbb{F}_p$, denotado como $\mathbb{F}_p^*$, forman un grupo cíclico bajo la operación de la multiplicación. Por ello existe un elemento $b \in \mathbb{F}_p^*$ llamado generador tal que:

$$\mathbb{F}_p^* = \{b^i : 0 \leq i \leq q-2\}$$

A.2.1 Campo primo $\mathbb{F}_p$ [6]

Sea p un número primo. El campo $\mathbb{F}_p$ es llamado campo primo, que está compuesto por el conjunto de enteros $\{0,1,2,\dots,p-1\}$ junto con la operación módulo p.

Las operaciones aritméticas definidas sobre estos elementos son:

1. Suma módulo p: Sea $a, b \in \mathbb{F}_p$ entonces $a + b = r$, donde r es el residuo de la división de $a + b$ entre p, $0 \leq r \leq p - 1$.
2. Multiplicación modulo p: Si $a, b \in \mathbb{F}_p$, entonces $a \cdot b = s$, donde s es el residuo de la división de $a \cdot b$ entre p y donde $0 \leq s \leq p - 1$.
3. Inversa: Sea a un elemento de $\mathbb{F}_p$, considerando que $a \neq 0$, la inversa de a módulo p, denotada a^{-1}, es el único entero $e \in \mathbb{F}_p$ tal que $a \cdot \mathrm{e} = 1$ ($a \cdot e$ es lleva a cabo módulo p).

[22] Isomorfismo: significa de "igual forma", se busca destacar la idea de que existen similitudes y correspondencias formales entre diversos sistemas.

A.2.2 Campos binarios $\mathbb{F}_{2^m}$ [6]

Los campos finitos de orden 2^m son llamados campos binarios o campos finitos característicos dos, que se representan como $\mathbb{F}_{2^m}$. Los campos $\mathbb{F}_{2^m}$ pueden ser vistos como espacios vectoriales de dimensión m sobre $\mathbb{F}_2$, es decir, existen m elementos $\{\alpha_0, \alpha_1, \dots, \alpha_{m-1}\}$en $\mathbb{F}_{2^m}$tales que cada elemento $\alpha \in \mathbb{F}_{2^m}$ puede ser escrito de manera única como:

$$\alpha = a_0\alpha_0 + a_1\alpha_1 + \cdots + a_{m-1}\alpha_{m-1}$$

Donde los coeficientes α_i puede ser algún elemento del conjunto $\{0,1\}$. Al conjunto $\alpha_0, \alpha_1, \dots, \alpha_{m-1}$ se le denomina base de $\mathbb{F}_{2^m}$ sobre $\mathbb{F}_2$. Dada una base tal, un elemento α del campo puede ser representado por la cadena de bits $(a_{0,}a_1, \dots, a_{m-1})$.

Existen diferentes bases de $\mathbb{F}_{2^m}$ sobre $\mathbb{F}_2$. Tres de las bases más comunes son: base polinomial, base normal y base dual. Cada una de estas presenta ventajas y desventajas cuando se realiza la implementación de la aritmética sobre los elementos del campo finito (multiplicación, división e inversión principalmente).

A.2.3 Subcampo

Un subcampo de un campo $\mathbb{F}$ es un subconjunto de $\mathbb{F}$ que contiene a {0,1} y que es cerrado para las dos operaciones de $\mathbb{F}$. Si $\mathbb{P}$ es un subcampo de $\mathbb{F}$ entonces se dice que $\mathbb{F}$ es una extensión de $\mathbb{P}$.

ANEXO B: CURVAS ELÍPTICAS [38] [6]

Una curva elíptica E definida sobre el campo $\mathbb{F}$ está dada por la siguiente ecuación:

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5$$

Ecuación 4

Donde $a_1, a_2, a_3, a_4, a_5 \in \mathbb{F}$ y $\Delta \leq 0$. A la Ecuación 4 se la conoce como Forma Normal de Weierstrass (FNW). Si $\Delta \leq 0$, la curva es no singular, eso significa que no hay puntos en los cuales la curva tenga dos o más líneas tangentes distintas. Donde Δ es el discriminante de E y está definido de la siguiente manera:

$$\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6$$
$$d_2 = a_1^2 + 4a_2$$
$$d_4 = 2a_4 + a_1a_3$$
$$d_6 = a_3^2 + 4a_5$$
$$d_8 = a_1^2a_5 + 4a_2a_5 - a_1a_3a_4 + a_2a_3^2 - a_4^2$$

Se dice que E está definida sobre $\mathbb{F}$, debido a que los elementos a_1, a_2, a_3, a_4, a_5 definidos en la Ecuación 4 son elementos de $\mathbb{F}$. Se puede escribir $E/\mathbb{F}$ para hacer énfasis que Eestá definida sobre $\mathbb{F}$, y por lo tanto, $\mathbb{F}$ es llamado campo subyacente.

Si L es algún campo de extensión de $\mathbb{F}$, el conjunto de puntos racionales sobre E son:

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\mathcal{O}\}.$$

La condición que $\Delta \neq 0$ asegura que la curva elíptica sea “suave”, es decir, no posee puntos con dos o más líneas tangentes distintas.

El punto $\mathcal{O}$ es llamado el punto al infinito de la curva elíptica, es considerado un punto racional L para todos los campos de extensión L de $\mathbb{F}$. Es decir, es considerado el único punto en la línea al infinito que satisface la forma proyectiva de la Ecuación 4

Teniendo como referencia la Ecuación 4, es posible encontrar formas particulares para la ecuación de curva dependiendo de la característica del campo subyacente.

Se tienen dos curvas elípticas llamadas E_1 y E_2 definidas sobre $\mathbb{F}$ y dadas por la FNW:

$$E_1: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5$$
$$E_2: y^2 + \overline{a_1}xy + \overline{a_3}y = x^3 + \overline{a_2}x^2 + \overline{a_4}x + \overline{a_5}$$

Se dice que son isomorfas sobre $\mathbb{F}$ si existen $u, r, s, t \in \mathbb{F}, u$ sea diferente de 0, de modo que el cambio de variables

$$(x, y) \rightarrow (u^2x + r, u^3y + u^2sx + t)$$

Transforma la ecuación E_1 a la ecuación E_2. La transformación es llamada cambio admisible de variables.

La Ecuación 4 puede ser simplificada, aplicando el cambio admisible de variables. Se puede considerar los casos de manera separada, donde el campo subyacente $\mathbb{F}$ tenga característica $car(\mathbb{F}) \neq \{2,3\}$ ó $car(\mathbb{F}) = \{2,3\}$.

- Si $car(\mathbb{F}) \neq \{2,3\}$, entonces el cambio admisible de variables

$$(x, y) \rightarrow \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1x}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{24}\right)$$

Transforma la curva E

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

Ecuación 5

Donde $a, b \in \mathbb{F}$. El discriminante de esa curva es $\Delta = -16(4a^3 + 27b^2)$.

- Si $car(\mathbb{F}) = 2$, entonces hay dos casos para considerar

I. Si $a_1 \neq 0$, el cambio admisible de variables

$$(x, y) \rightarrow \left(a_1^2x + \frac{a_3}{a_1}, a_1^3y + \frac{a_1^2a_4 + a_3}{a_1^3}\right)$$

Transformando la curva E

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

Ecuación 6

Donde $a, b \in \mathbb{F}$ y tiene como discriminante $\Delta = b$

II. Si $a_1 = 0$, entonces el cambio admisible de variables

$$(x, y) \rightarrow (x + a_2, y)$$

Transformando la curva E

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

Ecuación 7

Donde $a, b, c \in \mathbb{F}$ y tiene como discriminante $\Delta = c^4$

- Si $car(F) = 3$, entonces hay dos casos para considerar:

  I. Si $a_1^2 \neq -a_2$, entonces el cambio admisible de variables

$$(x, y) \rightarrow \left(x + \frac{d_4}{d_2}, y + a_1 x + a_1 \frac{d_4}{d_2} + a_3\right)$$

  Donde $d_2 = a_1^2 + a_2$ y d_4=$a_4 - a_1 a_3$ transformando la curva E

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

Ecuación 8

  Donde $a, b \in \mathbb{F}$ y tiene como discriminante $\Delta = -a^3 b$

  II. Si $a_1^2 = -a_2$, entonces el cambio admisible de variables

$$(x, y) \rightarrow (x, y + a_1 x + a_3)$$

  Transforma la curva E

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

Ecuación 9

  Donde $a, b, c \in \mathbb{F}$ y tiene como discriminante $\Delta = -a^3$

B.1 Estructura de grupo (caso general) [38] [6]

Sea E una ecuación FNW definida sobre el campo $\mathbb{F}$. Entonces $E \subset \mathbb{F}^2$ es el conjunto de puntos $P = (x, y)$ que satisfacen la ecuación junto con el punto al infinito ($\mathcal{O} = [0: 1: 0]$). Sea $L \subset \mathbb{F}^2$ una línea recta. Como la ecuación tiene grado 3, L intersecta a E exactamente en tres puntos, a saber P, Q y R.

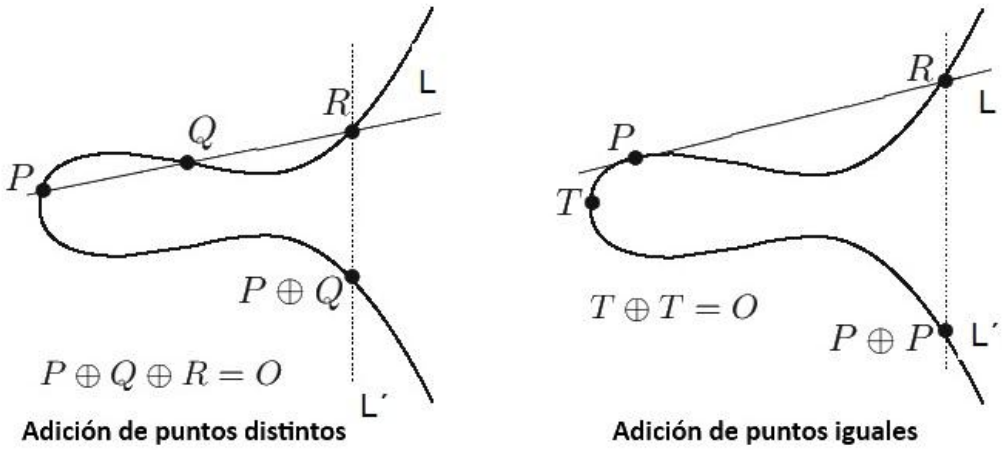

Figura 11: Adición de puntos [39]

Ley de composición:

Sean los puntos $P, Q \in E$, L la línea que pasa por P y Q (en caso en que P= Q, se forma una línea tangente sobre E) y R un tercer punto de intersección de

L con E. Sea L' la línea que conecta R y $\mathcal{O}$. Entonces L' intersecta a E en $R, \mathcal{O}$ y un tercer punto denotado como $P \oplus Q$. Esto se ilustra en la Figura 11.

La ley de composición tiene las siguientes propiedades

I. Si una línea L intersecta a E en los puntos (no necesariamente distintos) P, Q y R, entonces:

$$(P \oplus Q) \oplus R = \mathcal{O}$$

II. $(P \oplus \mathcal{O}) = P$ para todo P que pertenezca a E.

III. $(P \oplus Q) = Q \oplus P$ para todo $P, Q \in E$.

IV. Sea $P \in E$, existe un punto de E, denotado $\ominus P$, tal que:

$$P \oplus (\ominus P) = \mathcal{O}$$

V. Sea $P, Q, R \in E$, entonces:

$$(P \oplus Q) \oplus \mathrm{R} = \mathrm{P} \oplus (\mathrm{Q} \oplus \mathrm{R})$$

Con esto se concluye que la ley de composición hace de Eun grupo abeliano con el elemento identidad $\mathcal{O}$.

VI. Supóngase que E está definido sobre $\mathbb{F}$, entonces

$E(\mathbb{F}) = \{(x, y) \in \mathbb{F}^2 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\}$.

Es un subgrupo de E.

B.1.1 Curvas elípticas sobre campos $\mathbb{F}_p$ [6]

Sea $\mathbb{F}_p$ un campo finito de característica p diferente de 2 y 3, y sean $a, b \in \mathbb{F}_p$ tal que satisfacen la siguiente condición de equivalencia $4a^3 + 27b^2 \not\equiv 0 \ (mod\ p)$, entonces una curva elíptica, denotada $E_{(a,b)}$, sobre $\mathbb{F}_p$ se define por:

$$E_{(a,b)} = \left\{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : y^2 = x^3 + ax + b\right\} \cup \{\mathcal{O}\}$$

Ecuación 10

Los puntos de la curva forman un grupo abeliano bajo la operación aditiva.

Para poder sumar dos puntos de una curva elíptica en $\mathbb{F}_p$ $\left(E(\mathbb{F}_p)\right)$, existe una regla llamada de la tangente, que permite obtener un tercer punto sobre la curva.

Se puede comprender en forma intuitiva la operación de suma y duplicación de dos puntos desde una perspectiva geométrica.

Sean $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ dos puntos distintos sobre la curva elíptica E. Entonces el punto resultante de la suma de $P\ y\ Q$ denotado por $R = (x_3, y_3)$, es definida de la siguiente manera:

En primer lugar se traza una recta a través de los puntos $P\ y\ Q$, esta línea intersectará a la curva en un punto que se denotará como $-R$. El punto $-R$ es la reflexión en el eje de las abscisas al punto R. Esto queda ilustrado en la Figura 12

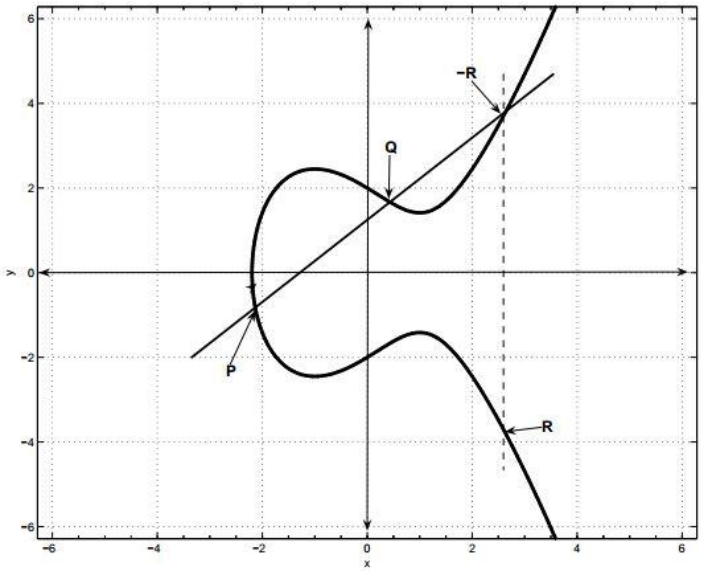

Figura 12: Perspectiva geométrica de la suma de dos puntos distintos

Otro aspecto a tener en cuenta dado el punto $P = (x_p, y_p)$, si se desea sumar los puntos P y $-P$, se debe trazar una línea recta que cruce a los dos puntos, la que no intersectará a la curva elíptica en un tercer punto. Por esta razón al grupo de puntos que conforman la curva elíptica se le agrega el punto al infinito $(\mathcal{O})$. Por definición se tiene que $P + (-P) = \mathcal{O}$, como se observa en la Figura 13.

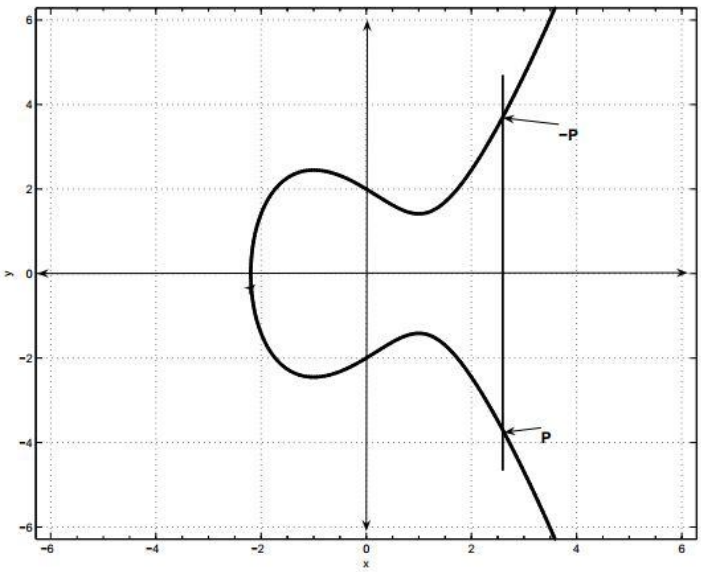

Figura 13: Perspectiva geométrica de la suma de dos puntos iguales

Por otro lado si se tiene el punto $P = (x_p, y_p)$, y se lo suma consigo mismo $(P + P)$, se dice que se duplica el punto P $(2P)$. El resultado de dicha operación es un punto sobre la curva denotado por $R = (x_r, y_r)$. Para esta operación se tiene que considerar dos casos:

I. Cuando $y_p \neq 0$: Primero se traza una línea tangente al punto P. Esta línea intersecta a la curva elíptica en otro punto, $-R$. Entonces $-R$ es la reflexión en el eje de las abscisas a R, como se ilustra en la Figura 14.

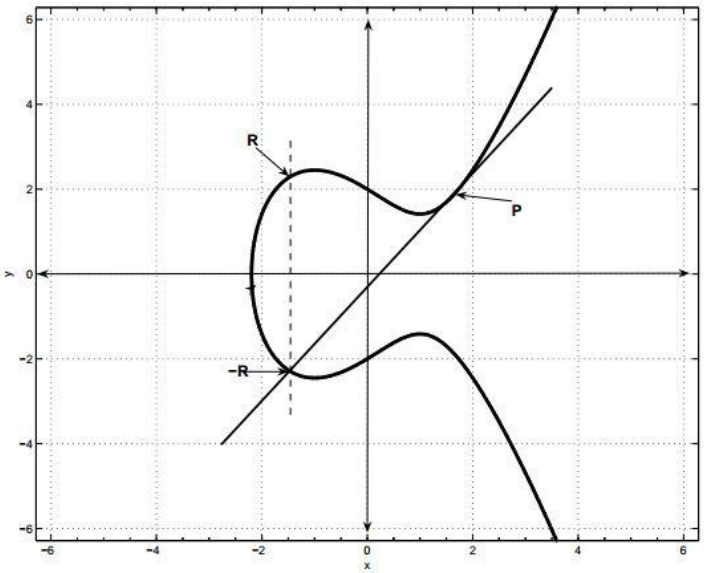

Figura 14: Perspectiva geométrica del doble de un punto: P+P=R, si $y_p \neq 0$

II. Cuando $y_p = 0$: La línea tangente a la curva elíptica en el punto P es vertical y no intersecta a la curva en otro punto. Por definición $2P = \mathcal{O}$, como se ilustra en la Figura 15

Teniendo una idea intuitiva de las distintas operaciones que pueden efectuarse con los puntos de una curva elíptica a partir de distintas perspectivas geométricas, pueden definirse las siguientes fórmulas algebraicas:

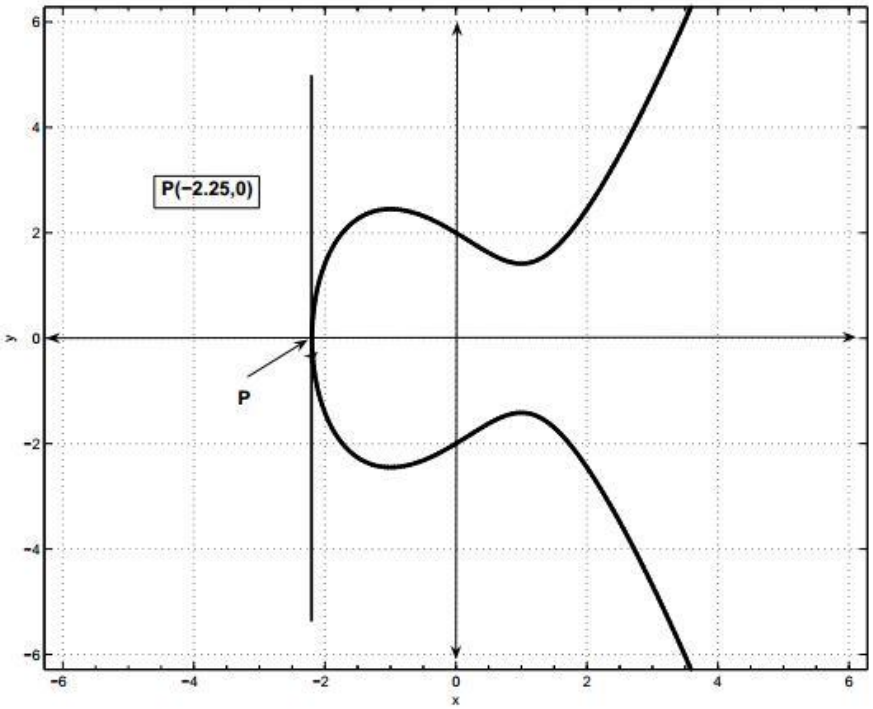

Figura 15: Perspectiva geométrica de la suma de dos punto iguales $\boldsymbol{P} + (-\boldsymbol{P}) = \boldsymbol{\mathcal{O}}$

I. Identidad: $P + \mathcal{O} = \mathcal{O} + \mathrm{P} = \mathrm{P}$, para todo $P \in E(\mathbb{F}_p)$.

II. Negativo: Si $P = (x, y) \in E(\mathbb{F}_p)$, entonces se tiene $(x, y) + (x, -y) = \mathcal{O}$. El punto $(x, -y)$ es denotado como $-\mathrm{P}$, y es llamado el negativo. Se tiene que tener en cuenta que $-\mathrm{P}$ es realmente un punto de la curva elíptica $E(\mathbb{F}_p)$.

III. Suma de un punto: Sea $P = (x_1, y_1) \in E(\mathbb{F}_p)$ y $Q = (x_2, y_2) \in E(\mathbb{F}_p)$, donde $P \neq \pm Q$. Entonces $P + Q = (x_3, y_3)$, donde

$$\lambda = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

IV. Doble de un punto: Sea $P = (x_1, y_1) \in E(\mathbb{F}_p)$ donde $P \neq -P$. Entonces $2P = (x_3, y_3)$, donde

$$\lambda = \frac{3x_1^2 + 9}{2y_1}$$

$$x_3 = \lambda^2 - 2x_1$$

$$y_3 = \lambda^2(x_1 - x_3) - y_1$$

B.1.2 Curvas elípticas sobre el campo $\mathbb{F}_{2^m}$ [6]

Se considerará el caso de las curvas elípticas definidas sobre los campos con característica 2, es decir sobre los campos $\mathbb{F}_{2^m}$. Una curva elíptica $E_{(a,b)}$sobre el campo $\mathbb{F}_{2^m}$ se encuentra definida por la siguiente ecuación:

$$E_{(a,b)} = \{(x.y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}: y^2 + xy = x^3 + ax^2 + b; b \neq 0\} \cup \{\mathcal{O}\}$$

En el apartado anterior referido a las curvas elípticas sobre el campo $\mathbb{F}_p$, se hizo mención a la regla de la tangente y la cuerda para sumar puntos sobre una curva, obteniéndose un tercer punto que pertenece a la curva. Así, junto una con la operación de la suma, el conjunto de los puntos que satisfacen la cuerda $E(\mathbb{F}_p)$ forman un grupo con el punto al infinito ($\mathcal{O}$), que sirve como su identidad.

Del mismo modo puede procederse con los puntos de una curva elíptica $E(\mathbb{F}_{2^m})$, sin embargo en el presente trabajo sólo se hará mención de las leyes de grupo:

V. Identidad: $P + \mathcal{O} = \mathcal{O} + \mathrm{P} = \mathrm{P}$, para todo $P \in E(\mathbb{F}_{2^m})$.

VI. Negativo: Si $P = (x, y) \in E(\mathbb{F}_{2^m})$, entonces se tiene $(x, y) + (x, x + y)$ es denotado como $-P$ y es llamado el negativo del punto $(x, -y)$ es denotado como $-\mathrm{P}$, y es llamado el negativo de P. Se tiene que tener en cuenta que $-\mathrm{P}$ es realmente un punto de la curva elíptica $E(\mathbb{F}_{2^m})$.

VII. Suma de un punto: $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ y $Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, donde $P \neq \pm Q$. Entonces $P + Q = (x_3, y_3)$, donde

$$\lambda = \frac{(y_1 + y_2)}{(x_1 + x_2)}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_1$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

VIII. Doble de un punto: Sea $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ donde $P \neq -P$. Entonces $2P = (x_3, y_3)$, donde

$$\lambda = \frac{x_1 + y_1}{x_1}$$

$$x_3 = \lambda^2 + \lambda + a$$

$$= x_1^2 + \frac{b}{x_1^2}$$

$$y_3 = x_1^2 + \lambda x_3 + x_3$$

Se debe tener en cuenta que λ representa la pendiente de la recta.

B.2 Algunas propiedades básicas [38] [6] [11]

Orden de grupo

Sea E una curva elíptica definida sobre $\mathbb{F}_q$. El número de puntos en $E(\mathbb{F}_q)$, incluyendo el punto al infinito $\mathcal{O}$, denotado por $\#E(\mathbb{F}_q)$, es llamado el orden de una curva elíptica.

El teorema de Hasse, provee una cota para $\#E(\mathbb{F}_q)$:

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q}$$

Al intervalo $\left[q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}\right]$ se le llama el intervalo de Hasse. Una alternativa para la formulación del teorema de Hasse es el siguiente: si E está definida sobre $\mathbb{F}_q$, entonces $\#E(\mathbb{F}_q) = q + 1 - t$ donde $|t| \leq 2\sqrt{q}$; t es llamado la traza de la curva elíptica (E) sobre $\mathbb{F}_q$. Ya que $2\sqrt{q}$ es relativamente pequeño para q se tiene que $\#E(\mathbb{F}_q) \approx q$.

Sea E una curva elíptica y P un punto sobre la curva. Si k es un entero positivo, definimos la potencia k de P, denotada como kP, como:

$$kP = P + P + \cdots + P$$

El orden de un punto (P), que se denota como $ord(P)$, se define como el menor entero k tal que $kP = \mathcal{O}$ (el punto al infinito)

Punto generador de curva elíptica

Una curva elíptica E definida sobre el campo $\mathbb{F}_q$ es isomorfa a $\mathbb{Z}_{n1} \times \mathbb{Z}_{n2}$ (se usa $\mathbb{Z}_n$ para denotar un grupo cíclico de orden n); donde n_1y n_2 son enteros positivos tales que n_2 divide a n_1 como a $q - 1$.

Se debe tener en cuenta que $\#E(\mathbb{F}_q) = n_1.n_2$. Si n_2=1, entonces $E(\mathbb{F}_q)$ es un grupo cíclico. Si n_2 es un entero pequeño $(n = 2, 3\ o\ 4)$, se dice que es casi cíclico. Si se tiene un grupo cíclico, $(\mathbb{F}_q)$ es isomorfa a $\mathbb{Z}_{n1}$y existe un punto que pertenece a $E(\mathbb{F}_q)$, que cumple:

$$E\left(\mathbb{F}_q\right) = \{kP: 0 \leq k \leq n_1 - 1\}$$

Dicho punto se llama generador de la curva elíptica sobre el campo finito $\mathbb{F}_q$.

ANEXO C: PARÁMETROS RECOMENDADOS PARA LA CURVA SECP521R1

p = 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

= $2^{521} - 1$

La curva $E: y^2 = x^3 + ax + b$ sobre $\mathbb{F}_p$ es definido por:

a = 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFC

b = 0051 953EB961 8E1C9A1F 929A21A0 B68540EE A2DA725B 99B315F3 B8B48991 8EF109E1 56193951 EC7E937B 1652C0BD 3BB1BF07 3573DF88 3D2C34F1 EF451FD4 6B503F00

E fue elegido en forma aleatoria según las especificaciones dadas en el estándar ANSI X9.62 de la semilla:

S = D09E8800 291CB853 96CC6717 393284AA A0DA64BA

El punto base G en forma comprimida es:

G = 0200C6 858E06B7 0404E9CD 9E3ECB66 2395B442 9C648139 053FB521 F828AF60 6B4D3DBA A14B5E77 EFE75928 FE1DC127 A2FFA8DE 3348B3C1 856A429B F97E7E31 C2E5BD66

Y en forma descomprimida es:

G = 04 00C6858E 06B70404 E9CD9E3E CB662395 B4429C64 8139053F B521F828 AF606B4D 3DBAA14B 5E77EFE7 5928FE1D C127A2FF A8DE3348 B3C1856A 429BF97E 7E31C2E5 BD660118 39296A78 9A3BC004 5C8A5FB4 2C7D1BD9 98F54449 579B4468 17AFBD17 273E662C 97EE7299 5EF42640 C550B901 3FAD0761 353C7086 A272C240 88BE9476 9FD16650

Finalmente el orden n de G y el cofactor son:

n = 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFA 51868783 BF2F966B 7FCC0148 F709A5D0 3BB5C9B8 899C47AE BB6FB71E 91386409

h = 01

ANEXO D: CODIGOS DE PORGRAMAS DE PRUEBA

D.1 Librería Bouncy Castle

```
1 package Ejemplos;
 2
 3 import java.security.KeyPair;
 4 import java.security.KeyPairGenerator;
 5 import java.security.MessageDigest;
 6 import java.security.SecureRandom;
 7 import java.security.Security;
 8 import java.security.spec.ECGenParameterSpec;
 9
10 import javax.crypto.KeyAgreement;
11 import org.bouncycastle.jce.provider.BouncyCastleProvider;
12
13 public class ECCBouncy
14 {
15     public static void main(String[] args) throws Exception
16     {
17
18         long startTime = System.nanoTime();
19
20         Security.addProvider(new BouncyCastleProvider());
21         KeyPairGenerator keyGen = KeyPairGenerator.getInstance("ECDH", "BC");
22         ECGenParameterSpec ecSpec = new ECGenParameterSpec("secp521r1");
23         keyGen.initialize(ecSpec, new SecureRandom());
24
25         // Setup
26         KeyAgreement aKeyAgree = KeyAgreement.getInstance("ECDH", "BC");
27         KeyPair      aPair = keyGen.generateKeyPair();
28         KeyAgreement bKeyAgree = KeyAgreement.getInstance("ECDH", "BC");
29         KeyPair      bPair = keyGen.generateKeyPair();
30
31         // Acuerdo de dos partes
32         aKeyAgree.init(aPair.getPrivate());
33         bKeyAgree.init(bPair.getPrivate());
34
35         aKeyAgree.doPhase(bPair.getPublic(), true);
36         bKeyAgree.doPhase(aPair.getPublic(), true);
37
38         // Generar las claves
39         MessageDigest    hash = MessageDigest.getInstance("SHA1", "BC");
40         byte[] aShared = hash.digest(aKeyAgree.generateSecret());
41         byte[] bShared = hash.digest(bKeyAgree.generateSecret());
```

```
42
43        System.out.println("Clave a:" + aShared);
44        System.out.println("Clave b:" + bShared);
45        long endTime = System.nanoTime();
46
47        System.out.println("Total time: "+(endTime - startTime));
48
49    }
50 }
```

D.2 Librería Flexi Provider

```
1 package Ejemplos;
 2
 3 import java.security.KeyPair;
 4 import java.security.KeyPairGenerator;
 5 import java.security.MessageDigest;
 6 import java.security.SecureRandom;
 7 import java.security.Security;
 8 import java.security.spec.ECGenParameterSpec;
 9
10 import javax.crypto.KeyAgreement;
11 import org.bouncycastle.jce.provider.BouncyCastleProvider;
12
13 public class ECCBouncy
14 {
15    public static void main(String[] args) throws Exception
16    {
17
18        long startTime = System.nanoTime();
19
20        Security.addProvider(new BouncyCastleProvider());
21
22        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("ECDH", "BC");
23
24        ECGenParameterSpec ecSpec = new ECGenParameterSpec("secp521r1");
25
26        keyGen.initialize(ecSpec, new SecureRandom());
27
28
29        // Setup
30        KeyAgreement aKeyAgree = KeyAgreement.getInstance("ECDH", "BC");
31        KeyPair      aPair = keyGen.generateKeyPair();
32        KeyAgreement bKeyAgree = KeyAgreement.getInstance("ECDH", "BC");
33        KeyPair      bPair = keyGen.generateKeyPair();
34
35
36        // Acuerdo de dos partes
```

```
37        aKeyAgree.init(aPair.getPrivate());
38        bKeyAgree.init(bPair.getPrivate());
39
40        aKeyAgree.doPhase(bPair.getPublic(), true);
41        bKeyAgree.doPhase(aPair.getPublic(), true);
42
43        // Generar las claves
44        MessageDigest    hash = MessageDigest.getInstance("SHA1", "BC");
45        byte[] aShared = hash.digest(aKeyAgree.generateSecret());
46        byte[] bShared = hash.digest(bKeyAgree.generateSecret());
47
48        System.out.println("Clave a:" + aShared);
49        System.out.println("Clave b:" + bShared);
50        long endTime = System.nanoTime();
51
52        System.out.println("Total time: "+(endTime - startTime));
53
54    }
55 }
```

D.3 Librería SunEC

```
1 package Ejemplos;
 2
 3 import java.security.KeyPair;
 4 import java.security.KeyPairGenerator;
 5 import java.security.MessageDigest;
 6 import java.security.SecureRandom;
 7 import java.security.Security;
 8 import java.security.spec.ECGenParameterSpec;
 9 import javax.crypto.KeyAgreement;
10
11 public class SunEC {
12    public static void main(String[] args) throws Exception
13    {
14        long startTime = System.nanoTime();
15
16        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC", "SunEC");
17        ECGenParameterSpec ecSpec = new ECGenParameterSpec("secp521r1");
18        keyGen.initialize(ecSpec, new SecureRandom());
19
20        // Setup
21        KeyAgreement aKeyAgree = KeyAgreement.getInstance("ECDH", "SunEC");
22        KeyPair      aPair = keyGen.generateKeyPair();
23        KeyAgreement bKeyAgree = KeyAgreement.getInstance("ECDH", "SunEC");
24        KeyPair      bPair = keyGen.generateKeyPair();
25
26        // Acuerdo de dos partes
```

```
27        aKeyAgree.init(aPair.getPrivate());
28        bKeyAgree.init(bPair.getPrivate());
29
30        aKeyAgree.doPhase(bPair.getPublic(), true);
31        bKeyAgree.doPhase(aPair.getPublic(), true);
32
33        // Generar las claves
34        MessageDigest    hash = MessageDigest.getInstance("SHA1");
35        byte[] aShared = hash.digest(aKeyAgree.generateSecret());
36        byte[] bShared = hash.digest(bKeyAgree.generateSecret());
37
38        System.out.println("Clave a:" + aShared);
39        System.out.println("Clave b:" + bShared);
40        long endTime = System.nanoTime();
41
42        System.out.println("Total time: "+(endTime - startTime));
43
44    }
45 }
46
```