ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS E INFORMÁTICA

Elaborado por:

Wendy Ayala

DIRECTOR: ING. GALÁRRAGA, FERNANDO

CODIRECTOR: ING. CAIZAGUANO, CARLOS

Agenda

TEMA

“Estudio comparativo de algoritmos de firma digital utilizando criptografía asimétrica RSA y criptografía basada en curvas elípticas”

Planteamiento del Problema

Información Sensible

- **Confidencialidad**
- **Identidad e Integridad**

RSA

Factorización de Números enteros primos

Computadora cuántica

ECC

Ampliamente estudiados

OBJETIVO GENERAL

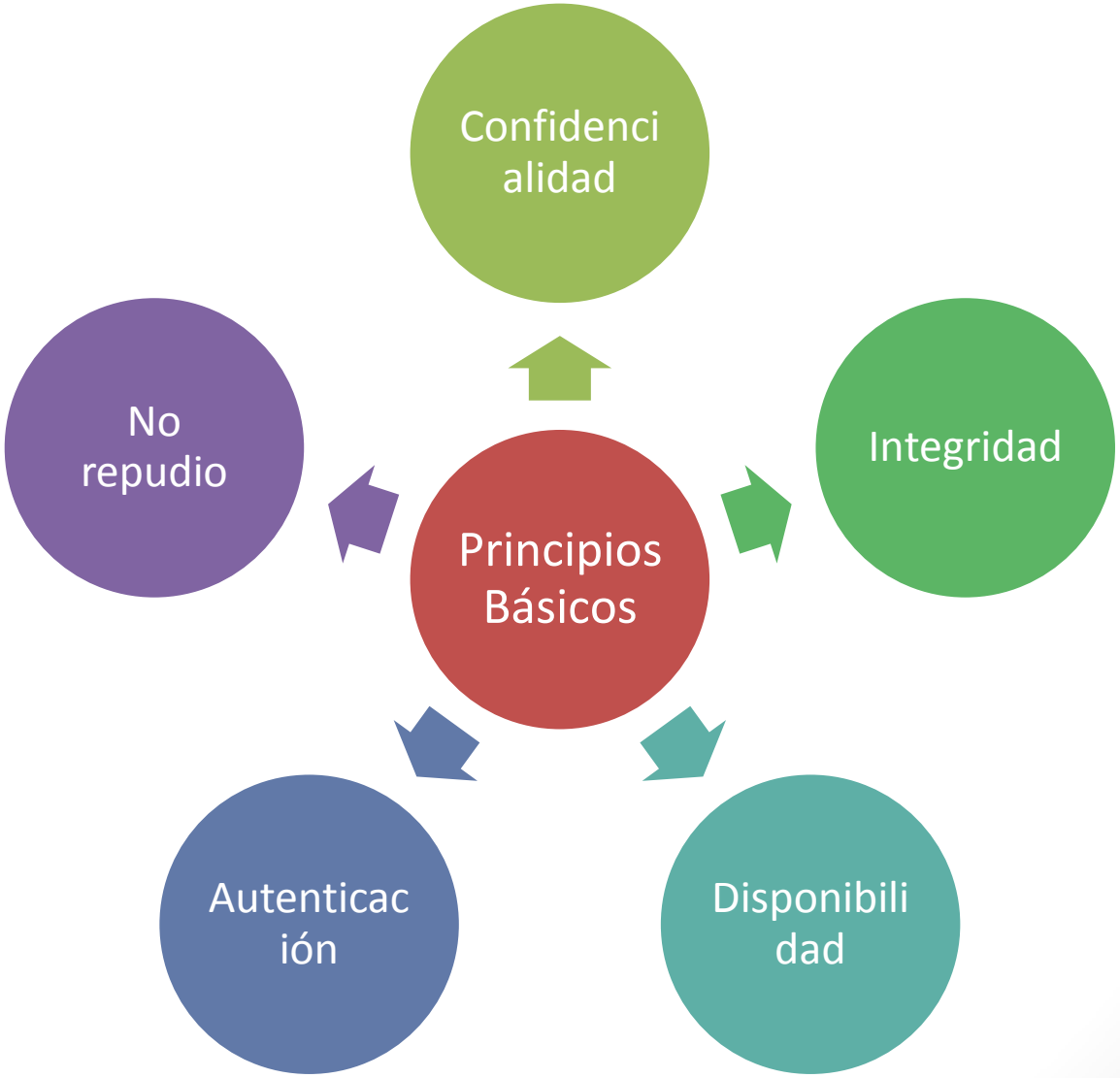
- Realizar el estudio comparativo de algoritmos de firma digital utilizando criptografía asimétrica RSA y criptografía basada en curvas elípticas.

Objetivos Específicos

- Analizar la teoría de la complejidad de los criptosistemas asimétricos y de curvas elípticas.
- Describir las propiedades necesarias para el uso de funciones hash que garanticen una criptografía útil.
- Formalizar los fundamentos matemáticos soportados en los algoritmos de firma digital RSA y ECDSA.
- Implementar los algoritmos que permitan la generación de firmas digitales.

MARCO TEÓRICO

Criptografía Teórica - Principios



Criptografía Teórica – Reglas

Reglas de Kerckhoff.	No se debe recuperar el texto original mediante el criptograma.
	Los sistemas criptográficos se incluirán información de tipo: Pública – algoritmos Privada – claves
	La forma de establecer la clave debe ser fácil de recordar y modificar.
	El criptograma debe poder enviarse a través de los diferentes canales de comunicación.
	El costo de descifrado debe ser equivalente al costo de su contenido.

Criptografía Teórica - Tipos de amenaza

Pasiva

- Robo
- Acceso no autorizado
- Intercepción
  - Confidencialidad

Activa

- Modificación
- Interrupción
- Falsificación
  - Integridad
  - Autenticidad

Criptografía Teórica - Tipos de ataque

- Criptograma

Con texto cifrado

- Tema del mensaje

Con texto original

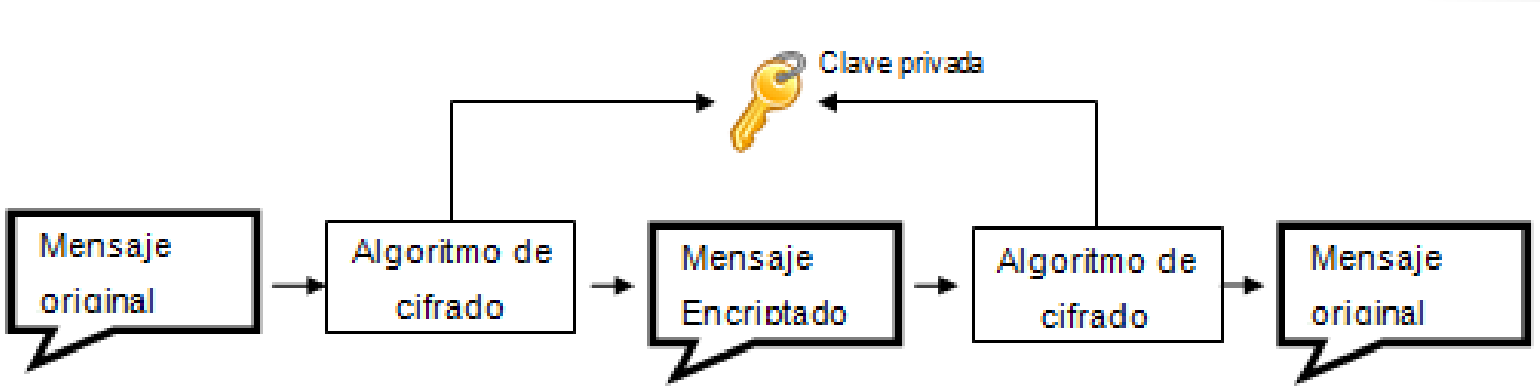
- Criptograma
- Mensaje cifrado escogido

Texto original escogido

- Mensaje original
- Mensajes cifrados escogidos.

Cifrado escogido

Criptografía Simétrica

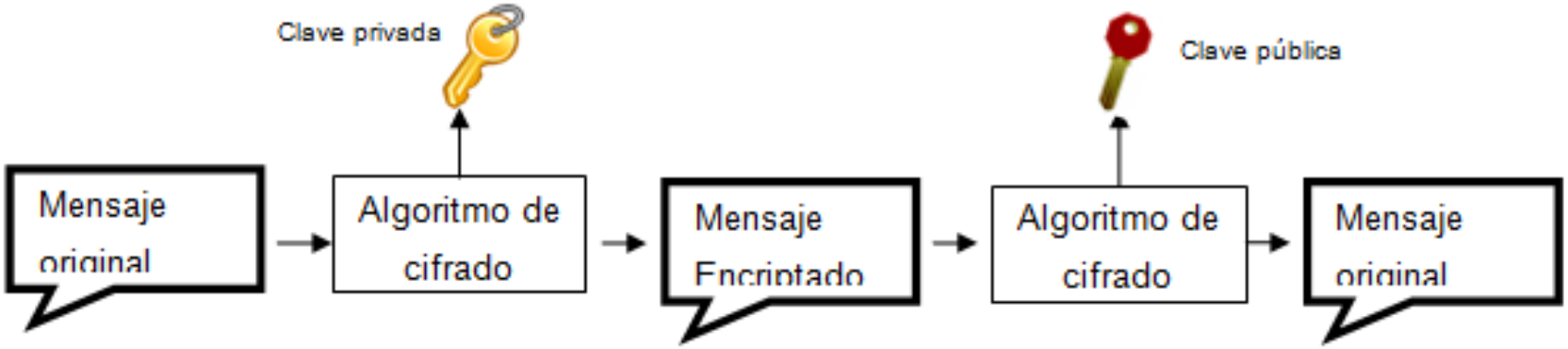


PRINCIPIOS BÁSICOS:

- Si se conoce el criptograma a través de éste no se pueda obtener ni el mensaje ni la clave.

- Si se conoce el mensaje original y el mensaje cifrado debe implicar un gasto mayor tiempo o dinero descifrar la clave que la información que será obtenida.

Criptografía Asimétrica



PRINCIPIOS BÁSICOS:

- La clave de descifrado no puede ser calculada a partir de la clave de cifrado.
- Si una de las claves es utilizada para cifrar el mensaje, entonces la otra es utilizada para descifrar el mensaje

Criptografía Asimétrica - Bases

Según Diffie y Hellman	Cualquier usuario puede calcular sus propias claves pública y privada.
	El emisor puede cifrar su mensaje con la clave pública del receptor.
	El receptor pude descifrar el criptograma con la clave privada.
	El criptoanalista que intente averiguar la clave privada mediante la pública, no podrá resolverlo.
	El criptoanalista que intente descifrar un criptograma teniendo la clave pública no podrá resolverlo.

Criptografía de Curvas Elípticas

- Los algoritmos de curvas elípticas pueden basarse en el logaritmo discreto, teniendo en cuenta que permite usar claves más pequeñas que se relacionan directamente con el almacenamiento, debido a que permiten la utilización de menos memoria y hardware de menor capacidad de procesamiento.

Criptografía de Curvas Elípticas

- El punto R se utiliza como clave pública la cual proviene de la suma de los puntos P y Q, los cuales corresponden a la clave de tipo privada.

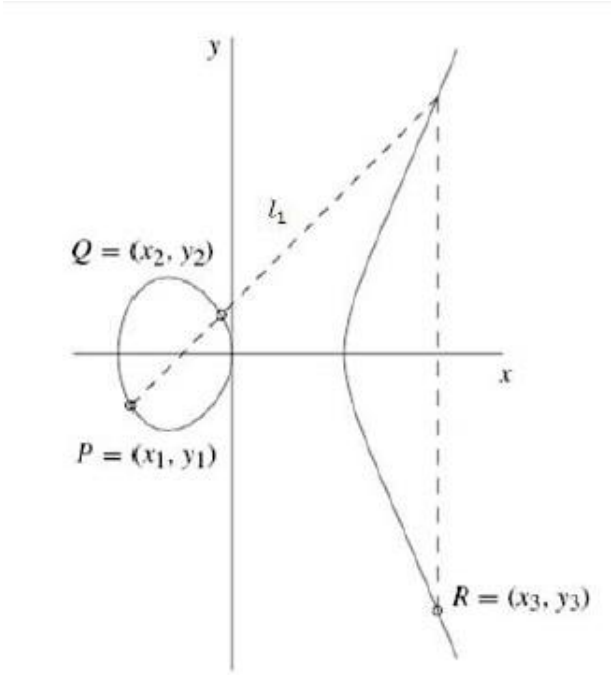


Figura 1 Representación geométrica de la suma de puntos de Curva Elíptica

Fuente: (Trujillo, 2011)

Criptografía de Curvas Elípticas - Aplicaciones

Test de primalidad

- Permite decidir si un número natural n es primo o compuesto
- Test de divisiones sucesivas.

Algoritmos de factorización

- Es un algoritmo de tiempo de ejecución sub exponencial para la factorización de enteros que emplea curvas elípticas

Firma Digital

La firma digital es un documento que garantiza la legitimidad e integridad de la información contenida en un **MENSAJE** mediante un **CIFRADO**.

Además permite verificar la autenticidad al comparar el MENSAJE con la llave privada del firmante.

Funciones Hash

- El objetivo de las funciones hash es comprobar la integridad de la información.

PROPIEDADES:

Difícil de hallar un mensaje m, tal que hash(m) =h

Difícil de hallar dos mensajes m1 y m2, tal que hash(m1)= hash(m2).

Funciones Hash – Tipos de funciones

SHA-1

- El proceso de SHA se basa en el proceso de MD5
- Su cálculo produce una función hash de 160 bits

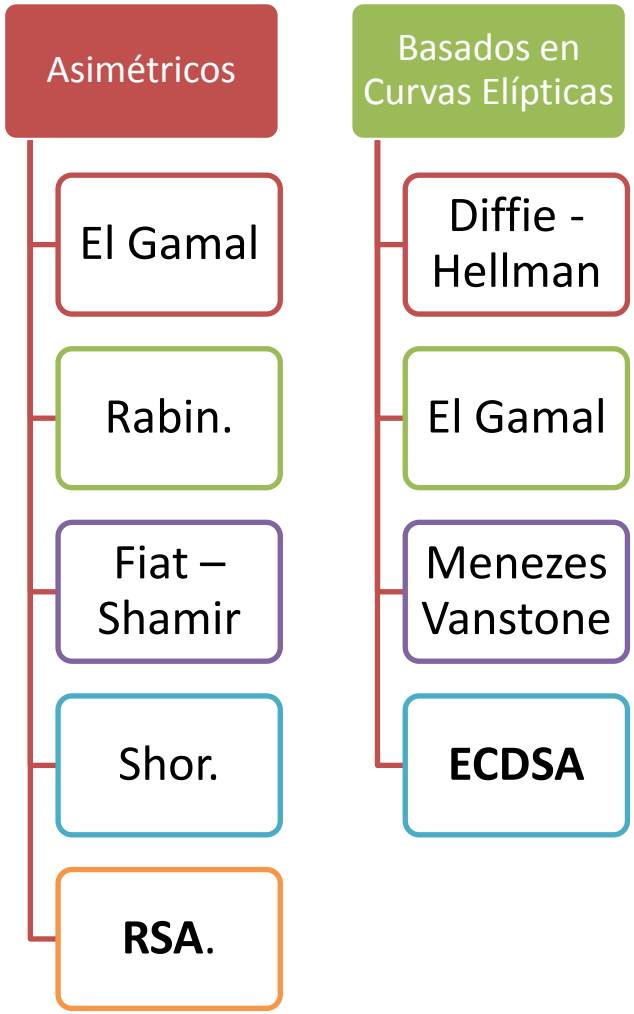
MD5

- Utiliza una constante numérica distinta para cada palabra del mensaje en cada pase
- Origina una función hash de 128 bits

RIPEMD-160

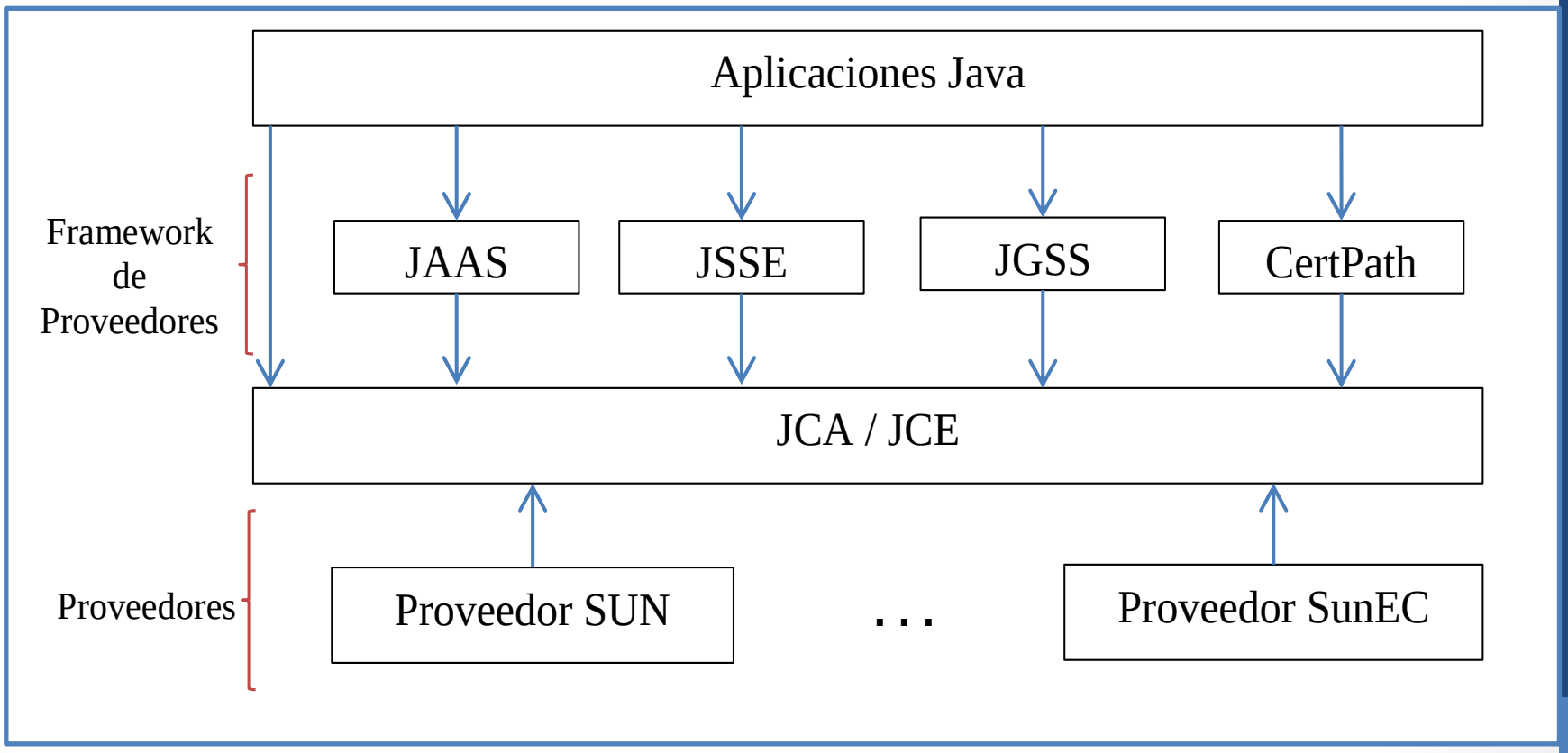
- Maneja claves más robustas, normalmente de 160 bits, aunque existen versiones de 128 y están en proceso nuevas de 256 y 320 bits.

FUNDAMENTACIÓN MATEMÁTICA DE ALGORITMOS DE FIRMA DIGITAL



IMPLEMENTACIÓN DE ALGORITMOS DE FIRMA DIGITAL

Esquema de seguridad de Java



Pruebas y Evaluación complementaria

Parámetros de dominio de curva elíptica	Fuerza / Nivel de Seguridad (56-256)	ECC (bits)	RSA (bits)
secp192k1/ secp192r1	96	192	1536
secp224k1/ secp224r1	112	224	2048
secp256k1/ secp256r1	128	256	3072
secp384r1	192	384	8192
secp521r1	256	521	15360

Propiedades de parámetros de dominio de curva elíptica recomendados sobre Zp

Generación de Firma RSA

```
KeyPairGenerator genClave = KeyPairGenerator.getInstance("RSA");

//genera o calcula números aleatorios
SecureRandom aleatorio = SecureRandom.getInstance("SHA1PRNG", "SUN");
genClave.initialize(tamañoClave, aleatorio);

KeyPair pardeClaves = genClave.generateKeyPair();
PrivateKey clavePrivada = pardeClaves.getPrivate();
PublicKey clavePublica = pardeClaves.getPublic();

/*
 * Implementación del motor utilizado para firmar datos
 * getInstance("MD5withRSA","SunEC") solicita a la clase java.security.Security que le proporcione el objeto solicitado
 * con proveedor MD5 con RSA el proveedor es buscaso el listado java.security
 */
Signature firmaRSA = Signature.getInstance("SHA1withRSA");
firmaRSA.initSign(clavePrivada);

FileInputStream archivo = new FileInputStream(mensaje);
BufferedInputStream buferEntrada = new BufferedInputStream(
        archivo);
byte[] buffer = new byte[1024];
int longitud;
while (buferEntrada.available() != 0) {
    longitud = buferEntrada.read(buffer);
    firmaRSA.update(buffer, 0, longitud);
}

buferEntrada.close();
byte[] firmaReal = firmaRSA.sign();

/* Guardando los datos firmados en un archivo */
FileOutputStream archivoFirma = new FileOutputStream("firmaRSA.txt");
archivoFirma.write(firmaReal);
archivoFirma.close();
```

Verificación de Firma RSA

```
FileInputStream clavepublica = new FileInputStream(files[0]);
byte[] clave = new byte[clavepublica.available()];
clavepublica.read(clave);
clavepublica.close();

X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(clave);
/*
 * Implementacion de un motor KeyFactory
 * clase que convierte llaves criptográficas de tipo key
 * en la llave del tipo especificado.
 */

KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PublicKey clavePublica = keyFactory.generatePublic(pubKeySpec);

FileInputStream archivoFirmado = new FileInputStream(files[1]);
byte[] firmaVerificada = new byte[archivoFirmado.available()];
archivoFirmado.read(firmaVerificada);
archivoFirmado.close();
Signature firma = Signature.getInstance("SHA1withRSA");

firma.initVerify(clavePublica);

FileInputStream mensaje = new FileInputStream(files[2]);
BufferedInputStream buferEntrada = new BufferedInputStream(
        mensaje);

byte[] buffer = new byte[1024];
int longitud;
while (buferEntrada.available() != 0) {
    longitud = buferEntrada.read(buffer);
    firma.update(buffer, 0, longitud);
}
buferEntrada.close();
boolean verifica = firma.verify(firmaVerificada);

return ("Verificación de la firma RSA" + verifica);
```

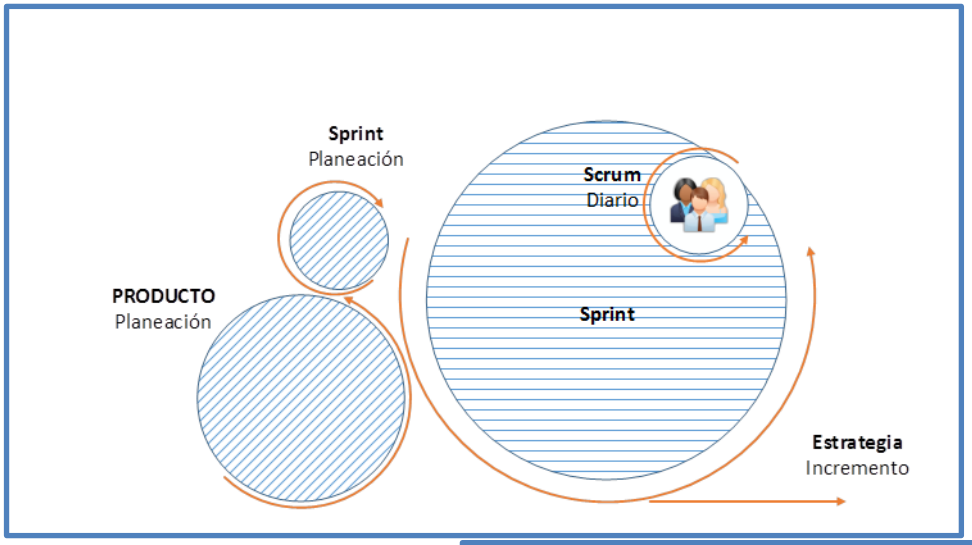
Generación de Firma ECDSA

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC", "SunEC");

ECGenParameterSpec ecsp = new ECGenParameterSpec(standar);
kpg.initialize(ecsp);

KeyPair kp = kpg.genKeyPair();
PrivateKey clavePrivada = kp.getPrivate();
PublicKey clavePublica = kp.getPublic();

System.out.println(clavePrivada.toString());
System.out.println(clavePublica.toString());

Signature firmaECDSA = Signature.getInstance("SHA1withECDSA", "SunEC");
firmaECDSA.initSign(clavePrivada);

FileInputStream archivo = new FileInputStream(mensaje);
BufferedInputStream buferEntrada = new BufferedInputStream(archivo);
byte[] buffer = new byte[1024];
int longitud;

while (buferEntrada.available() != 0) {
    longitud = buferEntrada.read(buffer);
    firmaECDSA.update(buffer, 0, longitud);
}
buferEntrada.close();

byte[] firmaReal = firmaECDSA.sign();

FileOutputStream archivoFirma = new FileOutputStream( "firmaECDSA.txt");
archivoFirma.write(firmaReal);
archivoFirma.close();
tamanoFirma = firmaReal.length;
```

Verificación de Firma ECDSA

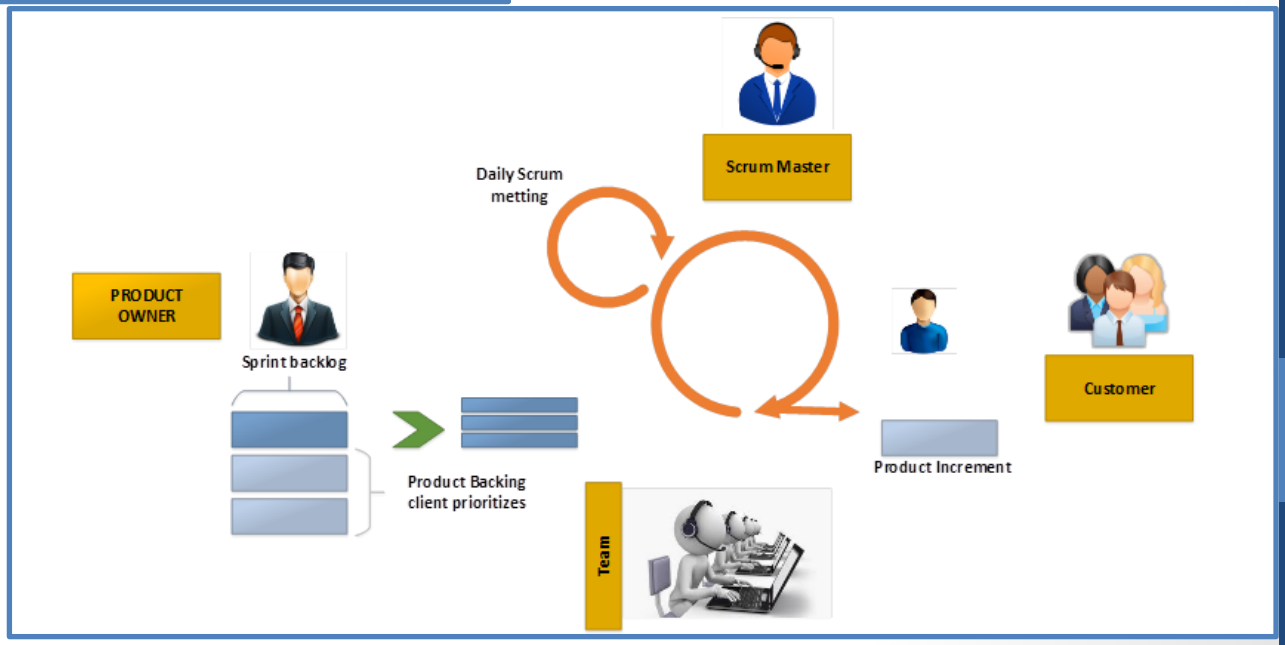
```
// para manejo de la clave publica
FileInputStream clavepublica = new FileInputStream(files[0]);
byte[] clave = new byte[clavepublica.available()];
clavepublica.read(clave);
clavepublica.close();

X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(clave);
KeyFactory keyFactory = KeyFactory.getInstance("EC", "SunEC");
PublicKey clavePublica = keyFactory.generatePublic(pubKeySpec);

// para manejo de la firma ya generada
FileInputStream archivoFirmado = new FileInputStream(files[1]);
byte[] firmaVerificada = new byte[archivoFirmado.available()];
archivoFirmado.read(firmaVerificada);
archivoFirmado.close();
Signature firma = Signature.getInstance("SHA1withECDSA",   "SunEC");
firma.initVerify(clavePublica);

// para manejo de los datos
FileInputStream mensaje = new FileInputStream(files[2]);
BufferedInputStream buferEntrada = new BufferedInputStream(
        mensaje);
byte[] buffer = new byte[1024];
int longitud;
while (buferEntrada.available() != 0) {
    longitud = buferEntrada.read(buffer);
    firma.update(buffer, 0, longitud);
}
buferEntrada.close();
boolean verifica = firma.verify(firmaVerificada);

return ("Verificación de la firma ECC " + verifica);
```

Aplicativo- Ingeniería de Software



Proceso Scrum

Productos y personal involucrado en metodología Scrum



Requisitos funcionales

RF	Nombre	Proceso	Descripción
RF01	Capturar y procesar información	Tratamiento	El aplicativo toma la información de entrada y genera o verifica la firma digital según haya sido especificado en los parámetros de ingreso, las cuales son: Acción (Generar o verificar), Algoritmo (RSA, ECDSA), tamaño de firma, mensaje, clave pública(si se desea verificar), archivo firmado (si se desea verificar).
RF02	Generar Archivos	Tratamiento	Si en el proceso de captura de información se ha seleccionado generación de firma, se generarán los archivos de firma y clave pública, que podrán ser visualizados en el lugar de instalación del aplicativo.
RF03	Almacenar información	Almacenamiento	La información de ingreso, así como tiempos de respuesta y datos relacionados son almacenados en la base de datos.
RF04	Generar reporte de actividad en la aplicación	Presentación	Obtiene un reporte con la especificación de la información procesada y previamente almacenada en la base de datos.

Requisitos no funcionales

Requerimiento No Funcional	Uso en el Aplicativo
Fiabilidad	La información almacenada deberá estar conforme lo procesado para garantizar que las consultas realizadas arroguen datos que no alteren la comparación de los algoritmos
Usabilidad	Se garantiza que el aplicativo es fácilmente manejable y entendible según el propósito del mismo
Eficiencia	La aplicación garantiza que el costo de desarrollo según los recursos utilizados esté de acuerdo a las operaciones a realizar.
Mantenibilidad	El aplicativo está desarrollado únicamente con fines del estudio comparativo entre los algortimos RSA y ECDSA por lo que no es necesario el mantenimiento del mismo.
Portabilidad	El aplicativo por tener pocos procesos específicos debido a su enfoque y al ser desarrollado en la plataforma java facilita la portabilidad del mismo.
*Seguridad	No se considera necesario, debido a que en el aplicativo existe un único usuario con acceso a todos los procesos.
*Disponibilidad	No se considera necesario, por tratarse de una aplicación que va a ser accedida únicamente cuando se desee realizar las comparaciones pertinentes.

Diseño de la Base de datos

Diagrama de casos de uso

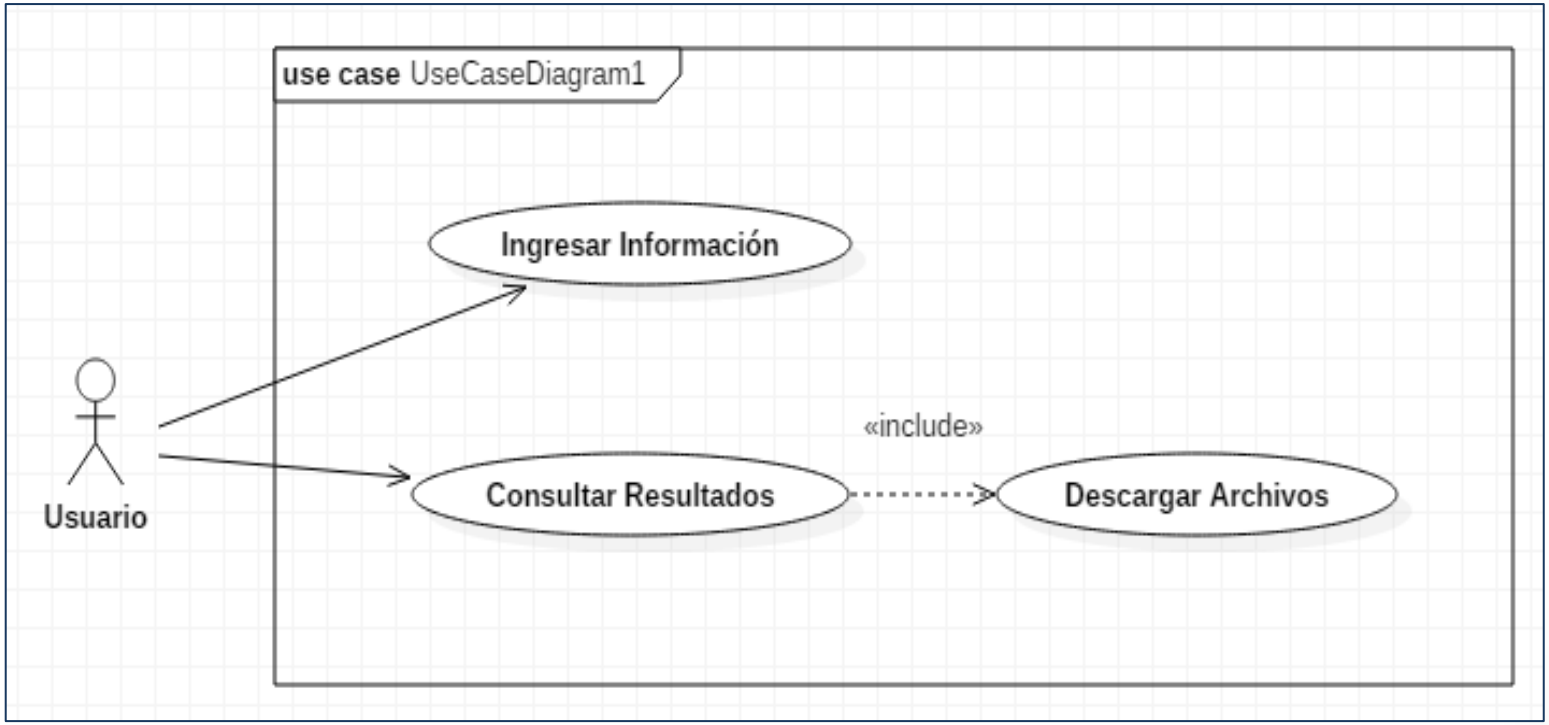


Diagrama de Secuencia

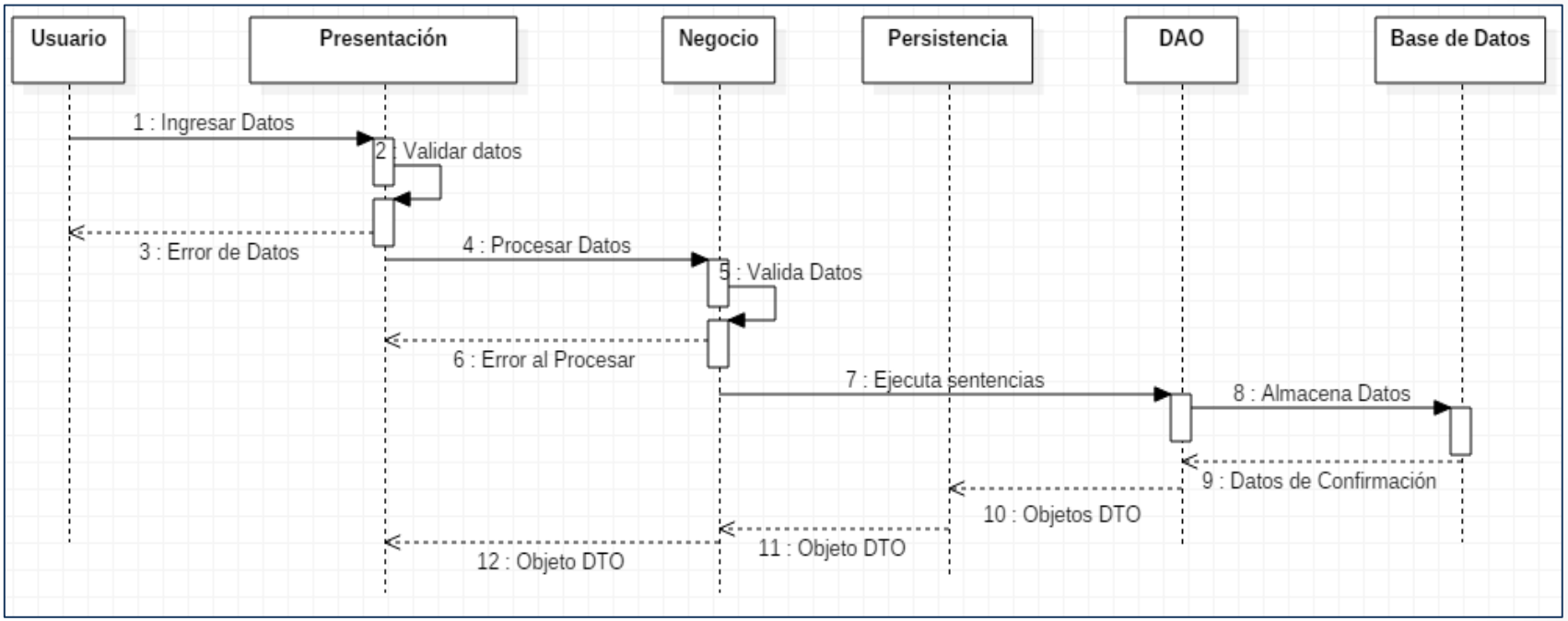


Diagrama de Clases

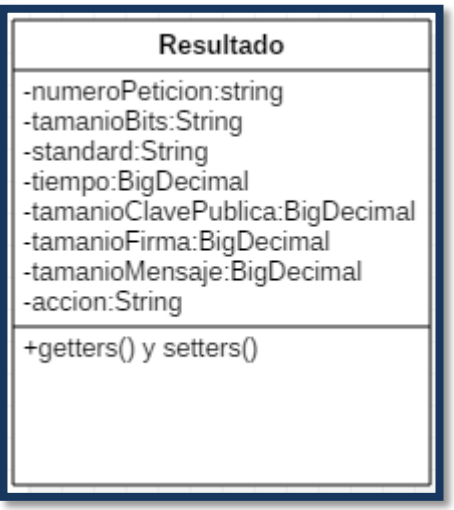


Diagrama de clase DTO

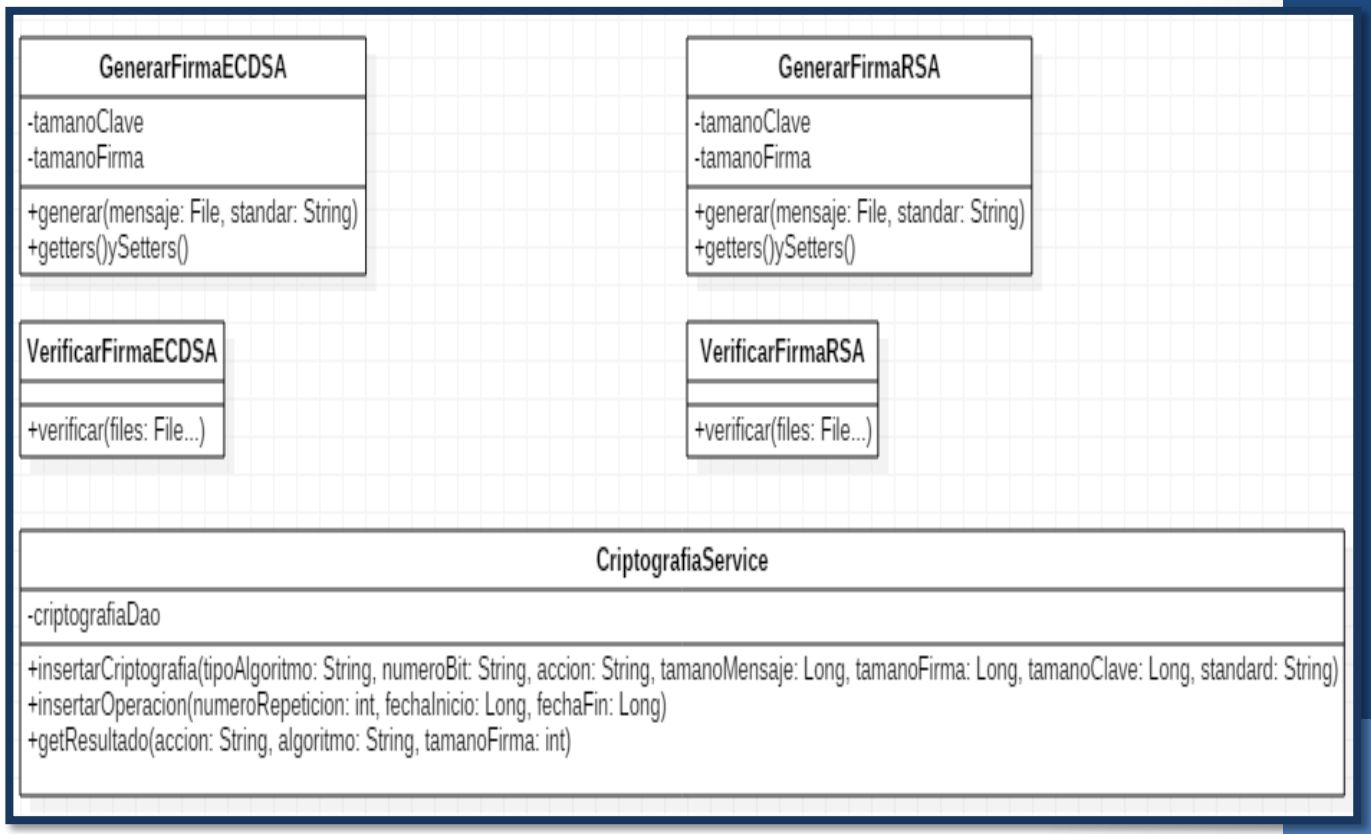


Diagrama de clases de negocio

Diagrama de paquetes

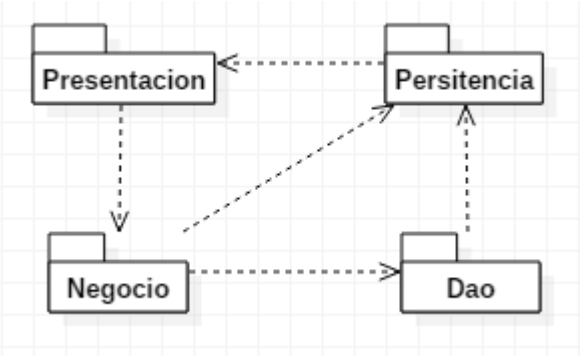
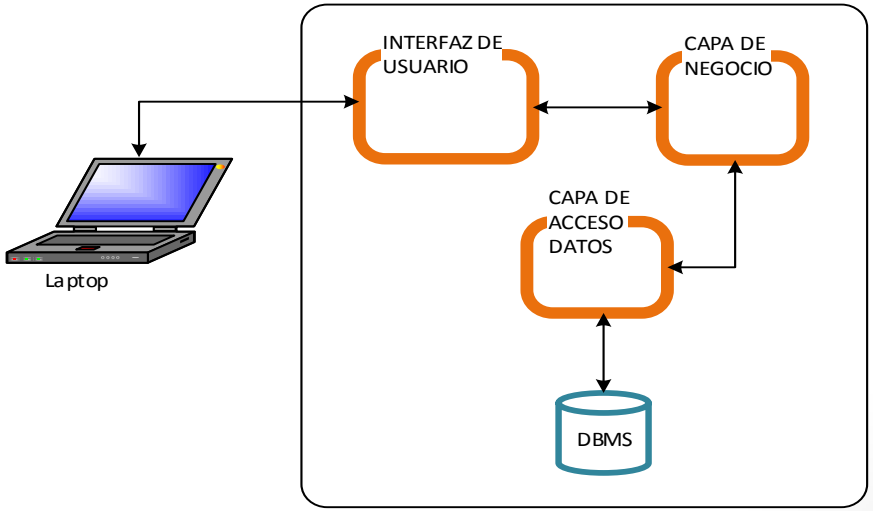


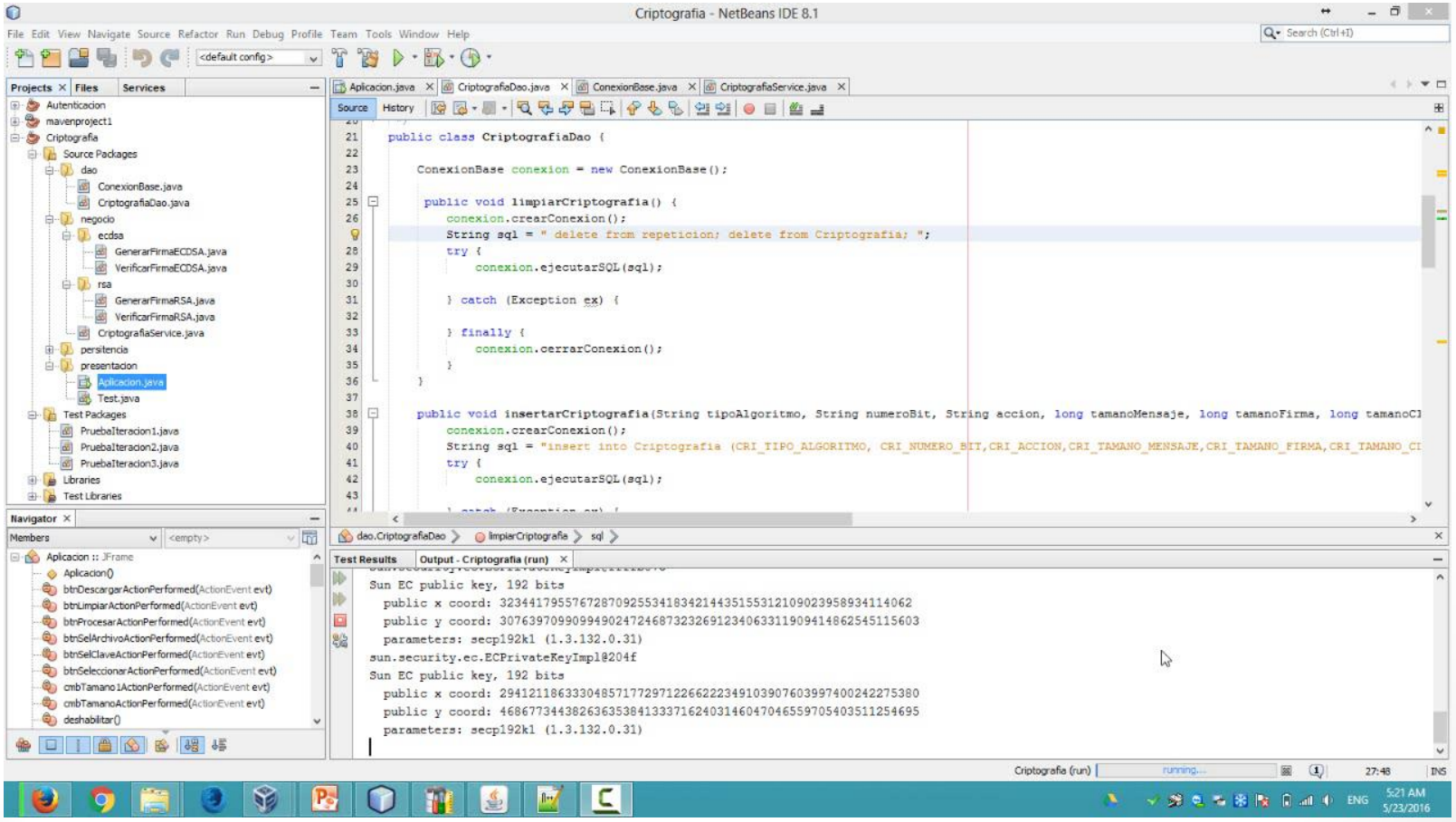
Diagrama de Arquitectura



Product Backlog

Id	Historia de Usuario	Estimación (días)	Prioridad	Criterio de aceptación
1	Capturar y procesar información	5	1	Ingresar al aplicativo, seleccionar: la acción a realizar, el algoritmo con el cuál se trabajará, el tamaño de firma y el mensaje o clave pública o archivo firmado, según la acción a realizar.
2	Generar Archivos	5	1	Seleccionar la opción: Generar, que se encontrará activa en caso que se desee generar los archivos de firma digital y clave pública. Si en el proceso ha sido exitoso los archivos podrán ser visualizados en el lugar de instalación del aplicativo.
3	Almacenar información	3	2	Al seleccionar Generar o Procesar, la información de ingreso, así como tiempos de respuesta y datos relacionados son almacenados en la base de datos.
4	Generar reporte de actividad en la aplicación	3	3	Seleccionar la pestaña Consultar, completar la información de ingreso (acción, algoritmo y tamaño de firma) y dar clic en buscar. Como resultado se obtiene un reporte con la especificación de la información procesada y previamente almacenada en la base de datos.

Aplicativo

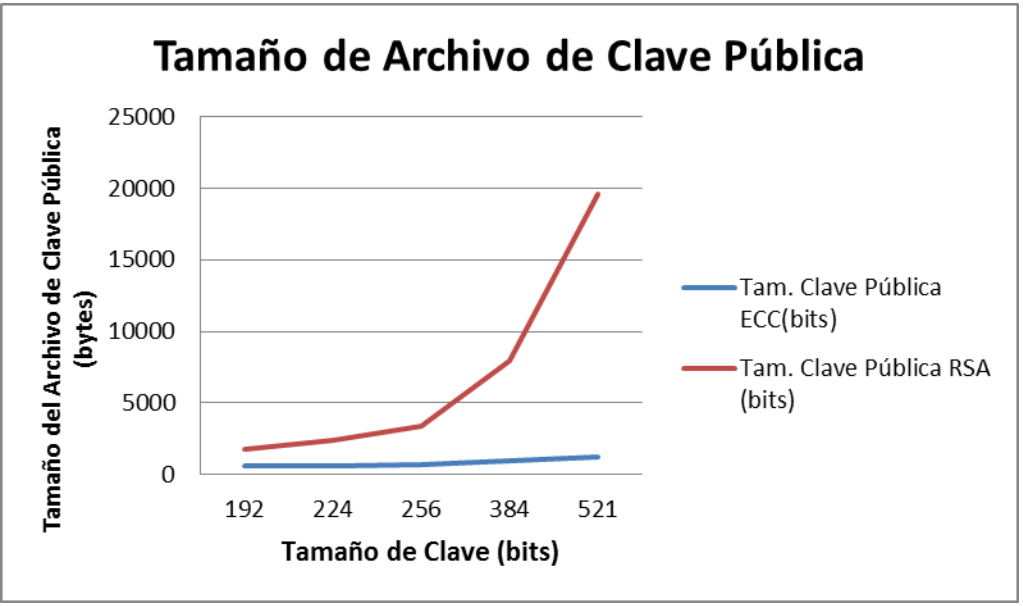

Criptografia - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
Autenticacion
mavenproject1
Criptografia
Source Packages
dao
ConexionBase.java
CriptografiaDao.java
negocio
ecdsa
GenerarFirmaECDSA.java
VerificarFirmaECDSA.java
rsa
GenerarFirmaRSA.java
VerificarFirmaRSA.java
CriptografiaService.java
persitencia
presentacion
Aplicacion.java
Test.java
Test Packages
PruebaIteracion1.java
PruebaIteracion2.java
PruebaIteracion3.java
Libraries
Test Libraries
Navigator
Aplicacion :: JFrame
Aplicacion()
btnDescargarActionPerformed(ActionEvent evt)
btnLimpiarActionPerformed(ActionEvent evt)
btnProcesarActionPerformed(ActionEvent evt)
btnSelArchivoActionPerformed(ActionEvent evt)
btnSelClaveActionPerformed(ActionEvent evt)
btnSeleccionarActionPerformed(ActionEvent evt)
cmbTamano1ActionPerformed(ActionEvent evt)
cmbTamanoActionPerformed(ActionEvent evt)
deshabilitar()
Aplicacion.java
CriptografiaDao.java
ConexionBase.java
CriptografiaService.java
Source History
public class CriptografiaDao {
ConexionBase conexion = new ConexionBase();
public void limpiarCriptografia() {
conexion.crearConexion();
String sql = " delete from repeticion; delete from Criptografia; ";
try {
conexion.ejecutarSQL(sql);
} catch (Exception ex) {
} finally {
conexion.cerrarConexion();
}
}
public void insertarCriptografia(String tipoAlgoritmo, String numeroBit, String accion, long tamanoMensaje, long tamanoFirma, long tamanoCl
conexion.crearConexion();
String sql = "insert into Criptografia (CRI_TIPO_ALGORITMO, CRI_NUMERO_BIT,CRI_ACCION,CRI_TAMANO_MENSAJE,CRI_TAMANO_FIRMA,CRI_TAMANO_CI
try {
conexion.ejecutarSQL(sql);
dao.CriptografiaDao
limpiarCriptografia
sql
Test Results
Output - Criptografia (run)
Sun EC public key, 192 bits
public x coord: 323441795576728709255341834214435155312109023958934114062
public y coord: 307639709909949024724687323269123406331190941862545115603
parameters: secp192k1 (1.3.132.0.31)
sun.security.ec.ECPrivateKeyImpl@204f
Sun EC public key, 192 bits
public x coord: 294121186333048571772971226622234910390760399740024227538O
public y coord: 468677344382636353841333716240314604704655970540351125469S
parameters: secp192k1 (1.3.132.0.31)
Criptografia (run)
running...
27:48
INS
ENG
5:21 AM
5/23/2016

Pruebas y Evaluación complementaria

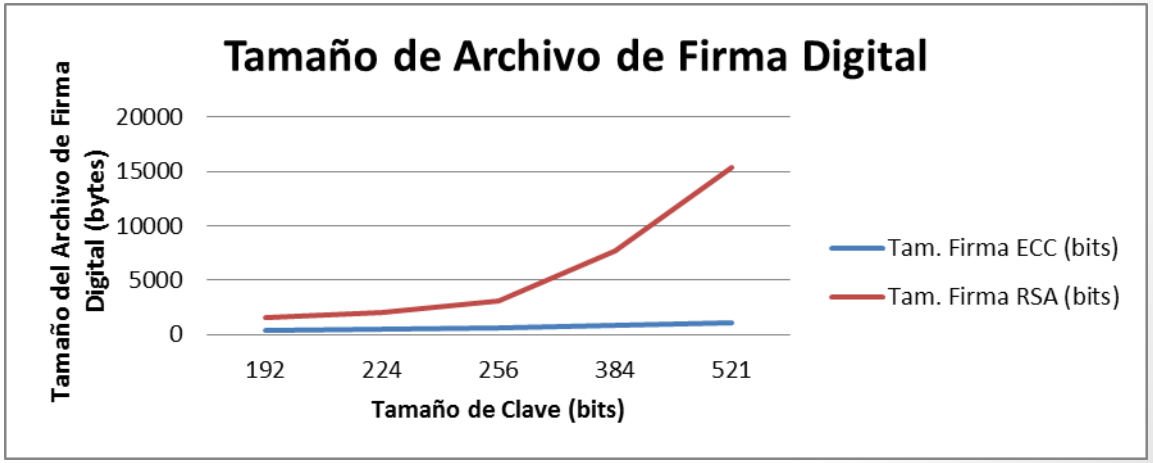
Tamaño de Clave (bits)	Estándar	ECDSA					Equivalente RSA (bits)	RSA			
		Generación de la Firma			Verificación de la Firma			Generación de la Firma			Verificación de la Firma
		Tiempo ECDSA (ms)	Tam. Clave Pública ECC(bits)	Tam. Firma ECC (bits)	Tiempo ECDSA (ms)			Tiempo RSA (ms)	Tam. Clave Pública RSA (bits)	Tam. Firma RSA (bits)	Tiempo RSA (ms)
192	secp192r1	11072	600	440	10992		1536	11404	1808	1536	10974
224	secp224r1	11380	640	512	11329		2048	12006	2352	2048	10992
256	secp256k1	11307	704	568	11135		3072	12439	3376	3072	10945
384	secp384r1	11248	960	832	11098		7680	54656	7984	7680	11020
521	secp521r1	11313	1264	1112	11176		15360	1225960	19588	15360	11029

Resumen de generación y verificación de firma digital.

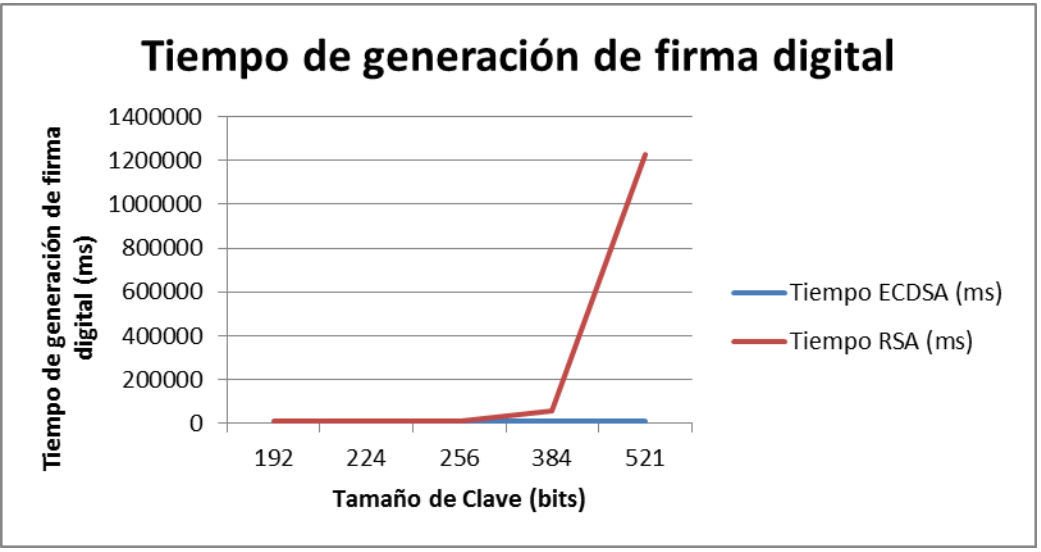
Pruebas y Evaluación complementaria



Comparación de archivos de clave pública generados

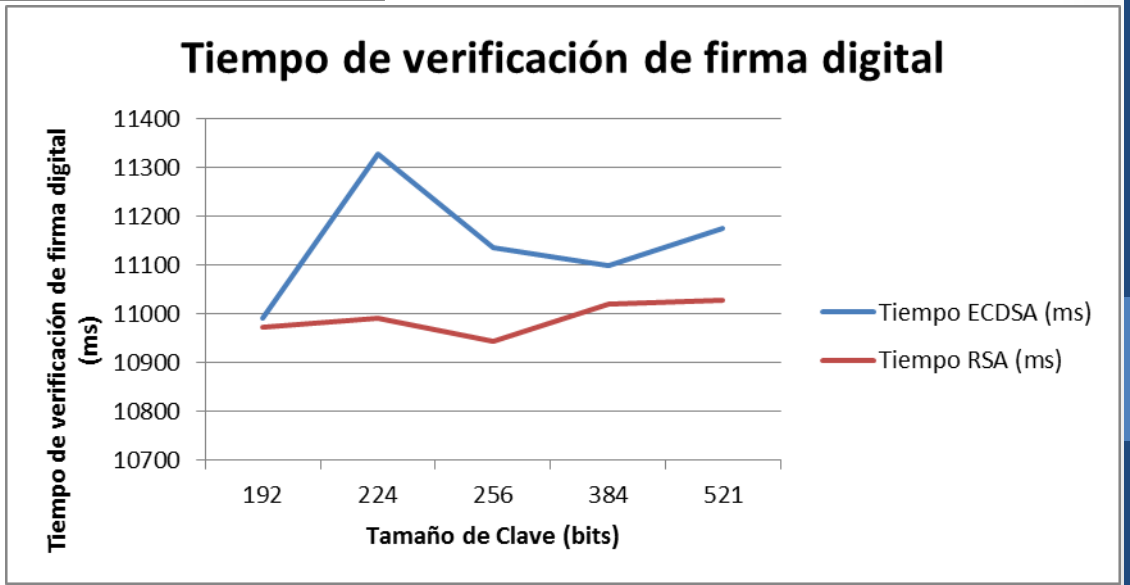


Pruebas y Evaluación complementaria



Comparación de Tiempo en generación de firma digital

Comparación de Tiempo de verificación de firma digital



CONCLUSIONES

- En la actualidad la mayoría de aplicaciones están orientadas a la web o dispositivos móviles por lo que es importante que el tamaño de los archivos generados sea el menor dado para mejorar los tiempos de transaccionalidad, y se ha comprobado con el presente estudio que el algoritmo de curva elíptica ECDSA es quien realiza este requerimiento posible en comparación con el algoritmo RSA

- Los algoritmos de firma digital basados en curvas elípticas son matemáticamente más complejos por lo que el proceso de descifrado en caso de un ataque es más demoroso, brindando así mayor seguridad de la información.

CONCLUSIONES

- ECDSA necesita un menor tamaño de clave para ofrecer el mismo nivel de seguridad brindado por RSA.

- Los tiempos de generación de la firma digital obtenidos por el algoritmo RSA son altos en comparación son los obtenidos por ECDSA y su diferencia se incrementa notoriamente proporcionalmente al tamaño de clave.

Trabajos citados

Amaya, L. A. (2014). *Memorias de Seminario El algoritmo de Elgamal.* Colombia.

Cristina Alcaraz, R. R. (s.f.). *Análisis de Primitivas Criptográficas para Redes de Sensores.* Málaga.

Grajales, T. (s.f.). *Tipos de Investigación*. Recuperado el 17 de 04 de 2014, de http://www.iupuebla.com/Maestrias/M_E_GENERO/MA_Maestria_Genero/Jose_Miguel_Velez/Tipos%20de%20investigacion.pdf

Jesús J. Ortega Triguero, M. Á. (2006). *Introducción a la Criptografía Historia y Actualidad.* La Mancha: Ediciones de la Universidad de Castilla.

Koblintz, N. (1994). *A course in Number Theory and Cryptography, segunda edición.*

Marcelo Andrés Saravia Gallardo, M. A. (s.f.). Metodología de Investigación Científica. Valle de méxico.

Pino Caballero Gil. (s.f.). *Introducción a la Criptografía.* Alfaomega.

postgrado, i. p. (s.f.). *Metodología de la investigación para elaborar una tesis*. Recuperado el 20 de 04 de 2014, de http://www.slideshare.net/juanchojuancho/metodologa-de-investigacin-para-elaborar-una-tesis-25826565?qid=908c70e7-b3c8-4337-be2c-0fab403786ea&v=qf1&b=&from_search=10

Santiago Fernandez. (2004). *Criptografía Clásica.*

Tilborg, H. C. (2005). Encyclopedia of Cryptography and Security. Springer.

Trujillo, R. T. (2011). *PROTOCOLO CRIPTOGRÁFICO*.

Wilder, M. (2006). *Test de primalidad, aplicación a la criptografía.* Palermo.

William Raúl García Muñoz, A. V. (2005). Estudio técnico para la implementación de una entidad certificadora para el CTT – ESPE CECAI. Sangolquí.

GRACIAS POR SU ATENCIÓN…