4.El proyecto

Desde el punto de vista de la seguridad, el conjunto de clases de seguridad distribuidas con el Java 2 SDK pueden dividirse en dos subconjuntos: clases relacionadas con el control de acceso, la gestión de permisos y las clases relacionadas con la criptografía.

Java proporciona funciones criptográficas de propósito general, conocidas colectivamente como la *Arquitectura Criptográfica de Java (JCA)* y la *Extension Criptográfica de Java (JCE)*. El JCA está formado por las clases básicas relacionadas con la criptografía y el soporte para la protección lo proporciona el paquete de extensión JCE.

La librería JCA es un marco de trabajo para acceder y desarrollar funciones criptográficas en la plataforma Java. Se diseñó alrededor de dos principios básicos, la independencia e interoperabilidad de las implementaciones y la independencia y extensibilidad de los algoritmos.

Las principales funciones criptográficas implementadas son: encriptar datos, desencriptar datos, encriptar claves, desencriptar claves, generar claves, generar otros parámetros, realizar conversiones entre distintas representaciones de los parámetros y de las claves, generar y verificar resúmenes de mensajes, códigos de autentificación de mensajes y firmas digitales.

Sin embargo, utilizar la arquitectura criptográfica de Java (JCA) y su extensión (JCE) resulta demasiado complicado. Es por ello por lo que se ha decidido desarrollar una nueva librería criptográfica llamada JCEF (Java Cryptographic Extension Framework), en lugar de mostrar un conjunto de aplicaciones concretas de la criptografía. JCEF se soporta sobre JCA y JCE facilitando enormemente su uso. Esta nueva librería sería mayormente una fachada de las librerías JCA y JCE. El cliente de esta nueva librería sería el desarrollador de software que necesita usar sistemas criptográficos de seguridad con suma facilidad sin las complejidades inherentes a JCA y JCE.

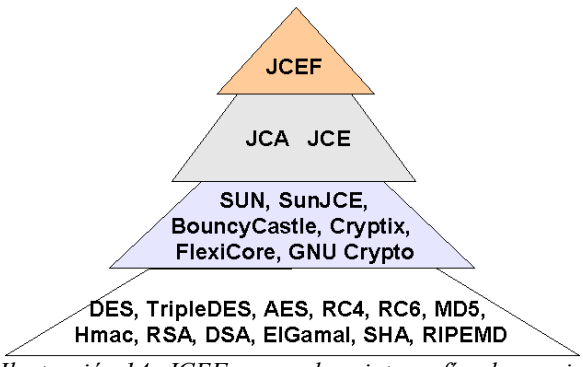

Ilustración 14: JCEF acerca la criptografía al usuario

A continuación se describirán las deficiencias encontradas y mejoradas en JCE (y JCA) y las mejoras realizadas por JCEF al mundo de la criptografía mediante Java. Y finalmente se muestran las mejoras que están pendientes de realizarse en futuras versiones de este proyecto. De esta forma, un usuario podrá decidir convenientemente cuál de las dos librerías debería usar.

JCEF mejora enormemente a JCE. Las deficiencias de JCE que han sido mejoradas son:

- JCE deja mucho que desear, puesto que la calidad de su código y diseño es baja.
- Dificulta su uso y aprendizaje al utilizar términos poco comprensibles.
- No proporciona al usuario un uso sencillo de sus funciones.
- Generar autentificadores es complicado al igual que su verificación, principalmente de huellas y sellos digitales al exigir que el usuario conozca cómo se verifican autentificadores como las huellas, sellos o firmas digitales.
- La protección y desprotección es muy complicada de utilizar.
- Los algoritmos hay que inicializarlos de una forma anticuada y engorrosa.
- Los parámetros se generan y gestionan de una forma nada sencilla.
- JCE no permite construir objetos seguros para cualquier algoritmo criptográfico.
- No reconstruye automáticamente las operaciones tras un cambio.
- Proporciona un modo de operación muy particular para cada algoritmo criptográfico.
- Finalmente, para terminar con la paciencia de cualquier usuario, la gestión de los proveedores es complicada y desgraciadamente necesaria.
- Los algoritmos JCE no se pueden configurar en tiempo de ejecución ni tampoco se puede probar su funcionamiento con facilidad.
- Para definir algoritmos se utilizan clases diferentes a las que se usan en la utilización de los algoritmos.

En definitiva, utilizar JCE requiere un gran esfuerzo en tiempo y dedicación al usuario del mismo. Sin embargo, JCEF proporciona los siguientes valores añadidos:

- Proporciona una serie de mejoras muy significativas.
- Vale la pena usarlo, es amigable y fácil de utilizar.
- Se aprende de forma intuitiva y no requiere demasiado tiempo de aprendizaje.
- No se cometen errores fácilmente y no requiere grandes conocimientos técnicos.
- Evita el uso de conceptos técnicos, y además, su diseño y código son de mayor calidad.
- Simplifica enormemente la autentificación, haciéndola sencilla y homogénea, tanto la generación como verificación de huellas, sellos y firmas digitales; por lo que la autentificación se aprende de forma más sencilla y rápida.
- Proporciona un uso sencillo de los algoritmos, además de hacer que éstos sean reutilizables.

- Suministra una jerarquía de objetos muy completa.
- Permite seleccionar los algoritmos de una forma muy sencilla.
- La inicialización de los algoritmos es infinitamente más sencilla.
- Proporciona adaptaciones de proveedores criptográficos ya existentes que cumplen con la especificación JCE o no. Incluso permite definir fácilmente nuevos algoritmos con especificación JCEF.
- Permite de forma sencilla obtener información completa de inicialización y de configuración.
- Proporciona gestión adeacuada de los parámetros de inicialización y configuración.
- Permite cambiar fácilmente de algoritmo y de implementación.
- Para reducir la dificultad, las operaciones de bajo nivel son automáticas, transparentes y sencillas, de esta forma es innecesario realizar un sinfín de operaciones como por ejemplo la encapsulación y desencapsulación de claves.
- Los algoritmos aceptan los parámetros en cualquiera de sus formas con el objetivo de mejorar la usabilidad.
- Aún hay más, JCEF potencia la creación de proveedores criptográficos personalizados.
- Se generan de forma automática todos los parámetros de los algoritmos.
- Para continuar simplificando, es posible evitar la realización de ciertas operaciones.
- Es innecesario conocer los algoritmos secundarios de los algoritmos criptográficos.
- Con JCEF, no hace falta tener conocimientos importantes.
- Proporciona mecanismos para la definición de algoritmos criptográficos compuestos.
- Permite construir objetos seguros de forma sencilla.
- La implementación y el algoritmo poseen la misma especificación.
- Proporciona un marco sencillo para pruebas de funcionalidad e implementación.
- Recopila una gran cantidad de algoritmos criptográficos y proveedores.

Por desgracia, JCEF no es perfecto y todavía puede y debe seguir creciendo. Algunas de las posibles mejoras que se le pueden hacer son:

- Proporcionar todas aquellas características que JCE proporciona y JCEF no, tales como los flujos de entrada/salida seguros, gestión de certificados digitales, almacenes seguros, protocolos de intercambio de claves, etc...
- Implementar flujos de entrada/salida seguros de forma sencilla para cualquier algoritmo criptográfico.
- Proporcionar algunos aspectos avanzados, tales como almacenes seguros para cualquier tipo de objetos incluyendo claves y certificados.
- Permitir gestionar certificados de una manera muy sencilla.

- Proporcionar sencillos mecanismos para los protocolos de intercambio de claves.
- Mejorar la gestión de los modos de operación y los esquemas de relleno en los algoritmos criptográficos de protección ya existentes.
- Verificar el correcto funcionamiento de todos los algoritmos criptográficos a través del uso de vectores de tests y corregir aquellos incorrectos.
- Comprobar el correcto funcionamiento de las pruebas de implementación.
- Terminar de implementar los algoritmos compuestos, probarlos y añadir a todos los proveedores JCEF existentes nuevos algoritmos compuestos reutilizando sus algoritmos criptográficos.
- Comprobar el correcto funcionamiento de algunos detalles de las implementaciones JCEF.
- Reducir la dependencia con otros proyectos.
- Aumentar el nivel de confianza de este proyecto.
- Implementar algoritmos nuevos no implementados hasta ahora en Java.

Como un ejemplo del valor añadido de este proyecto, observe el código siguiente donde se muestra cómo se asegura un objeto e inmediatamente después se recupera el mismo; y todo ello de una forma super sencilla:

```
Object object = new String("my object");
CryptoAlgorithm secureAlgorithm = new AES_BlockSymmetricProtectionRREXKY();
SecureObject secureObject = new SecureObject(object, secureAlgorithm);
object = (String)secureObject.getObject(secureAlgorithm);
```

Tabla 35: Pequeño ejemplo de uno de los valores añadidos de JCEF

En las próximas secciones podrán encontrarse más detalles sobre las deficiencias, mejoras realizadas y pendientes y otros detalles del proyecto como parte de su valor añadido, comparativa con lo que había antes (JCA-JCE), detalles de implementación, presentación, distribuciones, etc...

Si lo desea, también puede consultar la página web oficial del proyecto http://jcef.sourceforge.net y los recursos utilizados http://jcef.sourceforge.net/doc/resources.pdf.

1 Deficiencias mejoradas

JCE deja mucho que desear puesto que:

- La calidad de su código es baja.
- Dificulta su uso y aprendizaje.
- Se utilizan términos poco comprensibles.
- Su uso es muy complejo.
- La generación y verificación de autentificadores es complicada.

La calidad del código es baja, ya que:

- Complica el mantenimiento del código.
- Dificulta su reutilización.
- Complica el código volviéndolo incomprensible.
- Complica la accesibilidad del código volviéndolo cerrado.
- Se desconoce si su funcionamiento es totalmente correcto.
- No proporciona ningún conjunto de pruebas que se puedan utilizar.
- Carece de una documentación útil y completa.

Dificulta su uso y aprendizaje. Esto se debe principalmente a que proporciona muchísimos métodos no fundamentales, lo que hace que el usuario se pierda y acabe por no utilizar esta herramienta al haber demasiadas funciones donde elegir. Por dar algunos detalles, JCE proporciona múltiples formas de indicar los datos, ya sea para proteger o autentificarlos. Es decir, suministra gestión de arrays de bytes, por lo que rompe el patrón experto, o sea, realiza funciones que no le corresponde. Además, esto dificulta el aprendizaje ya que hay más funciones que aprender.

Se utilizan términos poco comprensibles para el gran público, tales como resumen de mensaje, código de autentificación de mensaje, encriptar, desencriptar, etc... Es decir, no evita el uso de conceptos técnicos al utilizar nombres complejos e incomprensibles para la mayoría del público.

Su uso es muy complejo. Esto se debe principalmente a que tanto la autentificación como la protección de datos son difíciles de utilizar porque requiere tener muchísimos aspectos en cuenta. JCE no realiza de forma automática muchas operaciones de bajo nivel, es decir, le deja mucha responsabilidad al usuario. Y todas sus operaciones son extremadamente difíciles de utilizar, aprender y mantener.

La generación de autentificadores es complicada, tanto la generación de huellas, sellos y firmas digitales, ya que cada uno de ellos se generan de forma muy distinta, proporcionando heterogeneidad y complejidad. JCE proporciona formas inapropiadas para generar cualquier tipo de autentificadores. Además, tampoco proporciona ninguna base común para los autentificadores ni los algoritmos que los gestionan. Con JCE se pueden utilizar más de 15 formas distintas para generar simplemente un autentificador.

La verificación de autentificadores es complicada, debido a que complica la verificación tanto de huellas, sellos y firmas digitales. Esto se debe principalmente al uso de multitud de métodos inapropiados para la verificación de los autentificadores ya que existen muchísimas formas de verificar un autentificador con JCE, cuando en realidad sólo una es la realmente útil. Además, tampoco proporciona ninguna base común para la verificación de autentificadores, es decir, las huellas, sellos y firmas digitales no son tratados de forma homogénea.

La verificación de huellas y sellos digitales es complicada. Esto sucede principalmente porque no se proporciona ninguna base común para los algoritmos de autentificación y porque es necesario conocer que para verificar la autenticidad de una huella digital hay que generar una nueva huella digital y compararla con la original; si son iguales entonces es auténtica, si no, no lo es. Sin embargo, las firmas digitales son relativamente sencillas de verificar, lo que añade heterogeneidad y complejidad a la verificación de autentificadores.

El usuario necesita saber cómo verificar autentificadores. Se requiere conocer que para comprobar la autenticidad de unos datos mediante huella o sello digital, es preciso volver a generar un nuevo autentificador de los mismos datos y comporarlo con el autentificador original. Sin embargo, el usuario no necesita saber cómo verificar firmas digitales. Esto es un aspecto positivo, pero también lo es negativo al poner de manifiesto que se rompe la homogeneidad debido a los dos puntos anteriores.

La protección y desprotección es compleja por muchos motivos entre los que destacan los siguientes: el modo de operación se indica de forma dificultosa, la inicialización de estos algoritmos no es nada sencilla y se proporcionan múltiples formas para obtener el mismo resultado.

La calidad de su diseño es baja, ya que no es homogéneo, ni actual y es complicado de entender y reutilizar.

Los algoritmos se inicializan de forma compleja debido a que los métodos de inicialización son heterogéneos e inapropiados; y sobre todo al suministrar múltiples formas complejas de realizar la inicialización. Además, para algoritmos muy similares se utilizan mecanismos de inicialización distintos. De esta forma se dificulta el aprendizaje y se complica su utilización. Por otro lado, además es necesario indicar los parámetros de inicialización en un orden concreto y por si esto no fuera poco, es tal la complejidad, que en el caso de los algoritmos de sellos digitales, se hace creer que estos algoritmos aceptar claves asimétricas cuando en realidad sólo aceptan claves simétricas.

Genera parámetros de forma muy compleja y oculta. Algunos algoritmos generan automáticamente algunos parámetros si éstos no son proporcionandos, pero desgraciadamente estas circunstancias no están documentadas y son muy difíciles de detectar. Esto pone de manifiesto lo complejo que puede resultar utilizar JCE.

No permite construir objetos seguros para cualquier algoritmo criptográfico, sólo acepta algoritmos de firmas digitales y protección. Es decir, no permite construir objetos seguros autentificables mediante huella o sello digital.

JCE no reconstruye automáticamente las operaciones tras un cambio, es necesario reconstruir la operación para realizar un cambio.

JCE proporciona un modo de operación particular para cada algoritmo criptográfico. Por ejemplo, para los algoritmos de protección es necesario indicar el modo de operación (protección o desprotección) mediante un entero a modo de tipo enumerado.

La gestión de los proveedores es complicada y necesaria y además no potencia la creación de proveedores criptográficos personalizados. Por otro lado, uno de los principales defectos de esta característica es que no es posible manejar todos los algoritmos criptográficos de la misma forma. Y además, algoritmos idénticos de proveedores distintos poseen nombres distintos.

Los algoritmos JCE no se pueden configuran en tiempo de ejecución. Además, **tampoco se puede probar su funcionamiento con facilidad** y por si no fuera poco, **para definir algoritmos se utilizan clases diferentes a las que se usan en la utilización de los algoritmos**.

2 Mejoras realizadas

JCEF proporciona una serie de mejoras muy significativas tales como:

- Muchos proveedores totalmente reutilizables.
- Además vale la pena su uso para usuarios que sepan muy poco o nada de criptografía y también para aquellos, que como yo, se hayan sentido decepcionados por una herramienta tan compleja de utilizar como JCE.

Vale la pena usar JCEF, debido a que es amigable, fácil de utilizar, se aprende de forma intuitiva, no se cometen errores fácilmente, no requiere grandes conocimientos técnicos y no requiere demasiado tiempo de aprendizaje y se ajusta a las necesidades de la mayoría de los usuarios.

Es amigable debido a que es fácil de utilizar en las situaciones habituales. Aunque también es cierto que puede ser complejo en algunas situaciones excepcionales.

Fácil de utilizar porque se aprende de forma intuitiva, no se cometen errores fácilmente, no requiere grandes conocimientos técnicos, evita el uso de conceptos técnicos y no requiere demasiado tiempo de aprendizaje.

Se aprende de forma intuitiva debido a la calidad del código, su diseño y documentación.

No requiere demasiado tiempo de aprendizaje ya que un aprendiz necesita como mucho una hora para su aprendizaje completo. Sin embargo, aprender por completo a utilizar JCE requiere entre una y cuatro semanas de dedicación. Esta reducción del tiempo de aprendizaje se debe primordialmente a que no se cometen errores fácilmente, al ahorro de operaciones, a la simplificación de muchas operaciones, a que no requiere grandes conocimientos técnicos y se ajusta más a las necesidades del usuario.

No se cometen errores fácilmente debido a la calidad de su código, su diseño, su documentación, al ahorro de operaciones, ...

No requiere grandes conocimientos técnicos. JCEF no requiere que el usuario tenga grandes conocimientos técnicos sobre criptografía, algoritmos criptográficos y su utilización mediante Java. Es decir, exige menor cantidad de conocimiento al usuario. Esto se consigue gracias a que no se usan conceptos técnicos, se utilizan nombres sencillos y comprensibles, se ahorran operaciones, se simplifica el uso de muchas operaciones y se ajusta mayormente a las necesidades del usuario.

Evita el uso de conceptos técnicos. JCEF no utiliza conceptos técnicos como hace JCE. Usa conceptos y nombres muy sencillos y comprensibles para el gran público tales como: autentificadores, protección, desprotección, huella digital, sello digital, firma digital, traductores de claves y parámetros, generadores de autentificadores, verificadores de autentificadores, generador de claves asimétricas, generador de claves simétricas, etc...

Su código es de calidad ya que fácilmente es mantenible, reutilizable y comprensible. Además, también es "Open Source" (Código Abierto) y está totalmente documentado y probado.

Su diseño es de mejor calidad debido a que es homogéneo, actual, simple, está enfocado a las necesidades del usuario, está basado en los principios de diseño contrastados como el de los JavaBeans y fomenta la reutilización.

Simplifica la autentificación, simplificando el uso de los algoritmos de autentificación al proporcionar una base común para todos ellos. Esto provoca además que usar este tipo de algoritmos sea muy sencillo.

La autentificación es sencilla y homogénea. Esto es debido a que la generación y verificación de autentificadores es sencilla, ya que tanto las huellas digitales, como los sellos y las firmas digitales se generan y verifican de forma sencilla; es decir, no es necesario distinguir entre huellas, sellos o firmas digitales.

La generación de huellas, sellos y firmas digitales es sencilla y homogénea porque todos estos autentificadores se generan igual, de manera homogénea y de una única forma.

La verificación de autentificadores es sencilla es sencilla y homogénea, principalmente debido a que tanto las huellas, sellos y firmas digitales se verifican de la misma forma, que además es homogénea y única, tal y como sucede con la generación.

La autentificación se aprende de forma más sencilla y rápida debido a su simplificación, sencillez y homogeneidad. En resumen, si se aprende a generar un autentificador, automáticamente se aprende a generar el resto de tipos de autentificadores. En general, sólo basta con aprender a generar y verificar autentificadores.

Proporciona un uso sencillo de los algoritmos además de hacer que éstos sean reutilizables, ya que proporciona bases comunes para todos los tipos de algoritmos, se utilizan principios de diseño adecuados y se proporciona un modo común homogéneo de uso. JCEF hace que sea sencillo realizar las siguientes funciones:

- Seleccionar, configurar, inicializar y utilizar algoritmos para asegurar y desasegurar datos.
- Autentificar, generar y verificar autentificadores como huellas, sellos o firmas digitales.
- Proteger y desproteger datos.
- Crear objetos seguros mediante algoritmos criptográficos con independencia de que se trate de algoritmos de protección, ya sean simétricos, asimétricos, de bloques, de flujo o basados en contraseña, o algoritmos de autentificación, se traten de huellas, sellos o firmas digitales.
- Obtener los objetos asegurados a partir de su versión segura.

JCEF proporciona una jerarquía de objetos muy completa. JCEF proporciona las siguientes categorías de objetos: objetos JCEF, proveedores JCEF, algoritmos, generadores, generadores de claves, generadores de claves asimétricas, generadores de claves simétricas, generadores de parámetros, generadores de datos aleatorios, traductores, traductores de claves, traductores de claves asimétricas, traductores de claves simétricas, traductores de parámetros, algoritmos criptográficos, algoritmos criptográficos de autentificación, algoritmos criptográficos de autentificación mediante huellas, sellos y firmas digitales tanto para generación como verificación, algoritmos de sellos digitales mediante contraseña, algoritmos de protección basada en contraseña, algoritmos de protección de datos, algoritmos criptográficos de protección/desprotección asimétrica o simétrica, simétrica de bloques y simétrica de flujo.

JCEF permite seleccionar los algoritmos de una forma muy sencilla, ya que basta simplemente con utilizar el constructor del algoritmo.

La inicialización de los algoritmos es infinitamente más sencilla. JCEF emplea los principios de los JavaBeans para acceder a los objetos, simplemente accediendo a sus propiedades utilizando métodos sencillos de lectura y escritura de las propiedades.

Proporciona un nuevo proveedor completo de algoritmos criptográficos llamado RREXKY.

Proporciona adaptaciones de proveedores criptográficos ya existentes que no cumplen con la especificación JCE. Tales proveedores son Mindbright, Jacksum y LogiCrypto.

Proporciona adaptaciones de proveedores criptográficos ya existentes que sí cumplen con la especificación JCE. Tales proveedores son BouncyCastle, Cryptix, CryptixCrypto, IAIK, GNU Crypto, FlexiCore, FlexiEC, FlexiNF, JHBCI, SUN, SunJCE, SunJSSE y SunRsaSign.

Permite, de forma sencilla, obtener información completa de inicialización y de configuración de los algoritmos, simplemente accediendo a través de los métodos de lectura de dichas propiedades. O dicho de otro modo, proporciona una única forma de inicialización y configuración. También permite distinguir fácilmente entre los distintos tipos de algoritmos.

Permite definir fácilmente nuevos algoritmos con especificación JCEF, al proporcionar un soporte sencillo de ampliación para adaptar los algoritmos existentes desde la especificación JCE y cualquier otra que no sea JCE a la especificación JCEF, además de permitir la definición de algoritmos de nueva implementación.

Proporciona gestión adecuada de los parámetros de inicialización y configuración, ya que:

- Permite recuperar los parámetros de inicialización y configuración.
- Reconstruir la operación automáticamente tras un cambio.
- No complica la reutilización de las operaciones.
- Facilita la reutilización de los algoritmos.
- Permite indicar los parámetros en cualquier orden sin obligar a proporcionarlos en un orden concreto.
- Además, los algoritmos JCEF se pueden reconfigurar en tiempo de ejecución.

Permite cambiar fácilmente de algoritmo y de implementación, ya que los algoritmos se seleccionan de una forma natural, tienen una base común y no es necesario gestionar los proveedores criptográficos.

Las operaciones de bajo nivel son automáticas, transparentes y sencillas. Estas operaciones son: traducción de claves, conversión entre las distintas representaciones de los parámetros y claves y la generación de claves y parámetros.

Es innecesario realizar un sinfín de operaciones. El usuario se ahorra realizar un sinfín de operaciones de forma manual. Tales operaciones son:

- Las operaciones de bajo nivel.
- Las operaciones relacionadas con el uso de los algoritmos.
- Encapsular claves, ya que encapsular una clave es protegerla como si fuera un objeto.
- Desencapsular claves, ya que desencapsular una clave es desprotegerla como si fuera un objeto.
- Generar parámetros y claves tanto simétricas como asimétricas.
- Realizar conversiones entre representaciones de parámetros y claves tanto simétricas como asimétricas.
- Indicar el modo de inicialización para los algoritmos de protección.
- Realizar traducciones de parámetros y claves tanto simétricas como asimétricas.

Es innecesaria la encapsulación/desencapsulación de claves. Estas operaciones son innecesarias ya que una clave es como cualquier otro objeto, por lo que "inventarse" nuevas operaciones tan particulares, dificulta su uso. La operación de encapsular una clave consiste en proteger dicha clave. Por otro lado, la operación de desencapsular una clave se trata de desproteger dicha clave.

Los algoritmos aceptan los parámetros en cualquiera de sus formas. No distingue entre claves conocidas, claves desconocidas, especificadas de forma trasparente, opaca o persistente. Igualmente, no distingue entre parámetros especificados de forma transparente o persistente.

Se generan de forma automática todos los parámetros de los algoritmos, siempre y cuando sean necesarios y no hayan sido especificados.

Se potencia la creación de proveedores criptográficos personalizados. La gestión de proveedores es sencilla, ya que se podría decir que es inexistente a diferencia de JCE, es decir, simplemente cada proveedor contiene un conjunto de algoritmos que puede que hayan implementado los desarrolladores de dicho proveedor o no; por lo que se potencia la creación de proveedores criptográficos personalizados, fomentando a su vez la reutilización.

Es posible evitar la realización de ciertas operaciones, tales como proteger y desproteger datos, generar autentificadores (huellas, sellos y firmas digitales), verificar autentificadores (huellas, sellos y firmas digitales). Para ello se pueden utilizar los objetos seguros.

Es innecesario conocer los algoritmos secundarios de los algoritmos criptográficos, tales como los generadores de claves (simétricas y asimétricas), generadores de parámetros, conversores y traductores entre las distintas representaciones de los parámetros y conversores y traductores entre las distintas representaciones de las claves (simétricas y asimétricas).

Es innecesario tener conocimientos importantes. JCEF reduce el conocimiento necesario para el usuario, ya que es innecesario:

- Conocer los detalles técnicos sobre las distintas claves.
- Tener conocimientos importantes para utilizar un algoritmo, ya que sólo basta con conocer el algoritmo que se desea utilizar.
- Diferenciar una clave de un objeto.
- Conocer cómo se verifican los autentificadores.
- Conocer las distintas representaciones de los parámetros.
- Conocer las distintas representaciones de las claves tanto simétricas como asimétricas.
- Conocer las clases de cada una de las representaciones de los parámetros.
- Conocer las clases de cada una de las representaciones de las claves tanto simétricas como asimétricas.
- Conocer los parámetros de configuración de los algoritmos.
- Conocer los parámetros para los generadores de claves tanto simétricas como asimétricas.
- Conocer los parámetros para los generadores de parámetros.

- Conocer los parámetros para los traductores de claves tanto simétricas como asimétricas.
- Conocer los parámetros para los conversores entre las distintas representaciones de las claves tanto simétricas como asimétricas.
- Conocer los parámetros para los conversores entre las distintas representaciones de los parámetros
- Conocer los parámetros para los traductores de claves tanto simétricas como asimétricas.
- Conocer los parámetros para los algoritmos criptográficos de protección, ya sea protección asimétrica, protección simétrica, simétrica de bloques o de flujo, o basada en contraseña.
- Conocer los parámetros para los algoritmos criptográficos de autentificación, ya sean de huellas, sellos, sellos basados en contraseña o firmas digitales.

Proporciona funciones especiales, tales como los algoritmos criptográficos compuestos y el marco para la realización de pruebas con el objetivo de comprobar el correcto funcionamiento de los algoritmos.

Permite construir objetos seguros de forma sencilla, ya sean objetos protegidos, o autentificables mediante huella, sello o firma digital.

La implementación y el algoritmo poseen la misma especificación. A diferencia de JCE, JCEF emplea los algoritmos directamente con la posibilidad de no perder la abstracción de la implementación de los algoritmos. Y además, utilizar e implementar un algoritmo se realiza del mismo modo.

Proporciona un marco sencillo para pruebas, basado en la verificación de un conjunto de pruebas más sencillas con el objetivo de comprobar el correcto funcionamiento del algoritmo y de su implementación. Se utiliza la reflexión para simplificar el código de las pruebas. Además, realiza pruebas generales para todos los algoritmos.

JCEF recopila una gran cantidad de algoritmos criptográficos:

- Huellas digitales: MD2, MD4, MD5, SHA1, SHA224, SHA256, SHA384, SHA512, RIPEMD128, RIPEMD160, RIPEMD256, RIPEMD320, Tiger, Whirlpool, HashGOST.
- Sellos digitales: HmacMD2, HmacMD4, HmacMD5, HmacSHA1, HmacSHA224, HmacSHA256, HmacSHA384, HmacSHA512, HmacRIPEMD128, HmacRIPEMD160, HmacTiger, MacDES, MacTripleDES, MacRC2, MacRC5, MacIDEA, MacSkipjack.
- Sellos digitales basados en contraseña: PBEwithHmacSHA1, PBEwithHmacRIPEMD160.
- Firmas digitales (incluyen variantes): MD2withRSA, MD4withRSA, MD5withRSA, RIPEMD128withRSA, RIPEMD160withRSA, RIPEMD256withRSA, SHA1withRSA, SHA224withRSA, SHA256withRSA, SHA384withRSA, SHA512withRSA, SHA1withDSA, NONEwithDSA, SignatureGOST, NONEwithRSA, MD5andSHA1withRSA.
- Protección simétricas de bloques: DES, TripleDES, Blowfish, RC2, AES, Skipjack, CAST5, CAST6, Square, RC6, MARS, IDEA, Serpent, Twofish, GOST, RC5, RC5-64.

- Modos de operación: ECB, CBC, PCBC, CTR, CFB<n>, OFB<n>, OpenPGPCFB, OpenPGPCFBwithIv, GOFB.
- Esquemas de relleno: ISO10126Padding, X9.23Padding, withCTS, TBCPadding, ZeroBytePadding, PKCS1Padding, PKCS5Padding, PKCS7Padding, SSL3Padding, ISO78164Padding.
- Protección simétrica de flujo: RC4.
- Protección asimétrica (incluyen variantes): RSA, RSA con huellas digitales (MD2, MD4, MD5, SHA1, SHA224, SHA256, SHA384, SHA512, RIPEMD128, RIPEMD160, Tiger), ElGamal, ECIES.
- Protección basada en contraseña: PBE con algoritmos de huellas digitales (MD5,SHA1, SHA256) y protección simétrica (DES, TripleDES, RC2, IDEA, RC4, AES).

3 Mejoras pendientes

JCE proporciona unas características que de momento no posee JCEF, como son los protocolos de intercambio de claves, almacenes de claves y certificados, la gestión de certificados y flujos de entrada/salida seguros.

Sería importante implementar flujos de entrada/salida seguros de forma sencilla para cualquier algoritmo criptográfico, ya sea de protección (simétrica, asimétrica, de bloques, de flujo, o basada en contraseña) o autentificación (huellas digitales, sellos digitales, firmas digitales o sellos digitales basados en contraseña). Por lo tanto, una futura versión de JCEF, debería incluir, flujos de entrada/salida seguros mediante cualquier algoritmo criptográfico. Cabe destacar que JCE sólo proporciona y de manera compleja, flujos de entrada/salida para algoritmos criptográficos de protección y de autentificación sólo mediante huellas digitales.

En futuras versiones también deberá proporcionar algunos aspectos avanzados, tales como:

- Almacenes seguros para cualquier tipo de objetos, incluyendo claves y certificados.
- Sencilla gestión de certificados.
- Uso sencillo de protocolos de intercambio de claves.

Mejorar la gestión de los modos de operación y los esquemas de relleno en los algoritmos criptográficos de protección ya existentes también será importante en futuras versiones.

Verificar el correcto funcionamiento de todos los algoritmos criptográficos a través del uso de vectores de tests y corregir aquellos incorrectos. De momento se puede confiar en los algoritmos proporcionados por el proveedor JCEF "RREXKY". Decir que esta tarea debería realizarse empezando por los proveedores más importantes y terminando por los menos importantes. La lista de proveedores JCEF de mayor a menor importancia sería: "RREXKY", "GNU Crypto", "BouncyCastle", "FlexiCore", "FlexiEC", "FlexiNF", "Jacksum", "Cryptix", "CryptixCrypto", "SUN", "SunJCE", "SunJSSE", "SunRsaSign", "JHBCI", "IAIK", "Mindbright" y "LogiCrypto".

Terminar de comprobar el correcto funcionamiento de las pruebas de implementación y de algunos detalles de las implementaciones JCEF, como bien pudiera ser la verificación de que todos los algoritmos JCEF se pueden reutilizar con JCE.

Terminar de implementar los algoritmos compuestos, probarlos y añadir a todos los proveedores JCEF existentes nuevos algoritmos compuestos reutilizando sus algoritmos criptográficos ya existentes.

Importante mejorar la dependencia con otras librerías. Sería interesante independizar al máximo a JCEF de librerías de terceros.

Se ve indispensable aumentar el nivel de confianza de esta librería, ya que hay que tener en cuenta que un trabajo desarrollado por un alumno de una universidad no proporciona el mismo nivel de confianza que el trabajo desarrollado por una empresa conocida y establecida desde hace bastantes años. Para ello se podrá, por ejemplo, mejorar las pruebas sobre los algoritmos.

Implementar algoritmos nuevos no implementados hasta ahora en Java de huellas digitales, sellos digitales, firmas digitales, protección asimétrica, protección simétrica de flujo, protección simétrica de bloques, y aumentar la cantidad de modos de operación y esquemas de relleno, tales como:

- Huellas digitales: Snefru128 (Tamaño de la huella en bits = 128), Snefru256 (256), Tiger2 (192), HAS-160 (160), N-Hash (128), SaphireII128 (128), SaphireII160 (160), SaphireII192 (192), SaphireII224 (224), SaphireII256 (256), SaphireII288 (288), SaphireII320 (320), SquareHash, XOR16 (16), XOR32 (32), PMD128 (128), PMD160 (160), PMD192 (192), PMD224 (224), PMD256 (256), CRC16CCITT (16), CRC32 (32), Panama (256).
- Sellos digitales:OMAC y PMAC.
- Firmas digitales: GMR, KCDSA, Schnorr, Desmedt, Lamport-Diffie, Rabin, Matyas-Meyer.
- Protección asimétrica: Rabin, Merkle-Hellman, Paillier, McEliece, Cayley-Purser, Williams, Goldwasser-Micali.
- Protección simétrica de flujo: ORYX, SEAL, Chameleon, Fish, A5/1, A5/2, Helix, Pike, Panama (Tamaño de clave en bits = 256), Sober (128), Wake (128), SCOP, Snow (128), SN3, Turing, Leviathan, MUGI, Sapphire2 ([8, 8, 2040]), Scream (128), SEAL3 (128), Cloak (992) y Solitare.
- Protección simétrica de bloques: 3-Way (Tamaño de bloque en bits = 96, Tamaños de clave en bits = 96), SHACAL (160, [128, 512]), SHACAL2 (256, [128, 512]), Magenta (128, {128, 192, 256}), DEAL (128, {128, 192, 256}), FEAL (64, {64, 128}), KHAFRE (64, {64, 128}), ICE (64, [64, 64, ...)), Madryga, LOKI89 (64, 64), LOKI91 (128, {128, 192, 256}), LOKI97 (128, {128, 192, 256}), Akelarre (128, [64, 64, ...)), KASUMI (64, 128), MISTY1 (64, 128), MISTY2 (64, 128), TEA (64, 128), XTEA (64, 128), NewDES (64, 120), Red Pike (64, 64), CMEA (64, -), GDES (64, 64), GOST768 (64, 768), IBC (256, 160), S-1, Diamond2 (128, [8, 8, 65536]), Diamond2Lite (128, [8, 8, 65536]), REDOC2 (80, 160), REDOC3 (80, [0, 20480)), KHUFU (64, 512), DESX (64, {128, 192}), Lucifer (128, 128), MMB (128, 128), SEED (128, 128), Shark (64, 128), MacGuffin (64, 128), Bear, Lion, Lioness, LadderDES, HPC, Noekeon (128, 128), SPEED64 (64, [48, 16, 256]),

SPEED128 (128, [48, 16, 256]), SPEED256 (128, [48, 16, 256]), FROG ([64, 1024], [40, 1000]), Q128 (128, 128), SaferK40 (64, 40), SaferSK40 (64, 40), SaferK64 (64, 64), SaferSK64 (64, 64), SaferK128 (64, 128), SaferSK128 (64, 128), Safer+ (128, {128, 192, 256}), Safer++ (128, {128, 256}), Crypton (128), DFC (128, {128, 192, 256}), E2 (128, {128, 192, 256}), CS-Cipher (64, [0, 8, 128]), MDC (?, [64, 8, 640]), Rainbow (128, [128, 32, 256]), NSEA (128, 128), MPJ (128, 128), Cobra (128, [1, 1, 1152])].

- Modos de operación [PBC, PFB, CBCPD, OFBNLF, BCF, BOF]

4 Detalles

El núcleo principal del proyecto es una librería llamada "JCEF", la cual proporciona la interfaz necesaria para poder manejar algoritmos criptográficos y objetos seguros. El diagrama general de clases de dicha librería es el siguiente:

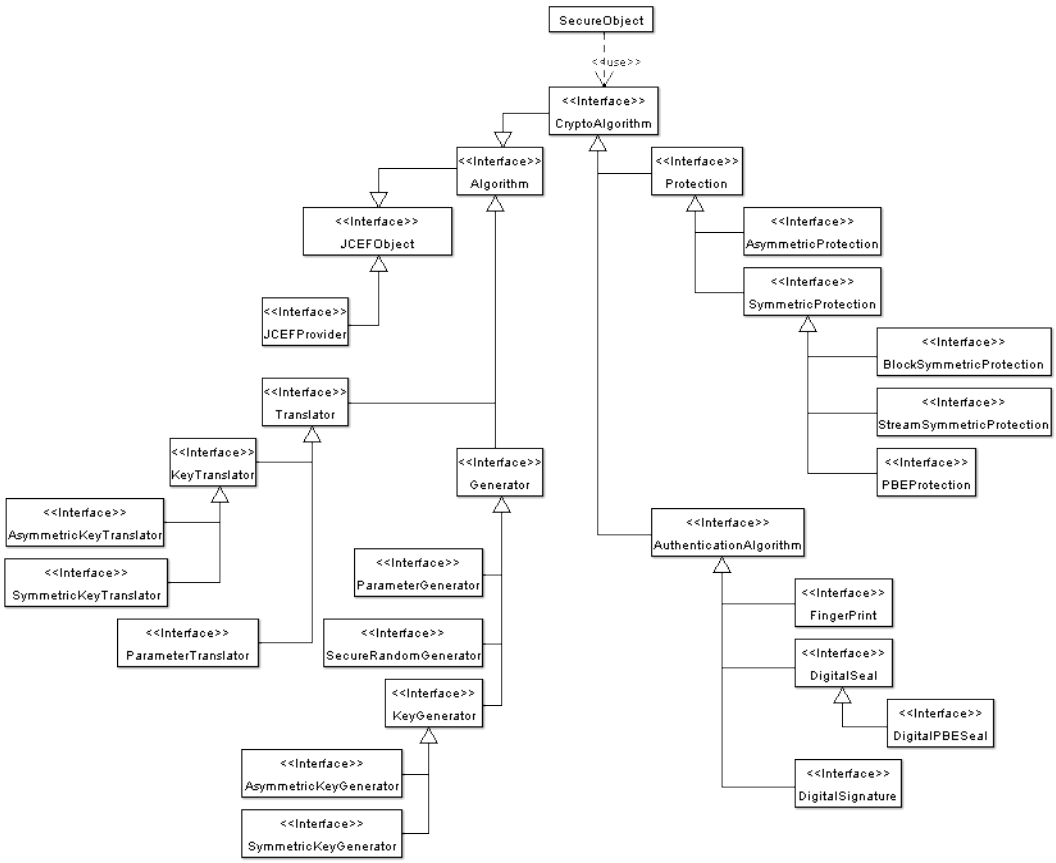

Ilustración 15: Diagrama general de clases de JCEF

La clase "SecureObject" es la clase que representa a los objetos seguros y éstos son generados utilizando algoritmos criptográficos que son representados mediante la interfaz "CryptoAlgorithm". Para obtener nuevamente los objetos asegurados es necesario volver a utilizar los algoritmos criptográficos.

Es tal la dimensión del proyecto, que de ahí el gran número de librerías debido principalmente a que se ha querido proporcionar el mayor número de proveedores y algoritmos criptográficos posible realizando adaptaciones de proveedores y algoritmos criptográficos ya existentes que cumplan con la especificación JCE u otra.

Para que el lector puede hacerse una idea de la dimensión de este proyecto, basta con decir que este proyecto contiene 3142 clases, 315 campos, 2914 métodos, 73959 líneas de código y 22941 líneas de comentarios. Dentro de estos datos ya se encuentran incluidos las métricas para las pruebas, cuyas dimensiones concretas son 1724 clases, 73 campos, 958 métodos, 30936 líneas de código y 8119 líneas de comentarios. Además, también se incluyen las métricas para el manual de usuario de JCEF, cuyos datos son 11 clases, 3 campos, 412 métodos, 4572 líneas de código y 5740 líneas de comentarios.

Otra métrica para imaginarse el coste de este proyecto es el tiempo de desarrollo y realización del mismo; el cual se estima entre un mínimo de 2000 horas de trabajo y un máximo de 3000 horas. Este proyecto ha tenido pocas pausas durante su desarrollo, el cual comenzó a finales de febrero de 2005 y terminó a finales de mayo de 2006, es decir, más de un año de desarrollo.

El manual de usuario de este proyecto, sin considerar este documento, se encuentra en una clase llamada "UserGuide". Se encuentra en una clase ya que no hay mejor manual para una librería de programación que una clase compuesta por numerosos ejemplos escritos en el mismo lenguaje de programación (Java) y explicaciones escritas con el lenguaje de documentación para código (Javadoc). Este manual de usuario podrá encontrarlo fácilmente en la dirección web http://jcef.sourceforge.net/api/org/rrexky/security/crypto/jcef/UserGuide.html. Para obtener más información es importante consultar la página web de este proyecto http://jcef.sourceforge.net y la página principal online de la documentación de JCEF que se encuentra en http://jcef.sourceforge.net/api/index.html.

A continuación se mostrarán los siguientes apartados:

- Visión general del diseño, donde se hablará del diseño de este proyecto.
- Componentes. Dicha sección enumerará los partes de las que se compone el proyecto.
- Dependencias, donde se mostrarán las dependencias del proyecto y de sus componentes.
- Distribuciones que son conjuntos de ficheros que contienen diferentes construcciones del proyecto. Unas destinadas para el uso general, otra para desarrolladores, etc...
- También se describirán las pruebas realizadas al proyecto, el contenido del CD-ROM que se adjunta con la memoria de este proyecto y una pequeña guía de instalación y uso del proyecto.

1 Visión general del diseño

El proyecto "Aplicaciones Criptográficas Java" ha sido descompuesto en numerosas librerías. Además, cada librería tiene asociada otra que contiene las pruebas para comprobar el correcto funcionamiento de dicha librería. Por si fuera poco, todas las librerías han sido documentadas.

Las librerías de este proyecto son "JCEF" como librería principal, "JCEF Addons" como librería para extensiones futuras y el resto de librerías son proveedores criptográficos que cumplen al menos la especificación JCEF y librerías de pruebas para comprobar el correcto funcionamiento de todas las librerías. Concretamente se prueba el correcto funcionamiento de todos los métodos de todas las clases que componen el proyecto y se realizan pruebas de funcionalidad a cada algoritmo.

El diseño de este proyecto sigue los principios básicos de los Java Beans. Es decir, todas las clases se componen de atributos a los cuales se accede mediante al menos un método de lectura (llamado "get + nombre del atributo") y al menos otro de escritura (llamado "set + nombre del atributo").

Otra consideración a tener en cuenta es que se ha intentado al máximo que el número de líneas de código por método fuera el menor posible y así de esta forma conseguir que:

- No sea necesario escribir grandes cantidades de documentación ya que los métodos casi se explican por sí solos mediante el código, necesitando tan solo la ayuda de unas pocas líneas de comentarios para la documentación del mismo.
- La realización de las pruebas y su ejecución fuera lo más sencillo posible.

2 Componentes

Este proyecto se compone de las siguientes librerías:

1. JCEF: Esta es la librería principal del proyecto, donde se encuentra la especificación JCEF, las interfaces para manipular algoritmos criptográficos y otros algoritmos de apoyo relacionados con la criptografía. Además también tiene la posibilidad de utilizar objetos seguros. Por si no fuera poco, también contiene varias implementaciones de la especificación o interfaz JCEF para implementar rápidamente nuevos algoritmos criptográficos, adaptar otros ya existentes y además ser compatible con la anterior especificación JCE. Su página web oficial y la de este proyecto es http://jcef.sourceforge.net.
2. JCEF Tests: Esta librería es la encargada de realizar pruebas a la librería "JCEF" para comprobar su correcto funcionamiento.
3. RREXKY JCEF Provider: Principal proveedor de algoritmos criptográficos que cumplen la especificación JCEF. Todos estos algoritmos han sido probados y funcionan correctamente. Se trata de un proveedor personalizado con algoritmos criptográficos extraidos de otros proveedores tales como "BouncyCastle JCEF Provider", "Jacksum JCEF

Provider" y "FlexiCore JCEF Provider". Este proveedor contiene 64 algoritmos de todos los tipos funcionando correctamente y son:

- *Algoritmos criptográficos de protección asimétrica*: RSA
- *Algoritmos criptográficos de protección simétrica de bloques*: AES, Blowfish, Camellia, CAST5, CAST6, GOST, RC2, RC6, Serpent, Skipjack, TripleDES, Twofish.
- *Algoritmos criptográficos de protección simétrica de flujo*: RC4
- *Algoritmos criptográficos de protección basada en contraseña*: PBEwithSHA1andTwofish, PBEwithSHA256andAES256
- *Algoritmos generadores de fuentes de datos aleatorios*: BBS
- *Algoritmos criptográficos de autentificación mediante huellas digitales*: GOST, Haval128, Haval160, Haval192, Haval224, Haval256, MD2, MD4, MD5, RIPEMD128, RIPEMD160, RIPEMD256, RIPEMD320, SHA1, SHA224, SHA256, SHA384, SHA512, Tiger, Whirlpool
- *Algoritmos criptográficos de autentificación mediante sellos digitales*: HmacMD2, HmacMD4, HmacMD5, HmacRIPEMD128, HmacRIPEMD160, HmacSHA1, HmacSHA224, HmacSHA256, HmacSHA384, HmacSHA512, HmacTiger, MacDES, MacRC2, MacSkipjack, MacTripleDES
- *Algoritmos criptográficos de autentificación mediante sellos basados en contraseña*: PBEHmacRIPEMD160, PBEHmacSHA1
- *Algoritmos criptográficos de autentificación mediante firmas digitales*: MD2withRSA, MD4withRSA, MD5withRSA, RIPEMD128withRSA, RIPEMD160withRSA, RIPEMD256withRSA, SHA1withRSA, SHA224withRSA, SHA256withRSA, SHA384withRSA, SHA512withRSA

4. RREXKY JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "RREXKY JCEF Provider" para comprobar su correcto funcionamiento.

5. JCEF Addons: Extensiones para JCEF. Actualmente incluye, aunque no se han probado correctamente, clases que permiten definir nuevos algoritmos criptográficos en base a otros algoritmos criptográficos ya existentes. También contiene las pruebas para poder realizarlas con el objetivo de comprobar que las implementaciones de los algoritmos criptográficos son correctas.

6. JCEF Addons Tests: Esta librería es la encargada de realizar pruebas a la librería "JCEF Addons" para comprobar su correcto funcionamiento.

7. BouncyCastle JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://www.bouncycastle.org. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

8. BouncyCastle JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "BouncyCastle JCEF Provider" para comprobar su correcto funcionamiento.

9. Jacksum JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente que no cumple ni la especificación JCE ni la JCEF cuya

web es http://www.jonelo.de/java/jacksum/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

10. Jacksum JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "Jacksum JCEF Provider" para comprobar su correcto funcionamiento.

11. FlexiCore JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://www.flexiprovider.de/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

12. FlexiCore JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "FlexiCore JCEF Provider" para comprobar su correcto funcionamiento.

13. FlexiEC JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://www.flexiprovider.de/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

14. FlexiEC JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "FlexiEC JCEF Provider" para comprobar su correcto funcionamiento.

15. FlexiNF JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://www.flexiprovider.de/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

16. FlexiNF JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "FlexiNF JCEF Provider" para comprobar su correcto funcionamiento.

17. CryptixCrypto JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://www.cryptix.org/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

18. CryptixCrypto JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "CryptixCrypto JCEF Provider" para comprobar su correcto funcionamiento.

19. Cryptix JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://www.cryptix.org/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

20. Cryptix JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "Cryptix JCEF Provider" para comprobar su correcto funcionamiento.

21. GNUCrypto JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://www.gnu.org/software/gnu-crypto/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

22. GNUCrypto JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "GNUCrypto JCEF Provider" para comprobar su correcto funcionamiento.

23. IAIK JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://jce.iaik.tugraz.at/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

24. IAIK JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "IAIK JCEF Provider" para comprobar su correcto funcionamiento.

25. JHBCI JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://www.jhbci.de/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

26. JHBCI JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "JHBCI JCEF Provider" para comprobar su correcto funcionamiento.

27. Logi Crypto JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero una especificación desconocida que no es JCE cuya web es http://www.logi.org/logi.crypto/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

28. Logi Crypto JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "Logi Crypto JCEF Provider" para comprobar su correcto funcionamiento.

29. Mindbright JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de una especificación desconocida que no es JCE cuya web es http://www.appgate.com/products/80_MindTerm/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

30. Mindbright JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "Mindbright JCEF Provider" para comprobar su correcto funcionamiento.

31. SUN JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://java.sun.com/j2se/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

32. SUN JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "SUN JCEF Provider" para comprobar su correcto funcionamiento.

33. SunJCE JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://java.sun.com/j2se/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

34. SunJCE JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "SunJCE JCEF Provider" para comprobar su correcto funcionamiento.

35. SunJSSE JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://java.sun.com/j2se/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

36. SunJSSE JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "SunJSSE JCEF Provider" para comprobar su correcto funcionamiento.

37. SunRsaSign JCEF Provider: Se trata de un proveedor JCEF que ha sido adaptado de otro proveedor criptográfico ya existente pero de especificación JCE cuya web es http://java.sun.com/j2se/. Este proveedor ha sido casi completamente desarrollado pero no del todo probado. Funcionan muchos algoritmos y el resto necesitan configurarse mejor.

38. SunRsaSign JCEF Provider Tests: Esta librería es la encargada de realizar pruebas a la librería "SunRsaSign JCEF Provider" para comprobar su correcto funcionamiento.

3 Dependencias

En esta sección se indican las librerías de las cuales este proyecto depende. En la siguiente lista se indicarán las dependencias de cada una de las librerías de este proyecto:

- JCEF: Depende de "API de J2SE" y "Apache Jakarta Commons Lang".
- JCEF Tests: Depende de "API de J2SE", "JUnit", "JCEF" y "JCEF Addons".
- RREXKY JCEF Provider: .Depende de "API de J2SE", "BouncyCastle JCEF Provider", "Jacksum JCEF Provider" y "FlexiCore JCEF Provider".
- RREXKY JCEF Provider Tests: Depende de "API de J2SE", "JUnit", "JCEF Tests" y "RREXKY JCEF Provider".
- JCEF Addons: Depende de "API de J2SE", "JCEF" y "JCEF Tests".
- JCEF Addons Tests: Depende de "API de J2SE", "JCEF Addons" y "JCEF Tests".
- BouncyCastle JCEF Provider: Depende de "API de J2SE", "JCEF" y "BouncyCastle JCE Provider".
- BouncyCastle JCEF Provider Tests: Depende de "API de J2SE", "JUnit", "JCEF Tests" y "BouncyCastle JCEF Provider".
- Jacksum JCEF Provider: Depende de "API de J2SE", "JCEF" y "Jacksum NonJCE Provider".
- Jacksum JCEF Provider Tests: Depende de "API de J2SE", "JUnit", "JCEF Tests" y "Jacksum JCEF Provider".
- FlexiCore JCEF Provider: Depende de "API de J2SE", "JCEF" y "FlexiCore JCE Provider".
- FlexiCore JCEF Provider Tests: Depende de "API de J2SE", "JUnit", "JCEF Tests" y "FlexiCore JCEF Provider".
- FlexiEC JCEF Provider: Depende de "API de J2SE", "JCEF" y "FlexiEC JCE Provider".
- FlexiEC JCEF Provider Tests: Depende de "API de J2SE", "JUnit", "JCEF Tests" y "FlexiEC JCEF Provider".

- FlexiNF JCEF Provider: Depende de “API de J2SE”, “JCEF” y “FlexiNF JCE Provider”.
- FlexiNF JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “FlexiNF JCEF Provider”.
- CryptixCrypto JCEF Provider: Depende de “API de J2SE”, “JCEF” y “CryptixCrypto JCE Provider”.
- CryptixCrypto JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “CryptixCrypto JCEF Provider”.
- Cryptix JCEF Provider: Depende de “API de J2SE”, “JCEF” y “Cryptix JCE Provider”.
- Cryptix JCEF Provider Tests: Depende de “J2SE”, “JUnit”, “JCEF Tests” y “Cryptix JCEF Provider”.
- GNUCrypto JCEF Provider: Depende de “API de J2SE”, “JCEF” y “GNUCrypto JCE Provider”.
- GNUCrypto JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “GNUCrypto JCEF Provider”.
- IAIK JCEF Provider: Depende de “API de J2SE”, “JCEF” y “IAIK JCE Provider”.
- IAIK JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “IAIK JCEF Provider”.
- JHBCI JCEF Provider: Depende de “API de J2SE”, “JCEF” y “JHBCI JCE Provider”.
- JHBCI JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “JHBCI JCEF Provider”.
- Logi Crypto JCEF Provider: Depende de “API de J2SE”, “JCEF” y “Logi Crypto NonJCE Provider”.
- Logi Crypto JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “Logi Crypto JCEF Provider”.
- Mindbright JCEF Provider: Depende de “API de J2SE”, “JCEF” y “Mindbright NonJCE Provider”.
- Mindbright JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “Mindbright JCEF Provider”.
- SUN JCEF Provider: Depende de “API de J2SE” y “JCEF”.
- SUN JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “SUN JCEF Provider”.
- SunJCE JCEF Provider: Depende de “API de J2SE” y “JCEF”.
- SunJCE JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “SunJCE JCEF Provider”.

- SunJSSE JCEF Provider: Depende de “API de J2SE” y “JCEF”.
- SunJSSE JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “SunJSSE JCEF Provider”.
- SunRsaSign JCEF Provider: Depende de “API de J2SE” y “JCEF”.
- SunRsaSign JCEF Provider Tests: Depende de “API de J2SE”, “JUnit”, “JCEF Tests” y “SunRsaSign JCEF Provider”.

4 Distribuciones

Aquí se describen las distribuciones que proporciona este proyecto desde su web http://jcef.sourceforge.net y http://sourceforge.net/projects/jcef:

- “jcef_v1.0_basicruntime.zip”: Es la distribución básica. Contiene las librerías para el uso de JCEF y del proveedor RREXKY tan solo. Se trata de una distribución sólo para usuarios que necesitan lo básico. Ocupa unos 2'54 MB y se distribuye en formato ZIP. Esta distribución viene acompañada de las librerías de las que depende excepto de la “API de J2SE” que es la estándar y no necesita incluirse.
- “jcef_v1.0_development.zip”: Es la distribución destinada a los desarrolladores que quieran mejorar este proyecto. Contiene el código fuente, las pruebas y la documentación tanto de “JCEF”, “JCEF Addons” y todos los proveedores criptográficos “JCEF”. Ocupa unos 24'3 MB y se distribuye en formato ZIP. Esta distribución viene acompañada de las librerías de las que depende excepto de la “API de J2SE” que es la estándar y no necesita incluirse.
- “jcef_v1.0_runtime.zip”: Es la distribución completa para usuarios, no desarrolladores. Contiene únicamente todo el código compilado de “JCEF”, “JCEF Addons”, las pruebas y los proveedores. Ocupa unos 6'64 MB y se distribuye en formato ZIP. Esta distribución viene acompañada de las librerías de las que depende excepto de la “API de J2SE” que es la estándar y no necesita incluirse.
- “jcef_v1.0_all_src.zip”: Contiene todo el código fuente Java del proyecto, incluyendo obviamente los comentarios de documentación Javadoc. Ocupa unos 7'18 MB y se distribuye en formato ZIP. Esta distribución viene acompañada de las librerías de las que depende excepto de la “API de J2SE” que es la estándar y no necesita incluirse.
- “jcef_v1.0_doc.zip”: Contiene toda la documentación de la memoria y la documentación de todas las librerías del proyecto. Ocupa unos 22'4 MB y se distribuye en formato ZIP.
- “jcef_v1.0_web.zip”: Distribución compuesta por la página web del proyecto, pero sin incluir el resto de distribuciones. Ocupa unos 9'51 MB y se distribuye en formato ZIP.

5 Página web

Desde la página web del proyecto http://jcef.sourceforge.net (o también http://sourceforge.net/projects/jcef/) el visitante podrá acceder a los siguientes recursos:

- Documentación del proyecto, desde un sumario hasta la documentación completa. Podrá acceder además a los capítulos por separado de la documentación completa. Estos capítulos son: "Introducción a la seguridad", "Criptografía orientada a objetos", "El proyecto", "Futuros proyectos", "Recursos utilizados" y "Preguntas frecuentes".
- Documentación de la API JCEF. Tan solo se ha incluido la documentación de la librería "JCEF" y se han dejado fuera el resto de librerías por cuestiones de espacio en el servidor de alojamiento de la página web del proyecto ya que éste sólo permite un máximo de 100 MB y la documentación completa de todas las librerías del proyecto ocupa más de 150 MB.
- Zona de descarga de las distribuciones.
- Presentación. Se podrá acceder a una presentación de diapositivas para ver lo más interesante del proyecto y aquello que está sujeto de ser visionado en la lectura de este proyecto ante el tribunal evaluador del mismo.
- Multitud de enlaces de herramientas utilizadas para el desarrollo del proyecto.
- Últimas noticias sobre el proyecto.

6 Pruebas realizadas

Las librerías de pruebas se encargan de someter al proyecto a una serie de tests con el objetivo de comprobar el correcto funcionamiento del mismo.

Se han realizado pruebas para comprobar el correcto funcionamiento de todos los métodos de todas las clases que componen el proyecto.

La mayoría de las pruebas se basan en comprobar que las propiedades de los objetos de una clase se leen y escriben correctamente.

Existen otras pruebas que verifican la funcionalidad de los algoritmos. Esta prueba genera párametros y claves, realiza traducciones de parámetros y claves y especialmente trata de generar objetos seguros para posteriormente obtener el objeto asegurado con varios tipos de parámetros y claves. Si el objeto asegurado es el mismo que el original o si es auténtico, entonces el algoritmo supera la prueba de funcionalidad. Un algoritmo no supera esta prueba de funcionalidad en caso contrario o cuando se produce alguna excepción provocada principalmente por una mala configuración del algoritmo.

Hay otras pruebas llamadas de implementación. Su finalidad es la de verificar la correcta implementación de los algoritmos, no se trata de que funcionen, si no de que funcionen como tienen que hacerlo. Pero hay que decir que estas pruebas de implementación están implementadas pero no se ha realizado ninguna ya que esto se ha considerado parte de un proyecto futuro.

Otra curiosidad es que son tantas pruebas que se tarda bastante en poder comprobar si éstas son superadas. También es conveniente saber que si se lanzan muchas pruebas a la vez, se corre el riesgo de quedarse sin memoria y no poder ver si son superadas o no. Lo mejor que se puede hacer es lanzar las pruebas sobre los paquetes hoja (aquellos que no contienen otros paquetes).

Recuerde también que las únicas librerías que han superado completamente las pruebas, son "JCEF" y "RREXKY JCEF Provider". El resto de librerías requieren una revisión "clase por clase", "algoritmo por algoritmo". Esto último no se ha realizado por falta de tiempo y se ha dejado la responsabilidad de ello a un proyecto futuro.

7 Contenido del CD-ROM

El CD-ROM que acompaña a la memoria de este proyecto contendrá las siguientes carpetas:

- "Source Code": Todo el código fuente del proyecto.
- "Libs": Todas las librerías de las que depende este proyecto exceptuando la librería estándar J2SE.
- "Distributions": Almacenará las distribuciones del proyecto.
- "Papers": La memoria del proyecto en formato OpenOffice y PDF junto con las imágenes utilizadas. Además se incluye la documentación API sobre todas las librerías.
- "Web": Página web del proyecto, a la que se podrá acceder de forma offline u online si se desea.
- "Software": Contiene el software mínimo necesario para poder consultar el código fuente del proyecto y poder lanzar las pruebas para comprobar su correcto funcionamiento:
  - "JRE.exe": Archivo de instalación de Java 2 Runtime Standard Edition (J2SE).
  - "jce policy files": Carpeta que contiene los archivos que permiten desbloquear JRE/J2SE.
  - "Eclipse": Carpeta que contiene el IDE para Java llamado "Eclipse". Para instalarlo, basta con copiar esta carpeta al disco duro.
  - "7-Zip.exe": Archivo de instalación del programa "7-Zip", el cual se utiliza para comprimir/descomprimir archivos.
  - "AlZip.exe": Archivo de instalación del programa "AlZip" utilizado para comprimir/descomprimir archivos.
  - "Adobe Reader ES.exe": Archivo de instalación del programa "Adobe Reader" en español para Windows XP. Permite leer archivos PDF.

- "OpenOffice 2.exe": Archivo de instalación del programa "OpenOffice" que permitirá ver y sobre todo editar la memoria del proyecto.
- "OpenOffice 2 ES pack.exe": Archivo de instalación para traducir OpenOffice a español.

8 Guía de instalación y uso

Si lo que desea es ver qué se ha hecho exactamente, examinar el código fuente y comprobar que las pruebas son superadas, sólo es necesario que siga los siguientes pasos:

1. Instalar la máquina virtual de Java: "J2SE Runtime Environment" (JRE) en un directorio cualquiera que llamaremos simbólicamente <java-home>.
2. Desproteger JRE copiando los ficheros de políticas de seguridad llamados "US_export_policy.jar" y "local_policy.jar" al directorio "<java-home>/lib/security".
3. Copiar todo el código fuente del proyecto en una carpeta llamada <source-home>.
4. Copiar todas las librerías de las que depende este proyecto en una carpeta llamada <libs-home>.
5. Instalar el IDE "Eclipse", abrirlo, definirle un directorio cualquiera para el espacio de trabajo y cerrar la pestaña "Welcome" si le apareciera.
6. Crear un nuevo proyecto Java llamado "Aplicaciones Criptográficas Java" a partir de la carpeta <source-home> y a continuación pulsar en siguiente.
7. Añadir el JRE desprotegido como librería que está ubicado en <java-home>.
8. Añadir las librerías ubicadas en <libs-home> bajo el nombre de una única librería que podría llamarse "Dependences". Es muy importante asegurarse de que la librería JRE se encuentra encima de la librería "Dependences".
9. Esperar un tiempo a que se construya el espacio de trabajo.
10. Y ya está en disposición de ver el código fuente y verificar que las pruebas se superan al menos para las librerías "JCEF" y "RREXKY JCEF Provider".
11. Para ver el código, simplemente haga doble click en el nodo raíz del proyecto, en nodos librería, nodos carpeta y nodos archivo Java.
12. Para lanzar las pruebas sobre "JCEF", debe dirigirse a la librería "JCEF Tests" e ir ejecutando cada una de las pruebas, desde pruebas sencillas (ficheros cuyo nombre termina en "Test") hasta múltiples pruebas (ficheros cuyos nombres comienzan con "AllTests"). Para lanzar pruebas basta con pulsar el botón derecho sobre ella y hacer click en la opción "Run As --> JUnit Test". Recuerde que es importante que no ejecute muchas pruebas a la vez puesto que puede que se quede sin memoria y por ello no terminen correctamente las pruebas. Lo mejor es lanzar las pruebas de los paquetes hoja por cada librería. También puede comprobar el correcto funcionamiento de la librería "RREXKY JCEF Provider".

5 El uso habitual

En esta sección se muestra el uso más habitual de la criptografía, es decir, se muestra cómo se crean objetos seguros y se obtienen sus objetos asegurados a partir de sus versiones seguras utilizando parámetros nuevos y reutilizando unos ya existentes generados con anteriodad.

En los siguientes puntos se mostrará el mencionado uso habitual mediante la utilización de este proyecto. Se verá que es muy fácil construir objetos seguros con la librería JCEF. En resumen, este uso habitual se compone de los siguientes pasos:

1. Asegurar un objeto con nuevos parámetros criptográficos.
2. Almacenar parámetros criptográficos para un uso posterior.
3. Obtener el objeto asegurado utilizando los nuevos parámetros criptográficos.
4. Asegurar otro objeto reutilizando parámetros criptográficos ya existentes.
5. Obtener el objeto asegurado reutilizando parámetros criptográficos ya existentes.

1 Asegurar un objeto con nuevos parámetros criptográficos

```
/**
 * Mediante JCEF, crea un objeto seguro y obtiene su versión insegura
nuevamente. Posteriormente crea y deshace otro objeto seguro reutilizando
parámetros generados con anterioridad
 */
public static void example1_JCEF () {
 try {
  // 1. Se desea asegurar un objeto
  // 1.1. Definimos un objeto que se desea asegurar
  String object = "my object";
  // 1.2. Se selecciona un algoritmo de seguridad
          SymmetricProtection         secureAlgorithm         =         new
AES_BlockSymmetricProtectionRREXKY();
   // 1.3. Se inicializa el algoritmo con parámetros concretos si fuera
necesario (esta inicialización es opcional)
   // Esta inicialización es inexistente ya que se desea utilizar unos
parámetros nuevos (clave y parámetro generado automáticamente por el
algoritmo)
  // 1.4. Se asegura el objeto y se guarda en algún lugar
  SecureObject secureObject = new SecureObject(object, secureAlgorithm);
  saveObject(secureObject);
  // ...
```

Tabla 36: JCEF – 1. Asegurar un objeto con nuevos parámetros criptográficos

2 Almacenar parámetros criptográficos para un uso posterior

```
  // 2. Se guardan los parámetros para su posterior uso, ya sea para generar nuevos objetos seguros u obtener objetos asegurados (objetos inseguros) que es lo más habitual
  byte[] encodedKey = secureAlgorithm.getSymmetricKey().getEncoded();
  byte[] encodedParameter = secureAlgorithm.getEncodedParameter();
  saveKey(encodedKey); saveParameter(encodedParameter);
  // ...
```

Tabla 37: JCEF – 2. Almacenar parámetros criptográficos para un uso posterior

3 Obtener el objeto asegurado utilizando los nuevos parámetros criptográficos

```
  // ... tiempo más tarde en algún otro lugar del código ...
  // 3. Se desea recuperar el objeto de forma segura
  // 3.1. Cargamos los datos que se necesitan
  encodedKey = loadKey(); encodedParameter = loadParameter();
  // 3.2. Inicializamos el algoritmo con los parámetros apropiados para poder obtener el objeto asegurado
  secureAlgorithm = new AES_BlockSymmetricProtectionRREXKY();
  secureAlgorithm.setSymmetricKey(encodedKey);
  secureAlgorithm.setParameter(encodedParameter);
  // 3.3. Se obtiene el objeto asegurado a partir de su versión en forma de objeto seguro
  secureObject = (SecureObject)loadObject();
  object = (String)secureObject.getObject(secureAlgorithm);
  // ...
```

Tabla 38: JCEF – 3. Obtener el objeto asegurado utilizando los nuevos parámetros criptográficos

4 Asegurar otro objeto reutilizando parámetros criptográficos ya existentes

```
  // ... tiempo más tarde en algún otro lugar del código ...
  // 4. Se desea asegurar un nuevo objeto con la misma clave y parámetro en
un instante de tiempo posterior
  // 4.1. Se define el nuevo objeto
  String otherObject = "other object";
  // 4.2. Se cargan los parámetros que se necesitan
  encodedKey = loadKey(); encodedParameter = loadParameter();
  // 4.3. Se inicializa el algoritmo
  secureAlgorithm = new AES_BlockSymmetricProtectionRREXKY();
  secureAlgorithm.setSymmetricKey(encodedKey);
  secureAlgorithm.setParameter(encodedParameter);
  // 4.4. Se asegura el nuevo objeto
    SecureObject otherSecureObject = new SecureObject(otherObject,
secureAlgorithm);
  saveObject(otherSecureObject);
  // ...
```

Tabla 39: JCEF – 4. Asegurar otro objeto reutilizando parámetros criptográficos ya existentes

5 Obtener el objeto asegurado reutilizando parámetros criptográficos ya existentes

```
  // ... tiempo más tarde en algún otro lugar del código ...
  // 5. Se desea recuperar el nuevo objeto asegurado
  // 5.1. Cargamos los datos que se necesitan
  encodedKey = loadKey(); encodedParameter = loadParameter();
  // 5.2. Inicializamos el algoritmo con los parámetros apropiados para
poder obtener el objeto asegurado
  secureAlgorithm = new AES_BlockSymmetricProtectionRREXKY();
  secureAlgorithm.setSymmetricKey(encodedKey);
  secureAlgorithm.setParameter(encodedParameter);
  // 5.3. Se obtiene el objeto asegurado a partir de su versión en forma de
objeto seguro
  otherSecureObject = (SecureObject)loadObject();
  object = (String)otherSecureObject.getObject(secureAlgorithm);
 }
 catch (Throwable throwable) { throwable.printStackTrace(); return; }
}
```

Tabla 40: JCEF – 5. Obtener el objeto asegurado reutilizando parámetros criptográficos ya existentes

6 Comparativa con JCA y JCE

A continuación se podrá ver el uso más habitual de la criptografía, es decir, se mostrará cómo se crean unos objetos seguros y se obtienen sus objetos asegurados a partir de sus versiones seguras utilizando parámetros nuevos y reutilizando unos ya existentes generados con anteriodad. Pero en esta ocasión, no se utilizará el proyecto; se empleará la arquitectura criptográfica de Java (JCA) y su extensión (JCE). De esta forma, podrá verse una parte importante del valor añadido de este proyecto al mundo de la criptografía y Java.

1 Asegurar un objeto con nuevos parámetros criptográficos

- *Definición del objeto a asegurar y carga del proveedor*

```
/**
 * Mediante JCE, crea un objeto seguro y obtiene su versión insegura nuevamente. Posteriormente crea y deshace otro objeto seguro reutilizando parámetros generados con anterioridad
 */
public static void example1_JCE () {
 // 1. Se desea asegurar un objeto
 // 1.1. Definimos un objeto que se desea asegurar
 String object = "my object";
 // 1.2. Se selecciona un algoritmo de seguridad
 // 1.2.1. Es muy posible que sea necesario cargar el proveedor de los algoritmos que se desean utilizar. Esto hay que hacerlo una sola vez
 Provider provider = new BouncyCastleProvider();
 Security.addProvider(provider);
 String providerName = provider.getName();
 // ...
```

Tabla 41: JCE – 1.1. Definición del objeto a asegurar y carga del proveedor

- *Definición del generador de claves simétricas*

```
 // ...
// 1.2.2. Se deben generar los parámetros del algoritmo antes de seleccionar el mismo
// 1.2.2.1. Se genera la fuente de datos aleatorios si fuera necesario (no es el caso)
SecureRandom random = null;
// 1.2.2.2. Se genera la clave
Key key = null;
// 1.2.2.2.1. Se selecciona el algoritmo generador de claves simétricas para el algoritmo deseado
javax.crypto.KeyGenerator keyGenerator = null;
try { keyGenerator = javax.crypto.KeyGenerator.getInstance("AES", providerName); }
catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
catch (NoSuchProviderException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 42: JCE – 1.2. Definición del generador de claves simétricas

- *Inicialización del generador de claves simétricas y generación de la clave*

```
 // ...
 // 1.2.2.2.2. Se inicializa el algoritmo de generación de claves simétricas para el algoritmo deseado
int keySize = 256;
AlgorithmParameterSpec genParameter = null;
if (genParameter != null && random == null) {
 try { keyGenerator.init(genParameter); }
catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
} else if (genParameter != null && random != null) {
 try { keyGenerator.init(genParameter, random); }
catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
} else if (genParameter == null && keySize > 0 && random == null) {
 keyGenerator.init(keySize);
} else if (genParameter == null && keySize > 0 && random != null) {
 keyGenerator.init(keySize, random);
} else if (genParameter == null && keySize <= 0 && random != null) {
 keyGenerator.init(random);
}
// 1.2.2.2.3. Se utiliza el generador ya seleccionado e inicializado para generar la clave simétrica
key = keyGenerator.generateKey();
 // ...
```

Tabla 43: JCE – 1.3. Inicialización del generador de claves simétricas y generación de la clave

- *Definición del generador de parámetros*

```
 // ...
 // 1.2.2.3. Se necesita generar además un parámetro para el algoritmo además de la clave generada con anterioridad
 AlgorithmParameterSpec parameter = null;
 // 1.2.2.3.1. Se indica el tipo de parámetro del que se trata
 Class parameterType = IvParameterSpec.class;
 // 1.2.2.3.2. Se selecciona el algoritmo generador de parámetros para el algoritmo deseado
 AlgorithmParameterGenerator parameterGenerator = null;
 try { parameterGenerator = AlgorithmParameterGenerator.getInstance("AES", providerName); }
 catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
 catch (NoSuchProviderException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 44: JCE – 1.4. Definición del generador de parámetros

- *Inicialización del generador de parámetros*

```
 // ...
 // 1.2.2.3.3. Se inicializa el algoritmo de generación de parámetros
 genParameter = null; int parameterSize = 16;
 if (genParameter != null && random == null) {
  try { parameterGenerator.init(genParameter); }
 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
 } else if (genParameter != null && random != null) {
  try { parameterGenerator.init(genParameter, random); }
 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
 } else if (genParameter == null && random == null) {
  parameterGenerator.init(parameterSize);
 } else if (genParameter == null && random != null) {
  parameterGenerator.init(parameterSize, random);
 }
 // ...
```

Tabla 45: JCE – 1.5. Inicialización del generador de parámetros

- *Generación del parámetro*

```
 // ...
// 1.2.2.3.4. Se utiliza el generador para obtener el parámetro necesario para el algoritmo
AlgorithmParameters algorithmParameters = parameterGenerator.generateParameters();
try { parameter = algorithmParameters.getParameterSpec(parameterType); }
catch (InvalidParameterSpecException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 46: JCE – 1.6. Generación del parámetro

- *Definición del algoritmo de seguridad*

```
 // ...
// 1.2.3. Se selecciona el algoritmo de seguridad
Cipher secureAlgorithm = null;
try { secureAlgorithm = Cipher.getInstance("AES/CBC/PKCS7Padding", providerName); }
catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
catch (NoSuchProviderException e) { e.printStackTrace(); return; }
catch (NoSuchPaddingException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 47: JCE – 1.7. Definición del algoritmo de seguridad

- *Inicialización del algoritmo de seguridad*

```
 // ...
 // 1.3. Se inicializa el algoritmo con parámetros concretos si fuera necesario (esta inicialización NO es opcional)
 // 1.3.1. Se inicializa el algoritmo
 int mode = Cipher.ENCRYPT_MODE;
 if (parameter != null && random == null) {
  try { secureAlgorithm.init(mode, key, parameter); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
 } else if (parameter != null && random != null) {
  try { secureAlgorithm.init(mode, key, parameter, random); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
 } else if (parameter == null && random == null) {
  try { secureAlgorithm.init(mode, key); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
 } else if (parameter == null && random != null) {
  try { secureAlgorithm.init(mode, key, random); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
 }
  // ...
```

Tabla 48: JCE – 1.8. Inicialización del algoritmo de seguridad

- *Obtención del parámetro que se haya podido generar automáticamente*

```
 // ...
 // 1.3.2. Se capturan los parámetros si se han generado automáticamente. Esto es necesario si no se especifican los algoritmos, e incluso es necesario aunque se especifique puesto que el algoritmo puede ignorar el parámetro indicado por el usuario y generar otro de forma automática
 try { algorithmParameters = secureAlgorithm.getParameters(); }
 catch (Throwable throwable) {}
 if (parameter == null && algorithmParameters != null) {
  if (parameterType == null) {
   parameterType = AlgorithmParameterSpec.class;
  }
  try { parameter = algorithmParameters.getParameterSpec(parameterType); }
  catch (InvalidParameterSpecException e) { e.printStackTrace(); return; }
 }
  // ...
```

Tabla 49: JCE – 1.9. Obtención del parámetro que se haya podido generar automáticamente

- *Creación del objeto seguro*

```
 // ...
// 1.4. Se asegura el objeto y se guarda en algún lugar
SealedObject secureObject = null;
try { secureObject = new SealedObject(object, secureAlgorithm); }
catch (IllegalBlockSizeException e) { e.printStackTrace(); return; }
catch (IOException e) { e.printStackTrace(); return; }
saveObject(secureObject);
 // ...
```

Tabla 50: JCE – 1.10. Creación del objeto seguro

2 Almacenar parámetros criptográficos para un uso posterior

- *Traducción de la clave*

```
 // ...
 // 2. Se guardan los parámetros para su posterior uso, ya sea para generar nuevos objetos seguros u obtener objetos asegurados (objetos inseguros) que es lo más habitual
 // 2.1. Se traducen los parámetros a su versión persistente
 // 2.1.1. Se traduce la clave a su versión persistente. En este caso es sencillo. Pero en otras ocasiones es distinto y en otras es necesario conocer el algoritmo de traducción dificultando el proceso mucho más al no ser homogéneo ya que también se utiliza para traducciones un algoritmo de tipo SecretKeyFactory tal y como se ve en los comentarios siguientes.
 // 2.1.1.1. Esta es la forma básica suministrada. Sin embargo, para los parámetros se proporcionan mecanismos más avanzados al permitir formatos de codificación pero son más complicados
 encodedKey = key.getEncoded();
 // ...
```

Tabla 51: JCE – 2.1. Traducción de la clave

- *Traducción del parámetro*

```
 // ...
 // 2.1.2. Se traduce el parámetro a su versión persistente. En este caso es sencillo pero no
es más que la traducción manual de un caso particular. En general se necesita manejar un
algoritmo del tipo AlgorithmParameters dificultando mucho más el proceso tal y como se
muestra a continuación. En total se muestra la forma manual pero particular y la forma
"automática" general. Observando en esta última que se utiliza un nombre distinto para el
algoritmo de traducción que para el algoritmo de protección/desprotección aunque sean nombres
"sinónimos"/alias o equivalentes
 // 2.1.2.1. Forma manual pero particular de traducir este parámetro
 if (parameter instanceof IvParameterSpec) {
  IvParameterSpec iv = (IvParameterSpec)parameter;
  encodedParameter = iv.getIV();
 }
  // ...
```

Tabla 52: JCE – 2.2. Traducción del parámetro

- *Otra traducción del parámetro*

```
 // ...
// 2.1.2.2. Forma "automática" pero general de traducir cualquier parámetro
AlgorithmParameters parameterTranslator = null;
try { parameterTranslator = AlgorithmParameters.getInstance("RIJNDAEL", providerName); }
catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
catch (NoSuchProviderException e) { e.printStackTrace(); return; }
try { parameterTranslator.init(parameter); }
catch (InvalidParameterSpecException e) { e.printStackTrace(); return; }
try { encodedParameter = parameterTranslator.getEncoded(); }
catch (IOException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 53: JCE – 2.3. Otra traducción del parámetro

- *Almacenamiento de los parámetros*

```
 // ...
// 2.2. Se guardan los parámetros
saveKey(encodedKey);
saveParameter(encodedParameter);
 // ...
```

Tabla 54: JCE – 2.4. Almacenamiento de los parámetros

3 Obtener el objeto asegurado utilizando los nuevos parámetros criptográficos

- *Carga de los parámetros*

```
 // ...
// ... tiempo más tarde en algún otro lugar del código ...
// 3. Se desea recuperar el objeto de forma segura
// 3.1. Cargamos los datos que se necesitan
encodedKey = loadKey();
encodedParameter = loadParameter();
 // ...
```

Tabla 55: JCE – 3.1. Carga de los parámetros

- *Definición del algoritmo de seguridad*

```
 // ...
 // 3.2. Inicializamos el algoritmo con los parámetros apropiados para poder obtener el
objeto asegurado
 // 3.2.1. Se selecciona el algoritmo para desproteger el objeto seguro
 try { secureAlgorithm = Cipher.getInstance("AES/CBC/PKCS7Padding", providerName); }
 catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
 catch (NoSuchProviderException e) { e.printStackTrace(); return; }
 catch (NoSuchPaddingException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 56: JCE – 3.2. Definición del algoritmo de seguridad

- *Traducción de los parámetros a la forma adecuada*

```
 // ...
// 3.2.2. Se inicializa el algoritmo para desprotección
// 3.2.2.1. Es necesario convertir los parámetros en su forma adecuada
// 3.2.2.1.1. Se reconvierte la clave a la forma adecuada
key = new SecretKeySpec(encodedKey, "AES");
// 3.2.2.1.2. Se reconvierte el parámetro a la forma adecuada
parameterType = IvParameterSpec.class; parameterTranslator = null;
try { parameterTranslator = AlgorithmParameters.getInstance("RIJNDAEL", providerName); }
catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
catch (NoSuchProviderException e) { e.printStackTrace(); return; }
try { parameterTranslator.init(encodedParameter); }
catch (IOException e) { e.printStackTrace(); return; }
try { parameter = parameterTranslator.getParameterSpec(parameterType); }
catch (InvalidParameterSpecException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 57: JCE – 3.3. Traducción de los parámetros a la forma adecuada

- *Inicialización del algoritmo de seguridad para desprotección*

```
 // ...
// 3.2.2.2. Se inicializa el algoritmo para desprotección
mode = Cipher.DECRYPT_MODE;
if (parameter != null && random == null) {
 try { secureAlgorithm.init(mode, key, parameter); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
} else if (parameter != null && random != null) {
 try { secureAlgorithm.init(mode, key, parameter, random); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
} else if (parameter == null && random == null) {
 try { secureAlgorithm.init(mode, key); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
} else if (parameter == null && random != null) {
 try { secureAlgorithm.init(mode, key, random); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
}
 // ...
```

Tabla 58: JCE – 3.4. Inicialización del algoritmo de seguridad para desprotección

- *Obtención del objeto asegurado*

```
 // ...
// 3.3. Se obtiene el objeto asegurado a partir de su versión en forma de objeto seguro
try {
 secureObject = (SealedObject)loadObject();
 object = (String)secureObject.getObject(secureAlgorithm); }
catch (IllegalBlockSizeException e) { e.printStackTrace(); return; }
catch (BadPaddingException e) { e.printStackTrace(); return; }
catch (IOException e) { e.printStackTrace(); return; }
catch (ClassNotFoundException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 59: JCE – 3.5. Obtención del objeto asegurado

4 Asegurar otro objeto reutilizando parámetros criptográficos ya existentes

- *Definición del objeto y carga de los parámetros*

```
 // ...
// ... tiempo más tarde en algún otro lugar del código ...
 // 4. Se desea asegurar un nuevo objeto con la misma clave y parámetro en un instante de tiempo posterior
 // 4.1. Se define el nuevo objeto
 String otherObject = "other object";
 // 4.2. Se cargan los parámetros que se necesitan
 encodedKey = loadKey();
 encodedParameter = loadParameter();
 // ...
```

Tabla 60: JCE – 4.1. Definición del objeto y carga de los parámetros

- *Traducción de la clave a su forma adecuada*

```
 // ...
// 4.3. Se inicializa el algoritmo
// 4.3.1. Se reconvierten los parámetros a la forma adecuada
// 4.3.1.1. Se reconvierte la clave a la forma adecuada
key = new SecretKeySpec(encodedKey, "AES");
 // ...
```

Tabla 61: JCE – 4.2. Traducción de la clave a su forma adecuada

- *Traducción del parámetro a su forma adecuada*

```
 // ...
// 4.3.1.2. Se reconvierte el parámetro a la forma adecuada
parameterType = IvParameterSpec.class; parameterTranslator = null;
try { parameterTranslator = AlgorithmParameters.getInstance("RIJNDAEL", providerName); }
catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
catch (NoSuchProviderException e) { e.printStackTrace(); return; }
try { parameterTranslator.init(encodedParameter); }
catch (IOException e) { e.printStackTrace(); return; }
try { parameter = parameterTranslator.getParameterSpec(parameterType); }
catch (InvalidParameterSpecException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 62: JCE – 4.3. Traducción del parámetro a su forma adecuada

- *Definición del algoritmo de seguridad*

```
 // ...
// 4.3.2. Se selecciona el algoritmo de seguridad
secureAlgorithm = null;
try { secureAlgorithm = Cipher.getInstance("AES/CBC/PKCS7Padding", providerName); }
catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
catch (NoSuchProviderException e) { e.printStackTrace(); return; }
catch (NoSuchPaddingException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 63: JCE – 4.4. Definición del algoritmo de seguridad

- *Inicialización del algoritmo de seguridad para protección*

```
 // ...
 // 4.3.3. Se inicializa el algoritmo con parámetros concretos (esta inicialización NO es
opcional)
 // 4.3.3.1. Se inicializa el algoritmo
 mode = Cipher.ENCRYPT_MODE;
 if (parameter != null && random == null) {
  try { secureAlgorithm.init(mode, key, parameter); }
  catch (InvalidKeyException e) { e.printStackTrace(); return; }
  catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
 } else if (parameter != null && random != null) {
  try { secureAlgorithm.init(mode, key, parameter, random); }
  catch (InvalidKeyException e) { e.printStackTrace(); return; }
  catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
 } else if (parameter == null && random == null) {
  try { secureAlgorithm.init(mode, key); }
  catch (InvalidKeyException e) { e.printStackTrace(); return; }
 } else if (parameter == null && random != null) {
  try { secureAlgorithm.init(mode, key, random); }
  catch (InvalidKeyException e) { e.printStackTrace(); return; }
 }
  // ...
```

Tabla 64: JCE – 4.5. Inicialización del algoritmo de seguridad para protección

- *Obtención del parámetro si fuera generado automáticamente*

```
  // ...
 // 4.3.3.2. Se capturan los parámetros si se han generado automáticamente. Esto es necesario
si no se especifican los algoritmos, e incluso es necesario aunque se especifique puesto que
el algoritmo puede ignorar el parámetro indicado por el usuario y generar otro de forma
automática
 try { algorithmParameters = secureAlgorithm.getParameters(); }
 catch (Throwable throwable) {}
 if (parameter == null && algorithmParameters != null) {
  if (parameterType == null) { parameterType = AlgorithmParameterSpec.class; }
  try { parameter = algorithmParameters.getParameterSpec(parameterType); }
  catch (InvalidParameterSpecException e) { e.printStackTrace(); return; }
 }
  // ...
```

Tabla 65: JCE – 4.6. Obtención del parámetro si fuera generado automáticamente

- *Creación del objeto seguro*

```
 // ...
// 4.4. Se asegura el nuevo objeto y se guarda en algún lugar
SealedObject otherSecureObject = null;
try { otherSecureObject = new SealedObject(otherObject, secureAlgorithm); }
catch (IllegalBlockSizeException e) { e.printStackTrace(); return; }
catch (IOException e) { e.printStackTrace(); return; }
saveObject(otherSecureObject);
 // ...
```

Tabla 66: JCE – 4.7. Creación del objeto seguro

5 Obtener el objeto asegurado reutilizando parámetros criptográficos ya existentes

- *Carga de los parámetros*

```
 // ...
// ... tiempo más tarde en algún otro lugar del código ...
// 5. Se desea recuperar el nuevo objeto asegurado
// 5.1. Cargamos los datos que se necesitan
encodedKey = loadKey();
encodedParameter = loadParameter();
 // ...
```

Tabla 67: JCE – 5.1. Carga de los parámetros

- *Definición del algoritmo de seguridad*

```
 // ...
 // 5.2. Inicializamos el algoritmo con los parámetros apropiados para poder obtener el
objeto asegurado
 // 5.2.1. Se selecciona el algoritmo para desproteger el objeto seguro
 try { secureAlgorithm = Cipher.getInstance("AES/CBC/PKCS7Padding", providerName); }
 catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
 catch (NoSuchProviderException e) { e.printStackTrace(); return; }
 catch (NoSuchPaddingException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 68: JCE – 5.2. Definición del algoritmo de seguridad

- *Traducción de la clave a su forma adecuada*

```
 // ...
// 5.2.2. Se inicializa el algoritmo para desprotección
// 5.2.2.1. Es necesario convertir los parámetros en su forma adecuada
// 5.2.2.1.1. Se reconvierte la clave a la forma adecuada
key = new SecretKeySpec(encodedKey, "AES");
 // ...
```

Tabla 69: JCE – 5.3. Traducción de la clave a su forma adecuada

- *Traducción del parámetro a su forma adecuada*

```
 // ...
// 5.2.2.1.2. Se reconvierte el parámetro a la forma adecuada
parameterType = IvParameterSpec.class; parameterTranslator = null;
try { parameterTranslator = AlgorithmParameters.getInstance("RIJNDAEL", providerName); }
catch (NoSuchAlgorithmException e) { e.printStackTrace(); return; }
catch (NoSuchProviderException e) { e.printStackTrace(); return; }
try { parameterTranslator.init(encodedParameter); }
catch (IOException e) { e.printStackTrace(); return; }
try { parameter = parameterTranslator.getParameterSpec(parameterType); }
catch (InvalidParameterSpecException e) { e.printStackTrace(); return; }
 // ...
```

Tabla 70: JCE – 5.4. Traducción del parámetro a su forma adecuada

- *Inicialización del algoritmo de seguridad para desprotección*

```
 // ...
// 5.2.2.2. Se inicializa el algoritmo para desprotección
mode = Cipher.DECRYPT_MODE;
if (parameter != null && random == null) {
 try { secureAlgorithm.init(mode, key, parameter); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
} else if (parameter != null && random != null) {
 try { secureAlgorithm.init(mode, key, parameter, random); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); return; }
} else if (parameter == null && random == null) {
 try { secureAlgorithm.init(mode, key); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
} else if (parameter == null && random != null) {
 try { secureAlgorithm.init(mode, key, random); }
 catch (InvalidKeyException e) { e.printStackTrace(); return; }
}
 // ...
```

Tabla 71: JCE – 5.5. Inicialización del algoritmo de seguridad para desprotección

- *Obtención del objeto asegurado*

```
 // ...
// 5.3. Se obtiene el objeto asegurado a partir de su versión en forma de objeto seguro
try {
 otherSecureObject = (SealedObject)loadObject();
 object = (String)otherSecureObject.getObject(secureAlgorithm); }
catch (IllegalBlockSizeException e) { e.printStackTrace(); return; }
catch (BadPaddingException e) {e.printStackTrace(); return; }
catch (IOException e) { e.printStackTrace(); return; }
catch (ClassNotFoundException e) { e.printStackTrace(); return; }
}
 // ...
```

Tabla 72: JCE – 5.6. Obtención del objeto asegurado

Tras observar el proceso anterior, se ve claramente que JCEF simplifica enormemente el uso de la criptografía ya que se obtiene el mismo resultado pero con menos líneas de código, menos conocimiento y además de una forma más fácil y sencilla. Por consiguiente, JCE complica el uso de la criptografía al obligar al usuario a escribir más líneas de código y tener más conocimiento para poder usar JCE.

7 Conclusiones

En resumidas cuentas, las conclusiones más importantes sobre el proyecto en sí son los principales aspectos valorables positivamente y también los puntos negativos.

Comencemos por algunos puntos que podrían considerarse como negativos. Éstos son los siguientes:

1. El tiempo empleado en el proyecto podría considerarse excesivo.
2. No haber desarrollado exactamente lo que estaba previsto desde un principio.

Los principales aspectos valorables positivamente son:

1. Haber desarrollado un trabajo novedoso en lugar de lo que se tenía previsto.
2. El uso 100% de herramientas Open Source o Freeware.
3. Haber aplicado el conocimiento de varias áreas: Ingeniería del software, Gráficos, Programación, Ofimática, Programación web, etc...
4. El enfoque orientado a objetos de la criptografía.
5. Publicación del proyecto en una página web (http://jcef.sourceforge.net).
6. El diseño de imágenes propias originales.
7. Haber sido capaz de trabajar en el proyecto sin prisas y sin pausas durante algo más de un año.
8. Las propuestas realizadas sobre posibles futuros proyectos fin de carrera.
9. La sección de preguntas frecuentes como ayuda adicional.