Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

Tipo de artículo: Artículo original
Temática: Seguridad informática
Recibido : 22/02/2022 | Aceptado: 31/05/2022

Diseño e implementación de un framework criptográfico

Design and implementation of a cryptographic framework

Roberto González González [1*] https://orcid.org/0000-0001-7093-5061

[1] Empresa de Tecnología y Sistemas DATYS. Ave 5ta #3401 entre 34 y 36, Playa, La Habana, Cuba. roberto.gonzales@datys.cu

*Autor para la correspondencia. (roberto.gonzales@datys.cu)

RESUMEN

La División de Sistemas y Servicios Criptográficos de la empresa DATYS tiene entre sus objetivos el desarrollo de aplicaciones y componentes de seguridad para la propia empresa y para terceros. Con el fin de cumplir este objetivo cuenta con implementaciones de esquemas criptográficos propios y otros que son estándares internacionales, implementados tanto en software como en hardware. Para poder dar mantenimiento eficientemente a las funcionalidades implementadas, facilitar su despliegue, actualización y reutilización, fue desarrollado un framework que exporta funcionalidades criptográficas a través de una interfaz de aplicación única a la que se puede acceder desde varios lenguajes de programación. Los usuarios consumen los algoritmos disponibles en proveedores criptográficos que son cargados por el framework como complementos o *plugins*. El diseño permitió aislar los algoritmos de las aplicaciones y brindar un

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

único punto de acceso a todos los esquemas criptográficos. También puede servir como referente a otros trabajos en los que se requiera una arquitectura capaz de gestionar dinámicamente un grupo de algoritmos.

Palabras clave: criptografía; seguridad; framework; arquitectura; diseño.

ABSTRACT

The Cryptographic Systems and Services Division of the DATYS company has among its objectives the development of applications and security components for the company itself and for third parties. In order to meet this objective, it has implementations of its own cryptographic schemes and others that are international standards, implemented in both software and hardware. In order to efficiently maintain the implemented functionalities, facilitate their deployment, updating and reuse, a framework was developed that exports cryptographic functionalities through a single application interface that can be accessed from various programming languages. Users consume the algorithms available in cryptographic providers that are loaded by the framework as add-ons or plugins. The design allowed to isolate the algorithms of the applications and to provide a single point of access to all the cryptographic schemes. It can also serve as a reference for other jobs that require an architecture capable of dynamically managing a group of algorithms.

Keywords: cryptography; security; framework; architecture; design.

Introducción

La División de Sistemas y Servicios Criptográficos de DATYS desarrolla aplicaciones y componentes con el objetivo de proteger información de personas naturales y jurídicas. En las aplicaciones se utilizan algoritmos criptográficos implementados con diferentes lenguajes de programación y entornos de desarrollo integrado (IDE[1]), haciendo difícil el mantenimiento y la actualización de los algoritmos. Muchos algoritmos son dependientes de un determinado IDE o sistema operativo y están vinculados a una arquitectura

específica, además algunos algoritmos están ligados unívocamente a una aplicación, con un alto acoplamiento que les resta modularidad y dificulta la reutilización del trabajo realizado.

En los componentes implementados hay algoritmos propios que no se encuentran en otras bibliotecas (Pelaiz et al., 2013, Pelaiz, 2017). Algunos de los más importantes son un generador de números pseudoaleatorios, un esquema de transporte de llaves de curva elíptica, un formato propio de certificados digitales de curva elíptica, un paquete de clases para aritmética de múltiple precisión y un dispositivo de tipo token criptográfico o dongle USB (Parrinha et al., 2017, SHALO, 2021) para ejecutar operaciones en hardware y almacenar llaves criptográficas de forma segura empleando el estándar PKCS11[2].

A partir de la situación anterior fue trazado el objetivo de diseñar e implementar un framework criptográfico que cumpliera con las siguientes características: debía ser independiente de las aplicaciones, de forma tal que pudiera ser llamado por todos los programas que requirieran criptografía a través de una interfaz lo más genérica posible. Debía ser integrable con otros lenguajes de programación diferentes al nativo en que fuera desarrollado, y debía ser posible añadir nuevos algoritmos y actualizar los existentes manteniendo la compatibilidad de las interfaces, y facilitando el trabajo a los desarrolladores abstrayéndolos de la complejidad de la criptografía subyacente.

Este trabajo fue realizado porque algunos de los algoritmos existentes no forman parte de estándares presentes en otras bibliotecas o frameworks, se cuenta con implementaciones en software y hardware que quieren exportarse en una sola interfaz de programación de aplicaciones (API[3]), y se quiere contar con un componente que sea accesible de forma sencilla desde otros lenguajes de programación para poder emplear una misma implementación base en todas las aplicaciones que se desarrollan en la empresa.

Un framework fue la opción elegida porque éstos poseen un diseño reutilizable, permiten reducir el tiempo de desarrollo y mantenimiento, brindan una mayor uniformidad en el trabajo y reducen la curva de aprendizaje cuando se pasa de desarrollar una aplicación a otra diferente. Sobre este tema se han desarrollado muchos trabajos, destacando entre los más importantes los frameworks .NET y Java. Estos dos casos marcan un referente en la forma en que se reúnen varios algoritmos criptográficos en una misma plataforma, y fueron la guía principal a partir de la cual se diseñó la arquitectura del framework propio.

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

Métodos o Metodología Computacional

Se comienza con una breve comparación de los principales frameworks y bibliotecas criptográficas que fueron estudiadas. Luego se describe la arquitectura del framework propio, los conceptos más importantes de dicha arquitectura y sus características fundamentales.

Comparación de frameworks criptográficos

Debido a lo extendido de su uso tanto en aplicaciones de propósito general como en soluciones criptográficas los frameworks que más se estudiaron para el desarrollo de este trabajo fueron **Java** y **.NET**, aunque hay muchos otros frameworks critpográficos y de seguridad (KDE, 2021, Abubakar et al., 2019, Hussain et al., 2018). Tanto *Java Cryptography Architecture* (JCA, ver (Oracle, 2021)) como .NET (Microsoft, 2021) tienen una arquitectura basada en proveedores con un framework que implementa la infraestructura necesaria y un grupo de proveedores que trabajan como complementos o *plugins* y suministran los algoritmos criptográficos. Esto permite agregar implementaciones haciendo solo algunos cambios en la configuración. Se pueden usar indistintamente proveedores diferentes o un solo proveedor. Cada proveedor debe implementar un grupo de interfaces bien definidas por el framework y el framework a su vez ofrece un conjunto de interfaces bien definidas a los usuarios finales, creando una arquitectura por capas.

Comparación de bibliotecas criptográficas

Existe un gran número de bibliotecas empleadas en aplicaciones criptográficas. Algunas de las más populares son OpenSSL (OpenSSL, 2021), Cryptlib (Gutmann, 2019), Crypto++ (Crypto++, 2021), Botan (Botan, 2021), MIRACL (MIRACL, 2018), NaCl (NaCl, 2016) y BouncyCastle (Castle, 2021). Todas implementan un gran número de algoritmos, centrándose mayormente en la eficiencia, y la mayoría están implementadas en lenguaje C/C++. El mayor problema con estas bibliotecas es lo complejo de su interacción con los usuarios y la dificultad que se presenta para agregar nuevos algoritmos y protocolos (ver, por ejemplo (Del Real, 2015, OpenSSL, 2019)).

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

Si bien agregar un nuevo algoritmo puede hacerse modificando las clases o estructuras existentes, agregar otro tipo de interfaz es más complicado, por ejemplo, agregar el formato de certificado digital usado en nuestra empresa implicaría reimplementar una buena parte de la biblioteca que se modifique. Las bibliotecas OpenSSL, Cryptlib, MIRACL y NaCl están implementadas en C, lo que hace difícil agregar funcionalidades debido a la complejidad de las interfaces en los lenguajes estructurados, y también se hace difícil el uso de las operaciones para los usuarios, aunque en muchos casos las bibliotecas poseen *wrappers* propios o de terceros para permitir su uso en otros lenguajes de programación.

Las funcionalidades de OpenSSL y MIRACL son particularmente complejas para un usuario con pocos conocimientos criptográficos, y es muy fácil cometer errores debido al diseño poco amigable de las interfaces. Aunque se utilice ampliamente para proteger las conexiones en internet, la implementación de OpenSSL ha recibido críticas debido a la complejidad de su código y a la necesidad de mantenimiento o re implementación (Brodkin, 2014, Kamp, 2014). Estas críticas se han renovado después del fallo de seguridad del año 2014 conocido como Heartbleed (Durumeric et al., 2014).

En el caso de NaCl el objetivo de los desarrolladores fue crear una interfaz mínima que redujera al máximo la complejidad para los usuarios, pero al lograr esto también hacen más difícil tener en cuenta esta biblioteca a la hora de agregar nuevos algoritmos. El caso de BouncyCastle es posiblemente el que presenta el mejor diseño, pero está desarrollado en Java y C# y se desea reutilizar los algoritmos ya implementados en la empresa, que están desarrollados sobre todo en C/C++.

A la complejidad de agregar nuevos algoritmos e interfaces se suma el hecho de que por las características del tema de este trabajo es preferible contar con una solución propia que no dependa totalmente de terceros. Las bibliotecas de terceros se usarán siempre que sea necesario en la capa inferior de la arquitectura, de forma tal que se pueda aprovechar su eficiencia sin que esto afecte la extensibilidad y la sencillez de la comunicación entre los usuarios y el framework.

Estas bibliotecas pueden emplearse mejor si son utilizadas en complementos o como proveedores en la capa inferior de una arquitectura de varios niveles. De esta forma si una biblioteca deja de recibir mantenimiento o cambia su licencia solo implica dejar de utilizar una implementación en un proveedor. Se puede sustituir ese proveedor por otro o crear una nueva versión del mismo proveedor con algoritmos propios o con otra biblioteca de terceros sin que el usuario tenga que percibir el cambio realizado.

Conclusiones sobre los frameworks y bibliotecas analizadas

Los frameworks son ampliamente utilizados para implementar servicios criptográficos con el objetivo de estandarizar la forma de desarrollar las aplicaciones, facilitar la extensibilidad, el mantenimiento y reducir el tiempo de desarrollo. Estos tienden a estar organizados en componentes e instanciar un grupo de patrones de diseño. Para este trabajo se tendrán en cuenta como guía principalmente la arquitectura de Java JCA y en menor medida la del framework .NET. Las bibliotecas criptográficas se emplearán solo en los componentes de más bajo nivel de la arquitectura, para aprovechar su eficiencia sin tener que lidiar con sus interfaces en la API que exporte los servicios del framework a los usuarios.

Diseño de la arquitectura

La idea central de la arquitectura fue separar las interfaces de las implementaciones. Todas las funcionalidades del framework están centradas en el uso de servicios criptográficos, que son un grupo de interfaces exportadas por una única API. Los algoritmos que utilizan los servicios se encuentran en proveedores criptográficos y de esta forma una aplicación no está ligada a un proveedor específico. Además de la idea de los servicios y proveedores, se tomaron de JCA los conceptos de algoritmo o *engine* para las implementaciones disponibles en los proveedores, y CSPI[4] para las interfaces que permiten la comunicación entre el framework y los proveedores. Se definió como *modo* la combinación de uno o más algoritmos especificados por el usuario para llamar un servicio. Cada modo requiere el uso de uno o varios *engines* que se cargan desde uno o varios proveedores. Los *engines* son gestionados por el núcleo del framework, que decide el proveedor o los proveedores a utilizar, aunque el usuario puede forzar el uso de un proveedor de su preferencia.

Los servicios deben ocultar las particularidades de los algoritmos que utilizan, para que sea posible usarlos requiriendo sólo el mínimo de conocimientos sobre criptografía, y deben ser lo suficientemente genéricos para permitir agregar nuevos algoritmos sin que sea necesario modificar las interfaces con que los usuarios interactúan. Además de los servicios se implementaron funciones de control para obtener información de los servicios disponibles, instalar y desinstalar proveedores y otras utilidades.

El framework está organizado en cuatro capas de abstracción que forman la estructura básica de su arquitectura (ver figura 1). La capa más importante es la del núcleo o componente principal (segunda desde abajo), que contiene la definición de todos los servicios y tipos de datos y gestiona todos los proveedores disponibles. En la capa inferior se encuentran los proveedores criptográficos o CSP[5], que se comunican con el framework a través de las interfaces de acceso a proveedores CSPI. Cada servicio requiere el uso de al menos un *engine* y cada tipo de *engine* es implementado por un CSPI diferente. Un *engine* puede ser utilizado como parte de más de un servicio, y cuando el usuario modifica el modo del servicio que invoca está cambiando implícitamente el o los *engines* seleccionados.

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

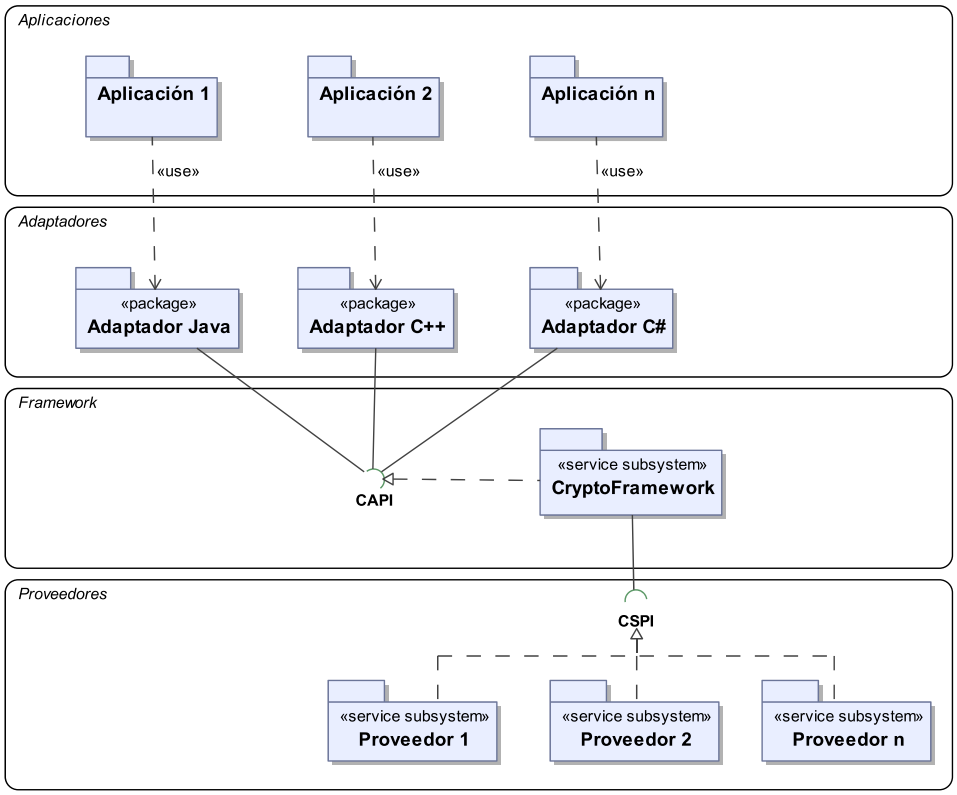

Fig. 1 – Arquitectura del framework criptográfico.

De esta forma los componentes adquieren gran modularidad. Un servicio como la firma puede utilizar el mismo *engine* para calcular el resumen de un dato que el que utiliza un servicio HMAC[6]. También es posible tener un proveedor que implemente sólo las interfaces CSPI necesarias para el cálculo del resumen u otra funcionalidad, mientras otros desarrolladores crean proveedores para otros CSPI, y luego todos los CSP pueden "colaborar" entre sí para permitir al usuario la ejecución de un servicio. El framework combina todas las funcionalidades de los proveedores permitiendo a los usuarios seleccionar distintos algoritmos para formar el modo que les sea más conveniente, dando la posibilidad de escoger los algoritmos adecuados entre cientos de combinaciones diferentes. En la segunda capa desde arriba se ubican los adaptadores para acceder al framework de forma sencilla y desde varios lenguajes de programación. Por último en la capa

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

superior se encuentran las aplicaciones implementadas por los usuarios, que usan los paquetes de clases de los adaptadores para invocar los servicios criptográficos.

Resultados y discusión

Para probar el framework se desarrollaron tres proveedores criptográficos con varios de los algoritmos existentes y algunas implementaciones nuevas. Las siguientes secciones muestras la organización de los proveedores y los servicios y *engines* disponibles en esta primera versión.

Modelo de implementación del sistema

En la figura 2 se pueden observar todos los componentes necesarios para utilizar el framework con los tres proveedores que están disponibles actualmente. En la parte superior del diagrama se encuentran las aplicaciones de los usuarios que utilizan el framework. Estas aplicaciones son enlazadas con los adaptadores de una forma específica para cada lenguaje de programación que los usuarios utilicen. Se chequea el archivo *security.cfg* cuando se inicializa el framework para conocer el nombre de los proveedores instalados y agregarlos a la lista de proveedores disponibles, siempre que la firma de los proveedores sea válida. Todos los componentes de color violeta mostrados en el diagrama son paquetes de clases, los de color amarillo son bibliotecas de enlace dinámico (DLL[7]) en Windows y objetos compartidos (SO[8]) en Linux.

Servicios y proveedores disponibles

Actualmente la API del framework exporta siete tipos de servicios. Estos permiten a los adaptadores ejecutar las funcionalidades disponibles en los proveedores. En la tabla 1 se muestra el número de servicios agrupado por tipo. Estos servicios consumen los *engines* de los proveedores implementados. Una parte de los algoritmos criptográficos disponibles en la empresa se reimplementaron en tres proveedores criptográficos nombrados DATYS, DATYS_GMP y DATYS_TOKEN.

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

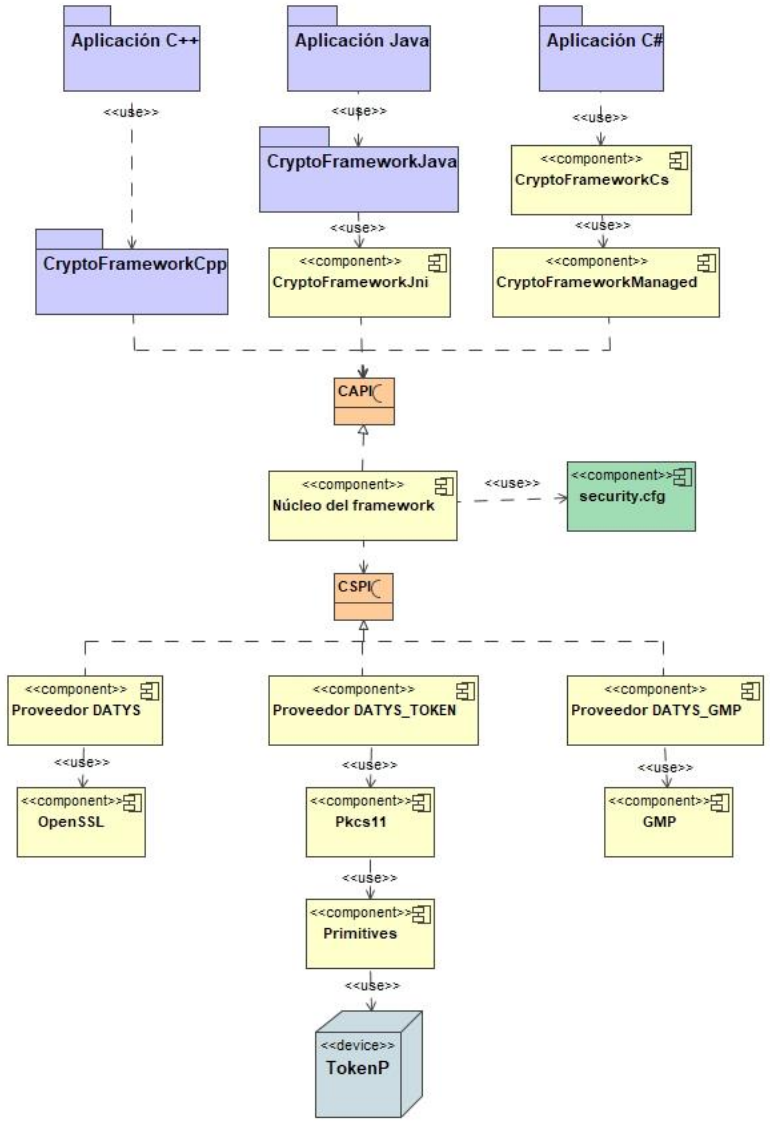

Fig. 2 – Modelo de implementación del sistema.

El proveedor DATYS contiene la mayor parte de los algoritmos y emplea OpenSSL para gestionar los certificados X509. El proveedor DATYS_GMP utiliza la biblioteca matemática GMP[9] para ejecutar las operaciones de múltiple precisión. Se trata de un proveedor que fue implementado para probar el rendimiento de la biblioteca GMP y contiene solamente algoritmos para el servicio de firma.

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

Tabla 1 – Tipos de servicios criptográficos definidos por el framework.

Tipo de servicio	Número de funciones	Tipo de operación
Firma digital de buffer y fichero	4	Firma/verificación
Resumen de buffer y fichero	2	Resumen
Cifrado simétrico de buffer y fichero	4	Cifrado/descifrado
MAC de buffer y de fichero	2	MAC
Generación de números aleatorios	2	Generación
Esquemas de transporte de llave con criptografía de llave pública	2	Cifrado/descifrado
Acceso a campos de certificados digitales	11	Ver datos de certificado

El proveedor DATYS_TOKEN no contiene algoritmos, sino una interfaz intermedia con el estándar PKCS11, que permite consumir los algoritmos disponibles en la llave de hardware Arcano Tokenp desarrollada por nuestra empresa. El principal objetivo de este proveedor es acceder fácilmente al dispositivo desde los adaptadores, sin tener que lidiar con la complejidad de la interfaz PKCS11. La mayor parte de los algoritmos fueron simplificados y adaptados a las nuevas interfaces del framework. En la tabla 2 se agrupan todos los algoritmos (*engines*) que contienen actualmente los proveedores.

Ejemplo integrador

Para finalizar en la figura 3 se muestra un ejemplo de uso del framework. El código permite obtener la firma digital de un arreglo de bytes empleando el adaptador para el lenguaje C++. Este ejemplo permite relacionar todos los conceptos descritos en un caso real. Para mayor sencillez se omite la captura de excepciones y la liberación de la memoria. Primero el usuario inicializa el framework, luego carga la llave privada y luego inicializa el servicio de firma especificando el modo "ECDSA/RAMG/SHA2-256", que hace que el framework busque *engines* para generar firma ECDSA, generación de pseudoaleatorios con RAMG y generación de resumen con SHA2-256. Cada *engine* puede estar en un proveedor independiente y el usuario no tiene que preocuparse por la implementación, pero en este caso el usuario especifica que desea utilizar el proveedor DATYS siempre que sea posible. El resultado es un arreglo de bytes que contiene la firma; de ser necesario esta firma se puede guardar en un fichero empleando otro servicio.

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

Tabla 2 – Engines o algoritmos disponibles en los proveedores criptográficos desarrollados.

Tipo de engine o algoritmo	Nombre	Descripción
Cifrado Simétrico	AES, GOST 28147-89, GOST Kuznyechik, Serpent, Twofish, modos CBC, OFB, CFB.	Estándar de cifrado de los Estados Unidos, estándar soviético, estándar ruso y dos candidatos a AES.
	CKM_AES_CBC	Mecanismo de AES CBC en hardware, PKCS11
Completamiento (padding)	PKCS7 y ANSI-X923	Completamiento según PKCS7 y ANSI
Esquema de transporte de llaves con criptografía de llave pública	ECIES	Esquema de cifrado con llave pública utilizando ECC
	ECIES-A	Variante propia de ECIES
	RSA	Esquema de cifrado con llave pública RSA
Funciones resumen	SHA-2 y SHA-3 (256, 384, 512)	Funciones resumen publicadas por el NIST
Funciones MAC	HMAC	Función MAC con resumen
Generación de números pseudoaleatorios	RAMG	Generador propio
	C_GenerateRandom	Función del generador en hardware, PKCS11
Firma digital	ECDSA	Estándar de firma digital
	GOST 34.10	Estándar ruso de firma digital
	ECGDSA	Esquema alemán de firma digital
	ECKDSA	Esquema coreano de firma digital
	ECSCHNORR	Esquema de firma digital
	RSA	Esquema de firma digital basado en el algoritmo RSA
	CKM_RSA_PKCS	Mecanismo de RSA en hardware, PKCS11
Gestión de certificados	X.509 (.p12, .pem, .cer)	Certificados para criptografía de llave pública
	ECER (.ECer, .ECpk) versiones 1 y 2	Certificados propios para criptografía de llave pública

La inicialización del framework empleando la clase Security debe hacerse solo una vez antes de usar el framework. El objeto de tipo IParameterSpecList es un listado de parámetros genéricos donde se cargan los datos de la llave privada usada para firmar. El framework se encarga internamente de extraer los parámetros de este objeto y verificar que son del tipo necesario, o en caso contrario retorna un error. Como se explicó anteriormente para cambiar los algoritmos el usuario solo tiene que cambiar el modo. Si por ejemplo quiere

Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 51-67

cambiar la función resumen es tan sencillo como sustituir el texto *SHA2-256* por el nombre de otro *engine* como *SHA2-512*.

```
#include "CryptoFrameworkCpp2/Wrapper/Security.h"
#include "CryptoFrameworkCpp2/Wrapper/Signature.h"
#include "CryptoFrameworkCpp2/Wrapper/CertParamSpecFactory.h"
using namespace CryptoFrameworkCpp2;

void main()
{
   // Se inicializa el framework
   Security::init(true);

   // Se cargan los parámetros de firma desde un certificado
   CertParamSpecFactory ecdsaPrivKeyFactory("CertParamSpecFactory.X509-PRIVKEY");
   CryptoFramework2::IParameterSpecList* pEcdsaPrivKeyParam = nullptr;
   std::string passw("123456");
   ecdsaPrivKeyFactory.setPasswords(&passw, 1);
   ecdsaPrivKeyFactory.getDecoded("TestVectors/ecc.p12", pEcdsaPrivKeyParam);

   // Se preparan los datos a firmar
   const uint32_t dataSize = 16;
   uint8_t pData[dataSize] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};

   // Se firman los datos
   Signature signature("Signature.ECDSA/Random.RAMG/Digest.SHA2-256", "DATYS", true);
   signature.initSign(pEcdsaPrivKeyParam->getParameterList()[0]);
   uint32_t signatureLength = signature.getSignatureLength();
   uint8_t *pSignature = new uint8_t[signatureLength];
   signature.update(pData, dataSize);
   signature.sign(pSignature, signatureLength);
}
```

Fig. 3 – Ejemplo de firma digital. Se ignora el manejo de excepciones y la liberación de memoria.

Conclusiones

El framework criptográfico ha solucionado varios problemas de diseño que tenían los mecanismos implementados anteriormente en la entidad, incrementando la modularidad y disminuyendo el alto acoplamiento, facilitando el desarrollo de las nuevas aplicaciones. Las características de este nuevo enfoque permiten aislar los mecanismos de seguridad de las aplicaciones de los usuarios, haciendo más fácil el mantenimiento y la actualización de los algoritmos. Otros usuarios pueden crear sus propios proveedores y

extender las funcionalidades actuales implementando módulos que cumplan con lo establecido en las interfaces CSPI. Al desarrollarse los *wrappers* los usuarios pueden acceder de forma sencilla a las mismas funcionalidades desde varios lenguajes de programación, sin que sea necesario tener amplios conocimientos de criptografía.

Todas estas características hacen del framework una solución propia efectiva, sencilla y extensible que puede solucionar la mayor parte de los problemas relacionados con la criptografía, ofreciendo una amplia gama de servicios criptográficos a las aplicaciones que lo requieran, en cualquiera de los lenguajes de programación para los que se encuentra disponible y otros que se puedan agregar en el futuro.

Como trabajo pendiente queda agregar más algoritmos e interfaces, probarlo en el sistema operativo Android y evaluar la posibilidad de agregar servicios para criptografía pos cuántica.

Este diseño basado en proveedores y C++ se puede reutilizar en otras soluciones en que sea necesario elegir dinámicamente entre un grupo de algoritmos y emplear una misma solución en varios lenguajes de programación.

Referencias

Pelaiz, S.; Tejera, R. Un Nuevo Procedimiento De Generación Seudo Aleatoria De Semillas. Computación Y Sistemas, 2013, 17 (1), Issn 1405-5546, P. 63-68.

Hussain, M.; Al-Haiqi, A., Et Al. A Security Framework For Mhealth Apps On Android Platform. Computers & Security, 2018, 75, P. 191-217.

Pelaiz, S. Algoritmia Matemático-Criptográfica Soportada En La Teoría De Las Curvas Elípticas. Doctorado En Matemáticas. Datys Y Cenatav. La Habana, 2017.

Abubakar, S.; Dong, Y., Et Al. Cyber Security Framework For Internet Of Things-Based Energy Internet. Future Generation Computer Systems, 2019, 93, P. 849-859.

Brodkin, J. Openssl Code Beyond Repair, Claims Creator Of "Libressl" Fork. [En Línea]. Ars Technica, 2014, [Consultado El: 13 De Diciembre De 2021]. Disponible En Https://Arstechnica.Com/Information-Technology/2014/04/Openssl-Code-Beyond-Repair-Claims-Creator-Of-Libressl-Fork.

Kamp, P. Please Put Openssl Out Of Its Misery. [En Línea]. Acm Queue, 2014, Vol. 12, [Consultado El: 13 De Diciembre De 2021]. Disponible En: Https://Queue.Acm.Org/Detail.Cfm?Id=2602816.

Durumeric, Z.; Kasten, J., Et Al. The Matter Of Heartbleed. En: Imc '14: Proceedings Of The 2014 Conference On Internet Measurement Conference. Acm Digital Library, 2014, P. 475-488.

Oracle. Java Cryptography Architecture (Jca) Reference Guide. [En Línea]. Oracle, Java Documentation, 2021. [Consultado El: 15 De Diciembre De 2021]. Disponible En: Https://Docs.Oracle.Com/Javase/8/Docs/Technotes/Guides/Security/Crypto/Cryptospec.Html.

Castle, B. Bouncy Castle Web Site. [En Línea]. The Legion Of The Bouncy Castle, 2021. [Consultado El: 15 De Diciembre De 2021]. Disponible En: Https://Www.Bouncycastle.Org.

Openssl. Openssl Web Site. [En Línea]. Openssl, 2021. [Consultado El: 15 De Diciembre De 2021]. Disponible En: Https://Www.Openssl.Org.

Gutmann, P. Cryptlib Security Toolkit. [En Línea]. Cryptlib Web Site, 2019. [Consultado El: 15 De Diciembre De 2021]. Disponible En: Https://Www.Cryptlib.Com/Downloads/Manual.Pdf.

Crypto++. Crypto++® Library. [En Línea]. Crypto++ Web Site, 2021. [Consultado El: 15 De Diciembre De 2021]. Disponible En: Https://Www.Cryptopp.Com.

Botan. Botan: Crypto And Tls For Modern C++. [En Línea]. Botan Web Site, 2021. [Consultado El: 15 De Diciembre De 2021]. Disponible En: Https://Botan.Randombit.Net.

Nacl. Nacl: Networking And Cryptography Library. [En Línea]. Nacl Web Site, 2016. [Consultado El: 15 De Diciembre De 2021]. Disponible En: Https://Nacl.Cr.Yp.To.

Miracl. Miracl Library. [En Línea]. Miracl Web Site, 2018. [Consultado El: 26 De Noviembre De 2018]. Disponible En: Https://Libraries.Docs.Miracl.Com.

Microsoft. .Net Cryptography Model. [En Línea]. Microsoft Technical Documentation, 2021. [Consultado El: 10 De Noviembre De 2021]. Disponible En: Https://Docs.Microsoft.Com/En-Us/Dotnet/Standard/Security/Cryptography-Model.

Kde. Qca. [En Línea]. Kde Api Reference, 2021. [Consultado El: 12 De Diciembre De 2021]. Disponible En: Https://Api.Kde.Org/Qca/Html.

Shalo. Shalo Auth. [En Línea]. Security Hardware For Authorization And Licensing Operations, 2021. [Consultado El: 12 De Diciembre De 2021]. Disponible En: Https://Shalo.Jp/En/Auth/.

Del Real, C. J. Openssl.Cu: Distribución Propia De Openssl. Tesis De Ingeniería Informática. Facultad De Informática. Ispjae, 2015.

Openssl. Creating An Openssl Engine To Use Indigenous Ecdh Ecdsa And Hash Algorithms. [En Línea]. Openssl Wiki, 2019. [Consultado El: 15 De Diciembre De 2021]. Disponible En: Https://Wiki.Openssl.Org/Index.Php/Creating_An_Openssl_Engine_To_Use_Indigenous_Ecdh_Ecdsa_And_Hash_Algorithms.

Parrinha, D.; Chaves, R. Flexible And Low-Cost Hsm Based On Non-Volatile Fpgas. En: International Conference On Reconfigurable Computing And Fpgas (Reconfig). Cancún, México: Ieee, 2017, P. 1-8.

Conflicto de interés

El autor autoriza la distribución y uso del artículo.

Contribuciones de los autores

El trabajo fue realizado completamente por su único autor, Roberto González González.

[1] Integrated Development Environment (IDE): Entorno de Desarrollo Integrado para el desarrollo de software.

[2] Public Key Cryptography Standards (PKCS): Estándares de Criptografía de Llave Pública.

[3] Application Programming Interface (API): Capa de abstracción publicada por una biblioteca de software.

[4] Crypto Service Programming Interface (CSPI): Interfaz para acceder a proveedor de servicios criptográficos.

[5] Crypto Service Provider (CSP): Proveedor de servicios criptográficos para framework criptográfico.

[6] Hash based Message Authentication Code (HMAC): Código de Autenticación de Mensaje con función resumen.

[7] Dynamic Link Library (DLL): Biblioteca de Enlace Dinámico del sistema operativo Windows.

[8] Shared Object file (SO): Biblioteca de Objeto Compartido de los sistemas operativos tipo GNU/LINUX.

[9] GNU Multiple Precision (GMP): Biblioteca de GNU para el trabajo con enteros de múltiple precisión.