DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI
ISSN: 2007-6363

Publicación Semestral Pädi Vol. 10 No. 19 (2022) 70-78

Firma electrónica habilitada por tarjetas inteligentes para fines de seguridad
Electronic signature enabled by smartcards for security purposes

O. De La Cruz-Vite[a,*], F. Barrera-Mora[a], A. M. Perez-Gomez[a]

[a] *Área Académica de Matemáticas y Física, Universidad Autónoma del Estado de Hidalgo, 42184, Pachuca, Hidalgo, México.*

Resumen

En la actualidad, las actividades sociales, académicas, económicas y tecnológicas generan una gran cantidad de información que es necesario transmitir. Esto ha motivado que gran parte de las tareas técnicas, relacionadas con el manejo de la información, se desarrollen para proveer medios confiables. Con este antecedente, en este trabajo se describe una forma para la implementación de firma electrónica usando un sistema para encriptar. Específicamente, usando el sistema de encriptamiento de llave publica RSA y tarjetas inteligentes, se muestra el proceso de firma electrónica cuya finalidad es transmitir información de manera segura cuando se realizan operaciones de autenticación y firma electrónica. Otro elemento que se aborda en este trabajo, es la definición y uso del Protocolo de Aplicación para la Unidad de Datos (APDU) que se utiliza al comunicarse de manera directa entre un lector de tarjetas y una tarjeta inteligente.

Palabras Clave: Tarjetas inteligentes, Cifrado RSA, Firma electrónica, Certificado Digital, ISO 7816.

Abstract

At present, social, academic, economic and technological activities generate a large amount of information that needs to be transmitted. This has motivated that a great amount of technical tasks, related to information management, need to be addressed to provide reliable means. With this starting point in mind, this paper describes a way to implement an electronic signature using an encryption system. Specifically, using the RSA public key encryption system and smart cards, the electronic signature process is described, whose purpose is to transmit information securely when authentication and electronic signature operations are performed. Another element that is addressed in this work, is the definition and use of the Application Protocol for Data Unit (APDU) which is used when communicating directly between a card reader and a smart card.

Keywords: Smartcard, RSA cipher, Electronic signature, Digital Certificate, ISO 7816.

1. Introducción

Se dice con frecuencia que vivimos en la era de la información, lo cual se puede interpretar de diversas formas, pero en cualquiera de estas, la transmisión de información conlleva dos problemas de importancia que deben ser abordados. El primero de ellos está asociado con la inmensa cantidad de datos que circulan en los diversos medios, lo cual hace imprescindible el uso de herramientas digitales para agilizar su fluidez, por lo que necesariamente lleva al uso de sistemas de cómputo; el segundo problema radica en la transmisión segura de la información, por lo tanto se deben utilizar sistemas que garanticen la inviolabilidad de los datos; para abordar este problema se utilizan sistemas de encriptamiento que sean seguros y de fácil uso; uno de estos sistemas es el bien conocido método de encriptamiento RSA, el cual ha probado, hasta nuestros días, satisfacer los estándares que se requieren para la transmisión segura de datos.

En la actualidad, casos específicos en donde el manejo y transmisión de la información requiere satisfacer estándares de seguridad estrictos son los relacionados con pasaportes electrónicos, tarjetas de banco, entre muchos otros. En estos últimos años la expedición de pasaportes electrónicos está cobrando mucha relevancia, la cual puede atribuirse a la alta movilidad de una cantidad creciente de personas; estos documentos aparte de cumplir con protocolos de seguridad en el proceso de manufactura física, también cuentan con un chip en el cual se almacena la información necesaria del portador, para que de esta forma se pueda revisar de manera rápida y sencilla. Dicho proceso requiere contar con una infraestructura simple y confiable, lo cual puede lograrse con el uso de un

*Autor para la correspondencia: drargoe13@gmail.com
Correo electrónico: drargoe13@gmail.com (Omar de la Cruz-Vite), fbarrera10147@gmail.com (Fernando Barrera-Mora), adidier.perez@gmail.com (Adidier M. Pérez Gómez).

Fecha de recepción: 14/01/2022 Fecha de aceptación: 29/04/2022 Fecha de publicación (postprint): 02/05/2022
Fecha de publicación: 05/07/2022 https://doi.org/10.29057/icbi.v10i19.8583

sistema de encriptamiento. De la diversidad de éstos, el que se describe y usa en este trabajo es el bien conocido sistema de encriptamiento RSA. Debemos mencionar que desde un punto de vista práctico, el uso de un sistema de encriptamiento requiere diseñar protocolos y dispositivos para validar y transmitir información.

En este trabajo encontraremos: primeramente, en la sección 2 presentamos los elementos que describen los fundamentos del sistema de encriptamiento RSA; posteriormente, en la sección 3 se detalla lo que es una tarjeta inteligente (Smartcad) así como su estructura de directorios y registros. En la sección 4 se describe el Protocolo de Aplicación para la Unidad de Datos (Application Protocol Data Unit) que es el instrumento utilizado para establecer la comunicación entre la tarjeta inteligente y el lector de tarjetas. Adicionalmente, presentamos algunos ejemplos reales para explicar la manera en que se realizan las operaciones utilizando una tarjeta inteligente, los cuales se utilizan en el proceso de firmar digitalmente. Finalmente, en el Apéndice A se puede encontrar el código necesario para realizar el proceso de firma digital el cual fue desarrollado usando el lenguaje de programación C++.

El presente trabajo surge como parte del proyecto SgSmartCardLib el cual se desarrolló a petición de la empresa Seguridata Privada S.A de C.V. La parte que aquí se reporta busca ser una guía para aquellos que quieran empezar a trabajar con el proceso de firma electrónica y/o con el uso del protocolo de aplicación para la unidad de datos. También puede ilustrar una forma de interacción entre la teoría de números y la computación, al resolver un problema practico.

2. El Sistema de Encriptamiento RSA

En esta sección presentamos los elementos fundamentales de uno de los métodos más conocidos para cifrar mensajes. La idea central con el uso de estos sistemas, es que al enviar información solamente el destinatario pueda conocer su contenido, aunque pudiese caer en manos de otros individuos.

Antes de presentar algunos detalles técnicos del sistema de encriptamiento RSA, hace falta describir ideas generales de lo que es tal sistema. De acuerdo con Koblitz (Koblitz, 1987), al mensaje que ha de cifrarse se le llama texto sin formato, mientras que el texto cifrado se le conoce como texto encriptado. Para precisar ideas se inicia con algunos términos. Un alfabeto es una colección de N letras o caracteres, con éste se representa el texto sin formato, así como el texto cifrado. El proceso de convertir un texto sin formato en texto encriptado se le llama cifrar o encriptar; mientras que al proceso inverso se le conoce como descifrar o desencriptar. A un texto sin formato o cifrado se le divide en unidades de mensaje que pueden consistir en uno o varios caracteres del alfabeto que se está usando. Una transformación de cifrado es una función (en sentido matemático) que transforma una unidad de texto sin formato en una unidad de texto cifrado. En forma más precisa, una transformación de cifrado es una función inyectiva de P en C, en donde P es el conjunto de todas las unidades de texto sin formato y C es el conjunto de todas las unidades de texto cifrado. Una función es inyectiva si manda elementos diferentes de P en elementos diferentes de C (Dugundji, 1966, p. 13). Si f representa la función de cifrado, como es inyectiva, entonces existe su inversa, la función f^{-1} que representa el proceso de descifrar. Un sistema de encriptamiento es la cuádrupla $\left(P, C, f, f^{-1}\right)$. Para declarar un sistema de encriptamiento se requiere "etiquetar" a todas las unidades de mensaje sin formato y cifradas. Esto se hace usualmente con un sistema algebraico, que puede ser los enteros módulo N, o cualquier otro en el cual se definirán las funciones de cifrado y sus inversas. El sistema algebraico que se elija tendrá entre sus características, la posibilidad de definir de manera práctica la función de encriptamiento, y a la vez la dificultad de poder encontrar su inversa. Si el alfabeto que se utiliza es el usual del español: a, b c, ..., z; se pueden usar los enteros del 0 al 26 para etiquetar las unidades de mensaje que constan de un solo caracter del alfabeto. Desde un punto de vista práctico, a los equivalentes numéricos del alfabeto los podemos considerar como los dígitos en base 27, de forma más precisa, los dígitos en base 27 son 0, 1, 2, ..., 26 y las letras del alfabeto están representadas así: a es cero; b es 1; c es 2 etc. Con esta notación, el texto sin formato, "hola", se representa como $7\times27^3+15\times27^2+11\times27+0$; en otras palabras, al texto "hola" se le puede identificar con el entero $7\times27^3+15\times27^2+11\times27+0$ cuyos dígitos en base 27 son 7, 15, 11 y 0. Note que hay diferencia entre los dígitos en base 10 y los dígitos en base 27, por ejemplo, en base 27, "11" es un dígito, mientras que en base 10 se trata de un par de dígitos.

Uno de los sistemas de cifrado que más se ha usado desde su publicación (1978), es el sistema RSA, denotado así por contener las primeras letras de los apellidos de sus inventores: Rivest, Shamir y Adleman (Rivest et al., 1978); los fundamentos de este sistema de cifrado serán descritos en seguida, para tal efecto requerimos introducir algunos elementos de la teoría de números.

Dados los números enteros a, b y n, con este último positivo, decimos que a y b son congruentes módulo n, escrito $a \equiv b (mód\ n)$, si n divide a a-b, es decir, si existe un entero q tal que $a-b = qn$. Una forma alternativa de saber que los enteros a y b son congruentes módulo n, es que la división entre n de a y b deje el mismo residuo; por ejemplo, si $n = 7$ entonces 18 y 11 son congruentes módulo 7, dado que ambos dejan residuo 4 al dividirse entre 7. De acuerdo con el algoritmo de la división, al dividir un entero a entre n, los posibles residuos son: $0, 1, 2, \ldots, (n\text{-}1)$, entonces a es congruente con exactamente uno de los enteros $0, 1, 2, \ldots, (n\text{-}1)$. A los enteros que son divisibles entre n se les llama la clase del 0 módulo n; a los que dejan residuo 1 al dividirse entre n se les llama la clase del 1 módulo n. En general, si $r \in \{0, 1, \ldots, (n\text{-}1)\}$ entonces a los enteros que dejan residuo r al dividirse entre n se les llama la clase de r módulo n y a la colección de todas las clases módulo n se les denota por $\frac{\mathbb{Z}}{n\mathbb{Z}}$.

En los números enteros es bien conocida la aritmética, es decir, se conocen bien los resultados que se obtienen a partir de las propiedades de la suma y producto de enteros. Ser congruentes da lugar a una suma y producto que tienen propiedades similares a las de la suma y producto en los números enteros. En particular se puede hablar de ecuaciones módulo n. Por ejemplo, se pueden formular ecuaciones del tipo $x^e \equiv a (mód\ n)$ que es uno de los ingredientes centrales en el método de encriptamiento RSA.

Uno de los términos más importantes en aritmética es el de *número primo*; éste es un número entero positivo que tiene exactamente dos divisores: el uno y él mismo. Por ejemplo, 2, 3 y 23 son números primos. Otro término de utilidad es el de

primos relativos. Dos números enteros a y b se dicen primos relativos, si el único entero positivo que los divide a ambos es el uno.

Las funciones en los números enteros son de tal importancia que algunas se asocian con símbolos y tienen significados específicos. Una de estas funciones es la llamada *función de Euler*, definida como: $\varphi(n)$ es la cantidad de números enteros, entre 1 y n, que son primos relativos con n, por ejemplo, $\varphi(5) = 4$. En general, si p es un número primo, $\varphi(p) = p - 1$. Una propiedad fundamental de la función de Euler es: si n y m son primos relativos, entonces $\varphi(mn) = \varphi(m)\varphi(n)$.

A continuación, se presentan 3 teoremas, el 1 y 2 aparecen sin demostración; sin embargo, el ávido lector podría intentar producir una o consultar las referencias citadas al final del trabajo. La demostración del Teorema 3 es parcialmente original.

Teorema 1. (Proposition 2.3.1 (Stein, 2009)). *Si a y b son números enteros, entonces son primos relativos si y solamente si, existen enteros a_0 y b_0 tales que:*

$$1 = aa_0 + bb_0. \qquad (1)$$

Teorema 2. (Theorem 9.1 (Silverman, 2006)). *Si p es un número primo y a es un entero no divisible por p, entonces:*

$$a^{p-1} \equiv 1 (mód\ p). \qquad (2)$$

Teorema 3. *Sean p y q números primos diferentes y $n = pq$. Si e y d son enteros positivos tales que $ed \equiv 1 (mód\ \varphi(n))$, entonces $x^{ed} \equiv x (mód\ n)$ para todo número entero x.*

Demostración. Sea x un entero. La hipótesis $ed \equiv 1 (mód\ \varphi(n))$ y el Teorema 1 implican que existe un entero t tal que $ed = 1+\varphi(n)t$. De esta ecuación se tiene: $x^{ed} = xx^{\varphi(n)t}$. Para terminar la demostración debemos probar que:

$$xx^{\varphi(n)t} \equiv x (mód\ pq). \qquad (3)$$

Como p y q son primos, (3) equivale a $xx^{\varphi(n)t} \equiv x (mód\ p)$ y $xx^{\varphi(n)t} \equiv x (mód\ q)$. Demostraremos cada una de las últimas congruencias; de hecho, basta demostrar una, la otra sigue exactamente la misma línea de argumentos, cambiando el primo en cuestión. Si p divide a x entonces $xx^{\varphi(n)t} \equiv x (mód\ p)$ se obtiene directamente. Si p no divide a x y como $\varphi(n) = (p-1)(q-1)$, entonces aplicando el Teorema 2 se obtiene la conclusión.

Nota 1 El Teorema 3, base del sistema de encriptamiento *RSA*, es válido para cualquier entero positivo n que se represente como producto de primos diferentes, es decir, si $n = p_1p_2 \cdots p_k$, con $p_i \neq p_j$ para $i \neq j$, entonces las hipótesis del teorema llevan a la misma conclusión. En aspectos de seguridad, debe mencionarse que si $n = pq$, conocer sus factores primos equivale a conocer a n y a $\varphi(n)$; esto se debe a que de la igualdad $\varphi(n) = (p-1)(q-1) = n-(p+q)+1$ se obtiene $p+q = n-\varphi(n)+1$. Ahora hay que notar que p y q son raíces de la ecuación $x^2-(p+q)x+pq = 0$. Si n tiene más de dos factores primos, este procedimiento no funciona.

Nota 2 Para fines prácticos, el método de encriptamiento RSA, sustentado en el Teorema 3, debe pasar por un proceso de "etiquetado" o codificación; lo cual esencialmente consiste en representar el texto que se desea encriptar mediante una sucesión de números, cada uno menor que $n = pq$. También debe mencionarse que la elección de los primos p y q, así como el exponente e siguen un proceso aleatorio con la finalidad de garantizar niveles de seguridad altos. Con la notación del Teorema 3, a la pareja (n, e) se le llama llave pública, que es la información para cifrar mensajes, mientras que la pareja (n, d) se le llama llave privada, la cual es confidencial y es lo que se requiere para descifrar los mensajes. Recuérdese que d se calcula usando la condición $ed \equiv 1 (mód\ \varphi(n))$.

El siguiente ejemplo ilustra en alguna medida el uso "práctico" del método de cifrado RSA; sin embargo, el uso que se da en las múltiples aplicaciones requiere de protocolos elaborados que lo hagan funcional, seguro y práctico; lo cual es parte de lo que se discute en el resto del trabajo.

Ejemplo En este ejemplo ilustraremos como se puede enviar el mensaje REYES MAGOS, utilizando el sistema RSA. Para tal efecto, usaremos los números primos $p = 189234919$ y $q = 189234971$. A partir de esto se obtiene $n = pq = 35809864409152349$ y $\varphi(n) = 35809864030682460$ que son elementos importantes en el proceso de cifrado. La llave pública es la pareja (e, n), con $e = 7$.

Los cálculos para desarrollar este ejemplo fueron realizados usando el sistema de cómputo SageMath (Stein, 2020), que dispone de varias herramientas de gran utilidad en el proceso de calcular. Para llevar a cabo el cifrado de "REYES MAGOS", usaremos el alfabeto en español y un caracter blanco adicional para separar palabras; procedemos a codificar los caracteres del alfabeto de la siguiente manera. El caracter blanco es 0, A es 1, B es 2, etc.; como el alfabeto en español tiene 27 letras, al agregar el caracter blanco resulta que el nuevo alfabeto tiene 28 caracteres; por esta razón, el texto REYES MAGOS se codifica en base 28 como:

$$\text{REYES MAGOS} \longleftrightarrow 19\cdot28^{10} + 5\cdot28^9 + 26\cdot28^8 + 5\cdot28^7 + 20\cdot28^6 + 0\cdot28^5 + 13\cdot28^4 + 1\cdot28^3 + 7\cdot28^2 + 16\cdot28 + 20 = 5690530809403908 \text{ (representación en base 10).}$$

De forma general, para cifrar el mensaje numérico x con la llave dada se procede a calcular $x^7 (mód\ 35809864409152349)$, lo cual se traduce a: $5690530809403908^7 \equiv$

$6638698648470078 (mód\ 35809864409152349)$, es decir, el "texto numérico" cifrado es 6638698648470078.

Para representar este mensaje en términos de los caracteres del alfabeto, procederemos a representarlo en base 28, el cual resulta ser $6638698648470078 = 22\cdot28^{10} + 11\cdot28^9 + 15\cdot28^8 + 25\cdot28^7 + 5\cdot28^6 + 0\cdot28^5 + 4\cdot28^4 + 26\cdot28^3 + 18\cdot28^2 + 14\cdot28 + 22$.

Traducido a los caracteres del alfabeto, el mensaje cifrado se lee como: "UKNXE DYQNU".

Para descifrar el mensaje se requiere resolver la congruencia $7d \equiv 1 (mód\ \varphi(n))$, cuya solución es $d = 20462779446104263$. Debido a que los números utilizados son grandes, uno de los comandos de SageMath que se utilizaron es power_mod(a, n, m), en donde a es la base, n es el exponente y m es el módulo. Usando SageMath se verifica que $6638698648470078^{20462779446104263} \equiv$

$5690530809403908 (mód\ 35809864409152349)$, como era de esperarse.

3. Smartcard

Una tarjeta inteligente o smartcard es un dispositivo que integra de manera interna un microprocesador, memoria RAM, además suele tener algunas aplicaciones preinstaladas y para poder utilizarse necesita interactuar con un lector de tarjetas inteligentes. Para tal fin existen dos tipos de tarjetas inteligentes: de contacto y a distancia. Las tarjetas de contacto deben tener contacto físico directo con el lector de tarjetas para que este pueda interactuar con la misma y realizar las operaciones necesarias. Por otra parte, las tarjetas sin contacto o contactless smartcards, las cuales son descritas en el ISO 7816, Parte 1 (ISO, 2011) y captan una radiofrecuencia de 13.56 MHz cuyo alcance es de a lo más 10 centímetros (ISO, 2006). Otro elemento importante de las tarjetas inteligentes es el protocolo de comunicación (ISO, 2013).

Las tarjetas inteligentes sin contacto son las que utilizaremos en la discusión siguiente; dentro de sus usos más frecuentes se pueden listar:

- Documentos de identidad, tales como pasaportes electrónicos.
- Sistemas de pago, como las tarjetas de crédito.
- Control de tráfico, como las tarjetas de pago que se usan en las carreteras de cuota.

Una tarjeta inteligente almacena en su estructura de directorios y registros de forma segura los certificados digitales, además de llaves públicas y privadas del propietario.

3.1. Certificado Digital

Cuando se transmite información por canales que son vulnerables, es de mucha importancia verificar que los canales, dispositivos y los usuarios sean los autorizados para hacer uso de la infraestructura. Esto motiva a definir lo que es un certificado digital. Un certificado digital es un archivo que contiene una clave electrónica que prueba la autenticidad de un dispositivo, servidor o usuario mediante el uso de un sistema de encriptamiento y la infraestructura de llave pública, que por sus siglas en inglés se escribe PKI (Public Key Infrastructure). El estándar X.509 establece el formato del certificado, el cual contiene el número de serie, el algoritmo con el que se firmó el certificado, fecha de expiración, nombre del emisor, nombre del propietario y llave pública del propietario (M. Zhang, 2013).

3.2. Estructura del sistema de directorios

Una de las grandes ventajas que tienen las tarjetas inteligentes es que cuentan con un sistema de archivos que permite guardar una cantidad considerable de información, según se documenta en el ISO/IEC 7816 Parte 4 (ISO, 2013), el cual está basado en:

- Dedicated file (DF) o archivos dedicados.
- Elementary file (EF) o archivos elementales.

La organización lógica de los datos en la tarjeta consiste de una jerarquía estructural de archivos dedicados. El archivo dedicado en la raíz del sistema se llama master file (MF) o archivo maestro. El archivo maestro es obligatorio mientras que los otros archivos dedicados son opcionales. Existen dos tipos de archivos elementales:

- EF interno, cuyo propósito es almacenar datos interpretados por la tarjeta inteligente.
- EF de trabajo los cuales guardan datos no interpretados por la tarjeta inteligente.

3.2.1. Estructura de archivos elementales.

Existen dos tipos de archivos elementales:

- Estructura transparente.
- Estructura de registro.

Gran parte del trabajo que se realizó, y del cual estamos reportando un segmento; fue para poder trabajar con cualquiera de las formas de archivos elementales existentes. Procederemos a explicar de una manera más extensa cada una de estas formas.

Estructura transparente

Este tipo de estructura en la interfaz es vista como una secuencia de unidades de datos; por lo cual tendremos un solo archivo elemental que estará conformado por varias entradas de información. Por lo general esta información será de un mismo tipo, por ejemplo: certificados, llaves públicas o llaves privadas.

Otro punto importante y quizá el más complicado de todos cuando trabajamos con una estructura transparente es tener que eliminar una entrada de información. Esto se debe a que cada entrada de información tiene una cantidad exacta de unidades de datos y para poder eliminarla se necesita identificar la entrada específica, desplazar todas las entradas siguientes la cantidad exacta de unidades de información que ocupaba la entrada a eliminar y después sobrescribir con ceros las unidades de información que se ubican al final del archivo.

Estructura de registro.

Este tipo de estructura en la interfaz es vista como una secuencia de registros identificables individualmente. En comparación con los archivos elementales con estructura transparente, esta es la manera fácil de manejar la información, puesto que cada entrada de información, ya sea certificado o llave, se registra de manera individual. Con lo cual, en el peor de los casos, si se llegara a corromper alguna pieza de información solo se pierde esta y no afecta a las demás.

4. APDU

El Protocolo de Aplicación para la Unidad de Datos o Application Protocol Data Unit, APDU por sus siglas en inglés, es la manera en cómo se realiza la comunicación entre la tarjeta inteligente y el lector de tarjetas inteligentes. Esta comunicación se realiza estrictamente en la capa de bajo nivel y se documenta en el ISO/IEC 7816 Parte 4 (ISO, 2013).

Existen dos tipos de APDU, los comandos y las respuestas. Los comandos están formados por las siguientes unidades de bytes:

Tabla 1: Estructura de un comando APDU.

Obligatorio	CLA	INS	P1, P2
Opcional	Lc	Data	Le

- CLA: 1 byte de clase.
- INS: 1 byte de instrucción.
- P1, P2: 2 bytes de parámetros.
- Lc: 1 byte de tamaño del bloque de datos.
- Data: Entre 0 y 255 bytes de datos a proporcionar.
- Le: Tamaño de la respuesta esperada.

Los primeros 4 bytes correspondientes a CLA, INS, P1 y P2 son obligatorios, el resto son opcionales. Con solo la CLA y la INS la tarjeta inteligente ya sabe la acción que debe realizar; los parámetros P1 y P2 sirven para precisar la operación. Un símil de esto sería a la hora de copiar un archivo en la computadora, la CLA e INS le dicen a la computadora que va a realizar una copia de un archivo, mientras que los parámetros le dicen a la computadora cual es el archivo específico que se quiere copiar.

Los APDU de respuesta están formados por:

Tabla 2: Estructura de una respuesta APDU.

Opcional	Data
Obligatorio	SW1, SW2

- Data: Bytes de datos de respuesta.
- SW1, SW2: 2 bytes dónde se codifica el estado de la operación.

Los bytes de datos son opcionales, mientras que los 2 bytes de estado de la operación son obligatorios; en estos últimos es donde se obtiene un código, el cual indica el estado final al que llegó la instrucción, siendo `90 00` sinónimo de que la instrucción se ejecutó de manera correcta.

4.1. Ejemplos APDU

Veamos ahora algunos ejemplos que nos ayuden a dejar claro lo que es un APDU; para ello usaremos la siguiente notación:

- Se usará >> para indicar que se está ingresando un comando APDU.
- Se usará << para indicar que se recibió una respuesta APDU.

4.1.1. Seleccionar archivo

En la tabla 3 se muestra un comando APDU y su respuesta.

Tabla 3: Ejemplo APDU comando y respuesta.

>>	00 A4 04 0C 0C A0 00 00 00 63 50 4B 43 53 2D 31 35
<<	90 00

Ahora identificamos cada una de las partes que lo componen. Tenemos para el comando APDU:

- CLA: `00`.
- INS: `A4`.
- P1, P2: `04`, `0C`.
- Lc: `0C`.
- Data: `A0 00 00 00 63 50 4B 43 53 2D 31 35`.
- Le: `vacío`.

La respuesta APDU es:

- Data: `vacío`.
- SW1, SW2: `90`, `00`.

La interpretación del comando anterior es que trabaja sobre una clase general que no depende del distribuidor de las tarjetas y le pedimos que seleccione directamente un archivo dedicado, cuyo identificador tiene un tamaño de 12 bytes y es `A0 00 00 00 63 50 4B 43 53 2D 31 35`, que corresponde a un grupo de estándares de criptografía llamado Public-Key Cryptography Standards o PKCS, los cuales fueron concebidos y publicados por los laboratorios RSA. Como el tamaño de la respuesta es vacío, se espera una respuesta con una cantidad desconocida de unidades de datos, los cuales pueden ser vacíos. En la respuesta APDU podemos observar que no se recibió ningún dato y además que el estado de la operación es satisfactorio, por lo cual el archivo fue seleccionado de manera correcta.

4.1.2. Autenticación de PIN

La tabla 4 ilustra un comando APDU y su respuesta.

Tabla 4: Ejemplo APDU comando y respuesta.

>>	00 20 00 83 0A 31 31 31 31 31 31 31 31 FF FF
<<	90 00

Ahora identificamos cada una de las partes que lo componen. Tenemos para el comando APDU:

- CLA: `00`.
- INS: `20`.
- P1, P2: `00`, `83`.
- Lc: `0A`.
- Data: `31 31 31 31 31 31 31 31 FF FF`.
- Le: `vacío`.

La respuesta APDU es:

- Data: `vacío`.
- SW1, SW2: `90`, `00`.

La interpretación de este comando es: se realizará una operación de autenticación sobre una clase general, que no depende del distribuidor de las tarjetas con un PIN de usuario el cual tiene un tamaño de 10 unidades de datos, los primeros 8 de estos datos son 1, los últimos 2 son 255 que dentro del ISO 7816 indica que están vacíos. Como el tamaño de la respuesta es vacío, se espera una respuesta con una cantidad

desconocida de unidades de datos, los cuales pueden ser vacíos. En la respuesta APDU observamos que los datos son vacíos y que el estado de la operación es satisfactorio, por lo que el PIN proporcionado por el usuario es correcto, con lo cual habilita la tarjeta para realizar las siguientes instrucciones.

4.1.3. Seleccionar llave privada

En la tabla 5 se muestra un comando APDU y su respuesta.

Tabla 5: Ejemplo APDU comando y respuesta.

>>	00 22 41 B8 06 80 01 0C 84 01 41
<<	90 00

Ahora identificamos cada una de las partes que lo componen. Tenemos para el comando APDU:

- CLA: `00`.
- INS: `22`.
- P1, P2: `41`, `B8`.
- Lc: `06`.
- Data: `80 01 0C 84 01 41`.
- Le: `vacío`.

La respuesta APDU es:

- Data: `vacío`.
- SW1, SW2: `90`, `00`.

El comando APDU anterior trabajará sobre una clase general, cuya instrucción es realizar una operación con el ambiente de seguridad criptográfico de la tarjeta, la cual, es habilitar el ambiente de seguridad y además seleccionar el número de llave privada que queremos utilizar para realizar el proceso de firma más adelante. En este punto es importante mencionar que la numeración de los identificadores de las llaves privadas inicia en el número 41 y va aumentando de uno en uno, de tal manera que, si se quisiera usar la séptima llave privada guardada en la tarjeta, el identificador que le corresponde es el número 47; el identificador de la llave viene en el sexto byte de los datos. En el comando anterior al tener en esta posición el número 41, se selecciona la primera de las llaves existentes en la memoria de la tarjeta. Como el tamaño de la respuesta es vacío, se espera una respuesta con una cantidad desconocida de unidades de datos, los cuales pueden ser vacíos. En la respuesta APDU observamos que no obtuvimos datos de respuesta y el estado de la operación indica que ésta fue correcta; por lo que se habilitó correctamente el ambiente de seguridad y se seleccionó la llave deseada.

4.1.4. Firma

Una vez que ya tenemos la llave privada seleccionada, veamos ahora el comando APDU y su respuesta que nos permiten realizar la firma electrónica.

Tabla 4: Ejemplo APDU comando y respuesta.

>>	00 2A 80 86 00 01 01 00 00 01 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 30 31 30 0D 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 66 8E 7F 16 63 F8 19 DB 7D 47 85 66 86 BD 00 5D 32 FC CD CA F1 0D F5 F1 E1 79 2C E7 A6 22 24 D2 00 00
<<	2B FA 33 71 FE 25 D7 0B 97 20 5C 0A 7B BE 5C C1 9C 7A 7F 8E 92 59 25 F0 C7 C3 97 C2 BE 43 E0 3F 4D 17 02 20 37 8B C9 E8 D0 5C 1D BD 16 79 1F E7 D6 54 71 91 DA 5E 40 F7 9B 9B B9 59 37 03 E4 BA F5 DB 36 4A AC 79 36 FF 2B 09 F2 4D AE 14 C6 E5 13 22 A0 E4 DE 01 EA 55 35 C6 A8 5D A9 AB 79 85 9A 71 A7 7F AB DD 9F 14 22 26 A6 FF A6 F4 14 E0 00 1A A8 51 C2 BC 6B 0B BD 94 51 48 0B 65 71 2C 06 BF D9 8B 06 3C 13 BA AA 0F EC BC 5B 9C 8D FC E7 5C 4E 8D BC 82 53 8F E0 3A 2B 76 E7 A3 61 33 15 99 4F A7 D1 FF CB DC A7 1E FF 27 4D FA 5A AF 9F BA AE B6 C6 14 79 54 6F E3 CA D7 F0 32 E4 06 3E 18 24 EE 98 77 39 49 9B AD 6D A4 DA 56 27 64 DE FC B5 F6 A1 5F 10 D0 C8 C6 23 31 5E 26 3D E0 B2 FF 1A 1E 06 29 1E AD 1A 0E 21 90 58 1C FA C2 AC C0 B2 90 ED AE 5C 34 BE D5 07 0E 99 0A E4 8E 90 00

Ahora identificamos cada una de las partes que lo componen. Tenemos para el comando APDU:

- CLA: `00`.
- INS: `2A`.
- P1, P2: `80`, `86`.
- Lc: `00 01 01 00`.
- Data:
```
01 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF 00 30 31 30 0D
06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 66
8E 7F 16 63 F8 19 DB 7D 47 85 66 86 BD 00 5D 32
FC CD CA F1 0D F5 F1 E1 79 2C E7 A6 22 24 D2 00
00
```
- Le: `vacío`.

Tenemos para la respuesta APDU:

- Data:
```
2B FA 33 71 FE 25 D7 0B 97 20 5C 0A 7B BE 5C C1
9C 7A 7F 8E 92 59 25 F0 C7 C3 97 C2 BE 43 E0 3F
4D 17 02 20 37 8B C9 E8 D0 5C 1D BD 16 79 1F E7
D6 54 71 91 DA 5E 40 F7 9B 9B B9 59 37 03 E4 BA
F5 DB 36 4A AC 79 36 FF 2B 09 F2 4D AE 14 C6 E5
13 22 A0 E4 DE 01 EA 55 35 C6 A8 5D A9 AB 79 85
9A 71 A7 7F AB DD 9F 14 22 26 A6 FF A6 F4 14 E0
00 1A A8 51 C2 BC 6B 0B BD 94 51 48 0B 65 71 2C
06 BF D9 8B 06 3C 13 BA AA 0F EC BC 5B 9C 8D FC
E7 5C 4E 8D BC 82 53 8F E0 3A 2B 76 E7 A3 61 33
```

```
15 99 4F A7 D1 FF CB DC A7 1E FF 27 4D FA 5A AF
9F BA AE B6 C6 14 79 54 6F E3 CA D7 F0 32 E4 06
3E 18 24 EE 98 77 39 49 9B AD 6D A4 DA 56 27 64
DE FC B5 F6 A1 5F 10 D0 C8 C6 23 31 5E 26 3D E0
B2 FF 1A 1E 06 29 1E AD 1A 0E 21 90 58 1C FA C2
AC C0 B2 90 ED AE 5C 34 BE D5 07 0E 99 0A E4 8E
```

- SW1, SW2: `90`, `00`.

El comando APDU anterior es mucho más complicado que los primeros ejemplos que mostramos, por lo cual lo explicaremos de manera más detallada. Empecemos por la clase que es de tipo general, la instrucción será realizar una operación de seguridad en la cual los parámetros sirven para definir que los bytes pertenecientes al tamaño de los datos, así como los datos, tendrán una configuración diferente en este caso particular. El tamaño de los datos que usualmente se especifica usando un byte ahora se describirá usando dos bytes, los cuales se encuentran descritos entre dos bytes que contienen 00, el tamaño de los datos se expresa en hexadecimal, por lo que al hacer la conversión a decimal obtenemos que 01 01 = 256 + 1 = 257 bytes de datos. Los bytes 00 01 se usan para indicar el inicio del mensaje codificado usado para generar una firma, según lo establecido en la documentación del algoritmo RSA (Moriarty et al., 2016). Los datos en este caso están divididos en tres bloques. Primero encontraremos un bloque de 202 bytes, los cuales contienen FF codificando el relleno requerido por el algoritmo, el tamaño de este bloque se determina dependiendo del tamaño de los bloques siguientes y explicaremos como se obtiene una vez que terminemos de listar todos los bloques que componen los datos. Después de los 202 bytes con FF encontramos el segundo bloque de datos, que inicia después del byte que contiene 00, son 19 bytes correspondientes a la función hash criptográfica SHA-256 en donde se especifica que trabajaremos con una firma RSA de 2048 bits, por lo que el APDU de respuesta tendrá 256 bytes de datos, esto lo sabemos a pesar de que el byte correspondiente a Le, se encuentre vacío. El siguiente bloque de datos comprende los 32 bytes siguientes y pertenecen al hash del archivo que vamos a firmar, este hash tiene siempre esa longitud sin importar si el archivo tiene un tamaño variable. Al terminar con los bloques de datos anteriores se agregan 2 bytes con 00 para completar el tamaño previamente definido de los datos y que indican que se espera un número indeterminado de bytes como respuesta. Con los datos previos procedemos a explicar el tamaño del bloque vacío; por definición sabemos que hay 257 bytes de datos totales, 32 de los cuales pertenecen al hash, 19 pertenecen a la función criptográfica, 3 que corresponden a los bytes que contienen 00 y 1 byte que corresponde al indicador de inicio de datos que contiene 01. Si sumamos tendremos 32+19+3+1 = 55, entonces 257-55 = 202 bytes de datos que corresponden al relleno del algoritmo. Por otro lado, los bytes que obtenemos en la respuesta APDU tienen una longitud de 256 bytes que corresponde con el tamaño que se definió en la función criptográfica; estos bytes son la firma electrónica del archivo deseado. El estado de la operación nos dice que todo se realizó de manera correcta.

4.2. Implementación del proceso de firma.

El proceso de firma requiere acciones previas de inicialización y verificación de la tarjeta para poder tener acceso al componente de firma. La primera parte del proceso es seleccionar el formato de los directorios que en el caso de las tarjetas inteligentes es el PKCS15 (RSA Laboratories, 1999), una vez seleccionado el formato de archivos verificaremos si el usuario tiene los permisos para hacer uso de la tarjeta. Para este proceso se envía el número de identificación personal del usuario; en caso de que el PIN sea el correcto podemos leer el sistema de archivos de la tarjeta. La llave pública y privada del usuario deberá estar almacenada en la tarjeta, pues será necesaria para realizar la firma, la cual toma como parámetros la llave privada, el arreglo de bytes a firmar y retorna la firma correspondiente. El siguiente diagrama muestra los pasos mínimos para realizar el proceso de firma:

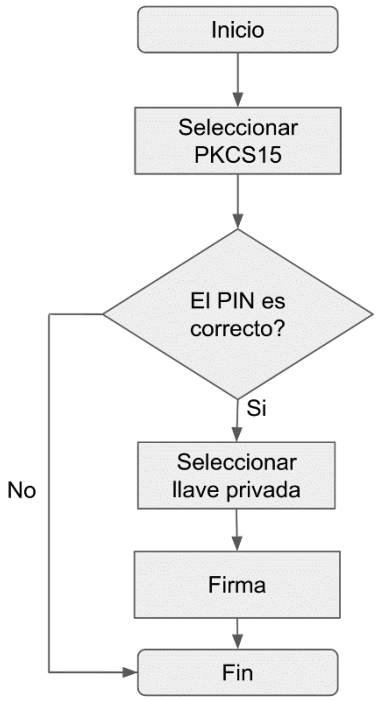

Figura 1: Proceso de firma digital.

La implementación (en lenguaje C++) de cada uno de estos pasos y el proceso de transmisión de los APDU se muestra en el apéndice A.

5. Conclusiones

En este trabajo se han desarrollado algunas ideas importantes en el proceso que se lleva a cabo, usando el sistema de encriptamiento RSA, para firmar digitalmente. Durante el desarrollo del mismo surgieron varias ideas que abren nuevas rutas para mejorar la administración dinámica de los archivos que contiene la tarjeta inteligente. Esto con la finalidad de elevar los rangos de seguridad y hacerlas funcionales para su uso cotidiano. Es importante mencionar que la transmisión del PIN y el mensaje, mediante el protocolo usado en este proyecto para firma electrónica, se envían sin encriptar, lo que tiene un rango de vulnerabilidad.

Una posible forma de continuar el presente trabajo, es desarrollar un protocolo que tenga por finalidad garantizar la seguridad al enviar el PIN y el mensaje mediante el software.

Consideramos que la firma electrónica es una aplicación de diversos conceptos y métodos que se aprenden en una licenciatura en Ciencias Computacionales y/o en Matemáticas Aplicadas. Por esto, creemos que el trabajo que estamos presentando puede ser de interés para aquellos estudiantes que desean conocer posibles aplicaciones reales de los conocimientos que están desarrollando.

Agradecimientos

Los autores agradecen al Ing. Juan González Cardozo, subdirector de Innovación en SeguriData Privada S.A de C. V. por sus comentarios y disponibilidad.

También el agradecimiento se extiende a los revisores del trabajo, cuyas observaciones contribuyeron a mejorarlo.

Referencias

Chirico, U. (2014). Smart card programming, second edition. Lulu.com. ISBN: 1291610502.

Conrad, K. Number theory and cryptography. Disponible en línea en: https://kconrad.math.uconn.edu/blurbs/ugradnumthy/RSAnotes.pdf

Dugundji, J. (1966). Topology. Editorial Allyn and Dacon, Inc.

ISO, 2006. Identification cards - Integrated circuit cards - Part 3: Cards with contacts - Electrical interface and transmission protocols. ISO/IEC 7816-1:2006.

ISO, 2011. Identification cards - Integrated circuit cards - Part 1: Cards with contacts - Physical characteristics. ISO/IEC 7816-1:2011.

ISO, 2013. Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange. ISO/IEC 7816-4:2013.

Koblitz, N. (1987). A course in number theory and cryptography. Springer.

Moriarty K. M., Kaliski B., Jonsson J., Rusch A. (2016). PKCS #1: RSA Cryptography Specifications Version 2.2. RFC Editor. DOI 10.17487/RFC8017

Rivest, R. L., Shamir, A., Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems, comm. Available online at http://people.csail.mit.edu/rivest/ Rsapaper.pdf

RSA Laboratories (1999). PKCS #15 v1.1: Cryptographic token information format standard.

Silverman, J. H. (2006). A friendly introduction to number theory. Third edition. Pearson Prentice Hall.

Stein, W. A. (2020). SageMath. Available online at https://www.sagemath.org/

Stein, W. (2009). Elementary Number Theory: Primes, Congruences, and Secrets, A computational approach. Springer.

Zhang, M. (2013). Secure digital certificate design based on the public key cryptography algorithm. DOI: 10.11591/telkomnika.v11i12.3824

Apéndice A. Implementación de funciones principales del proceso de firma electrónica.

Transmisión de comandos APDU

La comunicación con el lector de tarjetas se realiza con la ayuda de la biblioteca winscard.h, la cual está incluida en Windows, Linux y OSX. El método principal para hacer la transmisión de datos es el comando SCardTransmit, que recibe el APDU a ser enviado y un apuntador hacia un arreglo de bytes que almacenará la respuesta, además de un entero que es pasado por referencia en el cual se almacena la longitud total de la respuesta.

Se muestra en el siguiente bloque de código una implementación completa de la función de transmisión:

```
bool Transmit(BYTE* command, std::vector<BYTE>& response,
size_t commandLength, uint32_t& responseLength) {

   SCARD_IO_REQUEST pioSendPci;
   switch (dwActiveProtocol) {
   case SCARD_PROTOCOL_T0:
       pioSendPci = *SCARD_PCI_T0;
       break;
   case SCARD_PROTOCOL_T1:
       pioSendPci = *SCARD_PCI_T1;
       break;
   }
   SCardBeginTransaction(hCard);
   DWORD responseRawLength = 512;
   BYTE responseRaw[512];
   for (int i = 0; i < 512; ++i) {
       responseRaw[i] = 0;
   }
   SC_RESPONSE res = SCardTransmit(hCard, &pioSendPci,
                   command, commandLength, NULL,
responseRaw,
                   &responseRawLength);
   responseLength = responseRawLength;

   for (unsigned int i = 0; i < responseLength; ++i) {
       response.push_back(responseRaw[i]);
   }

   if (!CheckSuccess(res)) {
       res = SCardEndTransaction(hCard,

SCARD_LEAVE_CARD);
       return false;
   }
   res = SCardEndTransaction(hCard,

SCARD_LEAVE_CARD);
   return true;
}
```

Seleccionar PKC515

El método SelectPKC15 es el designado para seleccionar el tipo de estructura de sistema de archivos e inicializa la tarjeta para comenzar a ser usada. La implementación de dicha función está divida en dos funciones para su posible reutilización:

```
bool SelectByID(std::vector<BYTE>& rId, BYTE op,
                                BYTE firstRegister) {
     if (rId.size() > 0) {
        std::vector<BYTE> apduSelectFile = { 0x00, 0xA4
};
        apduSelectFile.push_back(op);
        apduSelectFile.push_back(firstRegister);
        apduSelectFile.push_back((BYTE)rId.size());
        apduSelectFile.insert(apduSelectFile.end(),
                              rId.begin(), rId.end());
         apduResponse response;
         if (Transmit(apduSelectFile.data(),
response.code,
                      apduSelectFile.size(),
response.length))  {
             return InterpretResponse(response);
         }
    }
    return false;
}

bool SelectPKCS15()  {
   return SelectByID({ 0xA0,0x00,0x00,0x00,0x63,0x50,
                       0x4B,0x43,0x53,0x2D,0x31,0x35 },
0x04);
}
```

Autenticación de PIN

La autenticación de PIN es un proceso en el que se revisa la longitud del PIN, el cual puede ser de longitud mínima de cuatro bytes con un máximo de 10 bytes, en esta función se arma el APDU correspondiente y se transmite a la tarjeta.

```
bool AuthenticatePin(const std::string &rPin, int
*errCode, TypeUser typeUser) {
    std::vector<BYTE> apduVerifyPin = { 0x00,0x20,0x00 };
    if (typeUser == TypeUser::user)  {
        apduVerifyPin.push_back(0x83);
    }
    else if (typeUser == TypeUser::systemUser)  {
        apduVerifyPin.push_back(0x84);
    } else {
        return false;
    }
    apduVerifyPin.push_back(0x0A);
    for (size_t i = 0; i < 10; ++i)  {
        if (i < rPin.length()) {
            apduVerifyPin.push_back(rPin[i]);
        }
        else {
            apduVerifyPin.push_back(0xFF);
        }
    }
    char *code;
    bool response = false;
    int apduSize = apduVerifyPin.size();
    if (Transmit(apduVerifyPin.data(), code,
        apduSize, response.length))  {
        response = true;
    }
    if (errCode && response.length == 2) {
       *errCode =code[0] * 0x100 +code[1];
    }
    return response;
}
```

Firma

El método de firma que se muestra a continuación solo implementa una firma de 2048 bits, se arma el APDU de firma y agrega la longitud de la cadena a firmar y transmite el APDU, para finalmente retornar la respuesta por medio del parámetro rSignedHash el cual modifica la memoria al ser una variable que se pasa por referencia, además retorna un valor verdadero en la salida de la función en caso de que el proceso de firma sea correcto.

```
bool SignHash(const std::string& rHash, std::string&
rSignedHash, HashAlgorithm hashAlgorithm, KeySize
privateKeySize) {
    std::vector<BYTE> sha2IOD =
{0x30,0x31,0x30,0x0D,0x06,0x09,0x60,
                                0x86,0x48,0x01,0x65,0x03
,0x04,0x02,
                                0x01,0x05,0x00,0x04,0x20
};
    if (hashAlgorithm == HashAlgorithm::SHA256) {
        sha2IOD[1]  = 0x31;
        sha2IOD[14] = 0x01;
        sha2IOD[18] = 0x20;
    } else {
        printError("SignHash algorithm is not
supported");
        return false;
    }
    unsigned int blockSize;
    for (auto part : rHash) {
        sha2IOD.push_back(part);
    }
    blockSize = (privateKeySize >> 3) - 3 -
sha2IOD.size();

    std::vector<BYTE> blockToEncrypt = { 0x00, 0x01 };
    for (unsigned int i = 0; i < blockSize; ++i) {
        blockToEncrypt.push_back(0xFF);
    }
    blockToEncrypt.push_back(0x00);
    blockToEncrypt.insert(blockToEncrypt.end(),
                          sha2IOD.begin(),
sha2IOD.end());
   int paramLen = blockToEncrypt.size() + 1;

    std::vector<BYTE> apduSignHash;
    apduSignHash = { 0x00,0x2A,0x80,0x86 };



    if (paramLen <= 0xFF) {
        apduSignHash.push_back((BYTE)paramLen);
    } else {
        apduSignHash.push_back(0x00);
        apduSignHash.push_back((paramLen>>8)&0xFF);
        apduSignHash.push_back(paramLen & 0xFF);
    }
    apduSignHash.push_back(0x00);
    apduSignHash.insert(apduSignHash.end(),
                        blockToEncrypt.begin(),
blockToEncrypt.end());
    apduSignHash.push_back(0x00);
    if (blockToEncrypt.size() > 0xFF) {
        apduSignHash.push_back(0x00);
    }
    apduResponse response;
    if (Transmit(apduSignHash.data(), response.code,
                 apduSignHash.size(), response.length)) {
        if (response.length == 2) {
            return InterpretResponse(response);
        }
        else if (response.length > 2) {
            for (unsigned long i = 0; i < response.length
- 2; ++i) {
                rSignedHash.push_back(response.code[i]);
            }
            return InterpretResponse(response);
        }
    }
    return false;
}
```