UNIVERSIDAD TECNOLOGICA DE PANAMA
FACULTAD DE INGENIERIA DE SISTEMAS COMPUTACIONALES
LICENCIATURA EN DESARROLLO DE SOFTWARE

SEGURIDAD EN LOS SISTEMAS DE INFORMACIÓN

PROYECTO FINAL
CRIPTOGRAFÍA EN PHP

Profesor:
Aizprúa, Saulo

Presentado por:
Andrade, Oscar
Estrada, Demetrio
Lau, Francisco
Zamorano A., Corina

Grupo: 1LS241

Miércoles 16 de julio de 2014

Índice

Introducción

A través de la criptografía la información puede ser protegida contra el acceso no autorizado, su interceptación, su modificación y la inserción de información extra.

También puede ser usada para prevenir el acceso y uso no autorizado de los recursos de una red o sistema informático y para prevenir a los usuarios la denegación de los servicios a los que sí están permitidos.

Modernamente, la criptografía es la metodología para proveer la seguridad de las redes telemáticas, incluyendo la identificación de entidades y autenticación, el control de acceso a los recursos, la confidencialidad de los mensajes transmitidos, la integridad de los mensajes y su no repudio.

Objetivo General

- Conocer los diferentes métodos de criptografía utilizando los diferentes algoritmos de cada uno de ellos en la implementación de los mismos en las diferentes herramientas de Desarrollo de Software.

Objetivos Específicos

- Conocer los conceptos generales de la criptografía y sus dos principales tipos.
- Describir y analizar los algoritmos criptográficos más conocidos.
- Identificar las ventajas y desventajas entre criptografía simétrica y asimétrica

Contenido:

A. Criptografía

1. ¿Qué es Criptografía?

La criptografía es la práctica y el estudio de la ocultación de información. En otras palabras, es el arte o ciencia de cifrar y descifrar cierta información mediante técnicas especiales y se emplea frecuentemente para permitir un intercambio de mensajes que sólo puedan ser leídos por personas a las que van dirigidos y que poseen los medios para descifrarlos.

La criptografía que nosotros conocemos y utilizamos, data de los años 70 aunque ya se utilizaban técnicas criptográficas en la segunda guerra mundial e incluso pueblos de la antigüedad como los Griegos y Romanos utilizaban mecanismos para enviar mensajes “cifrados” en sus campañas militares.

Existen muchos ámbitos en los que hoy en día la criptografía aporta seguridad en la comunicación por canales no seguros, ya sea en comunicaciones militares, protocolo https, envío de correos electrónicos cifrados o las telecomunicaciones en general.

La información original que debe protegerse se denomina texto plano o texto en claro. El cifrado es el proceso de convertir el texto plano en un texto imposible de leer llamado texto cifrado. Para obtener un texto cifrado, se aplica un algoritmo de cifrado, utilizando una clave, al texto plano.

Cifrado de datos

De la misma manera, si aplicamos un algoritmo de descifrado, que también utiliza una clave, al texto cifrado, obtendremos, de nuevo, el texto plano.

Descifrado de datos

2. Criptografía Simétrica

2.1. Descripción

En la criptografía simétrica se usa una única clave para cifrar y descifrar mensajes. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la clave a usar. Una vez compartida la clave, el remitente cifra un mensaje con ella, lo envía al destinatario y éste lo descifra con la misma.

En las siguientes imágenes se puede observar el funcionamiento del intercambio de claves y de cifrado simétrico:

El emisor envía al receptor la clave con la cifrar y descifrar los mensajes a través de un canal seguro.

Criptografía simétrica. Intercambio de claves.

El emisor envía el mensaje cifrado al receptor. Éste último descifra el mensaje para ver su contenido.

Criptografía simétrica. Cifrado/Descifrado.

Actualmente, los ordenadores pueden descifrar claves con extrema rapidez, y ésta es la razón por la cual el tamaño de la clave es importante en los criptosistemas modernos. El algoritmo de cifrado DES usa una clave de 56 bits, lo que significa que hay 2 elevado a 56 claves posibles. Esto representa un número muy alto de claves, pero un ordenador genérico puede comprobar el conjunto posible de claves en cuestión de días. Una máquina especializada puede hacerlo en horas. Algoritmos de cifrado de diseño más reciente como Triple-DES o Blowfish pueden usar claves de tamaños comprendidos entre los 128 bits y los 256 bits, lo que significa que existen 2 elevado a 128/256 claves posibles.

El principal problema con los sistemas de cifrado simétrico está ligado al intercambio de claves. Ningún canal de comunicación es lo bastante seguro como para garantizar que nadie interceptará el mensaje en el que se envía la clave.

Otro problema es el número de claves que se necesitan. Para que un número n de personas se comuniquen entre sí, son necesarias n-1 claves por cada persona, lo que implica que son necesarias n * ((n - 1) / 2) claves en total. Por ejemplo, para un grupo de 4 personas serían necesarias 6 claves, pero para un grupo de 100 personas serían necesarias 4950 claves. Por lo tanto, esto puede funcionar con un grupo reducido de personas, pero sería inviable llevarlo a cabo con grupos más grandes.

2.2. Algoritmos de criptografía simétrica

- **DES (Data Encryption Standard)**

El cifrado DES es un cifrado por bloques, este cifrado toma un mensaje y lo separa en bloques de un tamaño de bits determinado (en caso del DES el tamaño del bloque es de 64 bits), y transforma este mensaje en un criptograma tras una serie de complicadas operaciones, dando como resultado un criptograma de la misma longitud que el mensaje.

En el cifrado DES, se usa una clave de 64 bits para cifrar los bloques, aunque en realidad sólo se usan 56 bits. Los 8 bits restantes de la clave son usados para comprobar la paridad y después son descartados.

- **Triple-DES**

En criptografía, tipo de algoritmo que realiza un triple cifrado tipo DES, esto lo hace muchísimo más seguro que el cifrado DES simple. Triple DES fue desarrollado por IBM en el año 1998.

El Triple DES, no es un cifrado múltiple, pues no son independientes todas las subclases. Esto es porque DES tiene la propiedad matemática de no ser un grupo; esto implica que si se cifra el mismo bloque dos veces, con dos llaves distintas, se aumenta el tamaño efectivo de la llave.

TDES nació por la inseguridad que traía una clave de 56 bits. Las claves de 56 bits eran posible de decifrar utilizando un ataque de fuerza bruta. El TDES agrandaba el largo de la llave, sin necesidad de cambiar de algoritmo cifrador.

El método de cifrado TDES desaparece progresivamente, siendo reemplazado por el algoritmo AES que es considerado mucho más rápido (hasta 6 veces más rápido). De todas maneras, algunas tarjetas de créditos y otros métodos de pago electrónico, todavía tienen como estándar el algoritmo Triple DES.

- **AES (Advanced Encryption Standard)**

También conocido como Rijndael. Esquema de cifrado por bloques, que fue adoptado como estándar de cifrado por el gobierno estadounidense. Reemplaza progresivamente a su predecesor (DES y Triple DES). AES es uno de los algoritmos más utilizados en criptografía simétrica.

Fue anunciado el 26 de noviembre de 2001 por el NIST (Instituto Nacional de Estándares y Tecnología), luego de un proceso de estandarización que duró 5 años. Se transformó en estándar el 26 de mayo de 2002.

El cifrador fue desarrollado por Joan Daemen y Vincent Rijmen, dos criptólogos de Bélgica, estudiantes de la Universidad Católica de Leuven.

AES es una red de sustitución-permutación, no una red de Feistel (como DES). AES también es mucho más rápido que DES, tanto en hardware como en software y además, requiere poca memoria.

- **Blowfish**

En criptografía, Blowfish es un codificador de bloques simétricos, diseñado por Bruce Schneier en 1993 e incluido en un gran número de conjuntos de codificadores y productos de cifrado. No se han encontrado técnicas de criptoanálisis efectivas contra el blowfish. Sin embargo, se ha dado más atención de la decodificación de bloques con bloques más grandes, como AES y Twofish.

Schneier diseñó Blowfish como un algoritmo de uso general, que intentaba reemplazar al antiguo DES y evitar los problemas asociados con otros algoritmos. Al mismo tiempo, muchos otros diseños eran propiedad privada, patentados o los guardaba el gobierno. Schneier declaró “Blowfish no tiene patente, y así se quedará en los demás continentes. El algoritmo está a disposición del público, y puede ser usado libremente por cualquiera

3. Criptografía Asimétrica

3.1. Descripción

La criptografía asimétrica usa dos claves para el envío de mensajes: una clave pública y una clave privada.

Cada usuario tiene una clave pública y una privada asociadas a él. El usuario debe mantener en secreto la privada y distribuir la pública a todos los receptores con los que desea comunicarse.

Los métodos criptográficos garantizan que esa pareja de claves sólo se pueda generar una vez, de modo que se puede asumir que no es posible que dos personas hayan obtenido casualmente la misma pareja de claves.

En este sistema, lo que cifra la clave pública sólo puede ser descifrado con la privada y lo que cifra la clave privada sólo lo descifra la pública.

El procedimiento consiste en que el emisor cifra los datos con la clave pública del receptor, de esta forma se garantiza la confidencialidad del mensaje ya que sólo el receptor puede descifrarlo con su clave privada.

En las siguientes imágenes se puede observar el funcionamiento cifrado y descifrado en la criptografía asimétrica:

El emisor envía al receptor el mensaje cifrado con la clave pública del receptor. El receptor descifra el mensaje con su clave privada para así poder ver el mensaje en claro.

Criptografía asimétrica de emisor a receptor. Cifrado/Descifrado.

De igual manera, el receptor cifra el mensaje con la clave pública del receptor. El emisor descifra el mensaje con su clave privada.

Criptografía asimétrica de receptor a emisor. Cifrado/Descifrado.

La criptografía asimétrica es la base para realizar operaciones de autenticación y firma electrónica.

En este tipo de criptografía el tamaño de las claves es muy importante ya que toda la seguridad recae en la complejidad de las claves y en la imposibilidad de obtener la clave privada a partir de la pública. Por este motivo el tamaño de las claves es importante, cuanto más largas, más complejas, mayor dificultad. El tamaño de las claves a calcular puede variar entre los 512 bits y los 4096 bits. En la actualidad se considera como seguras las claves con un tamaño mínimo de 1024 bits.

Como todos los sistemas, la criptografía asimétrica también tiene desventajas, como por ejemplo el tamaño de las claves y el tiempo de proceso. Para una misma longitud de clave y mensaje, se necesita mayor tiempo de proceso que para la criptografía simétrica porque las claves tienen un tamaño mayor que las utilizadas en un sistema de clave simétrica y por la complejidad de los algoritmos.

3.2. Algoritmos de criptografía asimétrica

- **RSA (Rivest, Shamir y Adleman)**

En criptografía, RSA (Rivest, Shamir y Adleman) es un sistema criptográfico de clave pública desarrollado en 1977. Es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente.

La seguridad de este algoritmo radica en el problema de la factorización de números enteros. Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto. Actualmente estos primos son del orden de 10^{200}, y se prevé que su tamaño crezca con el aumento de la capacidad de cálculo de los ordenadores.

Como en todo sistema de clave pública, cada usuario posee dos claves de cifrado: una pública y otra privada. Cuando se quiere enviar un mensaje, el emisor busca la clave pública del receptor, cifra su mensaje con esa clave, y una vez que el mensaje cifrado llega al receptor, este se ocupa de descifrarlo usando su clave privada.

Se cree que RSA será seguro mientras no se conozcan formas rápidas de descomponer un número grande en producto de primos. La computación cuántica podría proveer de una solución a este problema de factorización.

- **DSA (Digital Signature Algorithm)**

DSA (Digital Signature Algorithm, en español Algoritmo de Firma digital) es un estándar del Gobierno Federal de los Estados Unidos de América o FIPS para firmas digitales. Fue un Algoritmo propuesto por el Instituto Nacional de Normas y Tecnología de los Estados Unidos para su uso en su Estándar de Firma

Digital(DSS), especificado en el FIPS 186. DSA se hizo público el 30 de agosto de 1991, este algoritmo como su nombre lo indica, sirve para firmar y no para cifrar información. Una desventaja de este algoritmo es que requiere mucho más tiempo de cómputo que RSA.

- **Diffie-Hellman**

El protocolo criptográfico Diffie-Hellman, debido a Whitfield Diffie y Martin Hellman, (Diffie–Hellman Problem->DHP) es un protocolo de establecimiento de claves entre partes que no han tenido contacto previo, utilizando un canal inseguro, y de manera anónima (no autentificada).

Se emplea generalmente como medio para acordar claves simétricas que serán empleadas para el cifrado de una sesión (establecer clave de sesión). Siendo no autenticado, sin embargo, provee las bases para varios protocolos autenticados.

Su seguridad radica en la extrema dificultad (conjeturada, no demostrada) de calcular logaritmos discretos en un cuerpo finito.

- **ElGamal**

El procedimiento de cifrado/descifrado ElGamal se refiere a un esquema de cifrado basado en problemas matemáticos de logaritmos discretos. Es un algoritmo de criptografía asimétrica basado en la idea de Diffie-Hellman y que funciona de una forma parecida a este algoritmo discreto.

El algoritmo de ElGamal puede ser utilizado tanto para generar firmas digitales como para cifrar o descifrar.

Fue descrito por Taher Elgamal en 1984 y se usa en software GNU Privacy Guard, versiones recientes de PGP, y otros sistemas criptográficos. Este algoritmo no está bajo ninguna patente lo que lo hace de uso libre.

La seguridad del algoritmo se basa en la suposición que la función utilizada es de un sólo sentido y la dificultad de calcular un logaritmo discreto.

El procedimiento de cifrado (y descifrado) está basado en cálculos sobre un grupo cíclico cualquiera G, lo que lleva a que la seguridad del mismo dependa de la dificultad de calcular logaritmos discretos en G.

B. Herramienta Utilizada

1. PHP

   Lenguaje de programación utilizado en el desarrollo del proyecto. Se utilizó una estructura por capas, las cuales son:

   - Vistas: contiene todas las pantallas de la aplicación.
   - Lógica: contiene todas las clases y librerías utilizadas, las cuales realizan todos los cálculos y operaciones de la aplicación.
   - Servicios: establece la conexión entre las vistas y la capa lógica. Las vistas se comunican con los servicios y éstos, utilizan la capa lógica para poder responderle al usuario. Por convención, se utilizó el lenguaje JSON para las comunicaciones.

2. jQuery

   Librería de JavaScript utilizada por las vistas. Sus usos fueron los siguientes:

   - Validar los formularios.
   - Establecer comunicación con la capa de servicios mediante Ajax.
   - Darle animación a los elementos de las pantallas.

C. Capa de Servicios

La capa de servicios se encarga de recibir las solicitudes de la capa de la vista, llamar a las clases encargadas de realizar la tarea solicitada y luego enviarle al usuario la información correspondiente.

```
require_once $_SERVER['DOCUMENT_ROOT'].'/src/config/constantes.php';

      require_once
$_SERVER['DOCUMENT_ROOT'].'/src/server/class.CifradoSimetrico.php';

      require_once
$_SERVER['DOCUMENT_ROOT'].'/src/libs/phpseclib/Math/BigInteger.php';

      require_once
$_SERVER['DOCUMENT_ROOT'].'/src/libs/phpseclib/Crypt/RSA.php';

      require_once
$_SERVER['DOCUMENT_ROOT'].'/src/libs/phpseclib/Crypt/AES.php';

      require_once
$_SERVER['DOCUMENT_ROOT'].'/src/libs/phpseclib/Crypt/TripleDES.php';

      require_once
$_SERVER['DOCUMENT_ROOT'].'/src/libs/phpseclib/Crypt/Random.php';

      require_once                $_SERVER['DOCUMENT_ROOT'].'/src/libs/openpgp-
php/openpgp.php';

      require_once                $_SERVER['DOCUMENT_ROOT'].'/src/libs/openpgp-
php/openpgp_crypt_rsa.php';

      require_once                $_SERVER['DOCUMENT_ROOT'].'/src/libs/openpgp-
php/openpgp_crypt_symmetric.php';



      $post = (object) $_POST;

      $get = (object) $_GET;

      $accion = array_shift($params);
```

```
	switch ($accion) {
		case 'descargar':
			if ($post->descarga == 'llave') {
				$data = $post->llave;
				header("Content-Disposition: attachment; filename=php-
{$post->tipo}.key");
				header('Content-Type: text/plain');
				header('Content-Length: ' . strlen($data));
				header('Connection: close');
				echo $data;
			}
			else {
				$data = base64_decode($post->data);
				header("Content-Disposition: attachment; filename=php-
{$post->tipo}.bin");
				header('Content-Type: application/octet-stream');
				header('Content-Length: ' . strlen($data));
				header('Connection: close');
				echo $data;
			}
			exit();
			break;
```

```
            case 'des-enc':

                  $cs = new CifradoSimetrico('des');

                  $data = $cs->cifrar($post->texto, $post->llave);

                  echo json_encode( array(

                        'code' => 0,

                        'data' => base64_encode($data),

                        'tipo' => 'des',

                        'llave' => $post->llave,

                  ));

                  break;



            case 'des-dec':

                  $cs = new CifradoSimetrico('des');

                  $texto     =     ($post->tipoInput     ==     'binario')     ?
file_get_contents($_FILES['binario']['tmp_name']) : base64_decode($post->texto);

                  $data = $cs->descifrar($texto, $post->llave);

                  echo json_encode( array(

                        'code' => 0,

                        'data' => $data

                  ));

                  break;



            case '3des-enc':

                  $cs = new CifradoSimetrico('3des');
```

```
                    $data = $cs->cifrar($post->texto, $post->llave);
                    echo json_encode( array(
                          'code' => 0,
                          'data' => base64_encode($data),
                          'tipo' => 'triple-des',
                          'llave' => $post->llave,
                    ));
                    break;

             case '3des-dec':
                    $cs = new CifradoSimetrico('3des');
                    $texto     =     ($post->tipoInput     ==     'binario')     ?
file_get_contents($_FILES['binario']['tmp_name']) : base64_decode($post->texto);
                    $data = $cs->descifrar($texto, $post->llave);
                    echo json_encode( array(
                          'code' => 0,
                          'data' => $data
                    ));
                    break;

             case 'blowfish-enc':
                    $cs = new CifradoSimetrico('blowfish');
                    $data = $cs->cifrar($post->texto, $post->llave);
                    echo json_encode( array(
```

```
                        'code' => 0,
                        'data' => base64_encode($data),
                        'tipo' => 'blowfish',
                        'llave' => $post->llave,
                ));
                break;

        case 'blowfish-dec':
                $cs = new CifradoSimetrico('blowfish');
                $texto = ($post->tipoInput == 'binario') ?
file_get_contents($_FILES['binario']['tmp_name']) : base64_decode($post->texto);
                $data = $cs->descifrar($texto, $post->llave);
                echo json_encode( array(
                        'code' => 0,
                        'data' => $data
                ));
                break;

        case 'aes-enc':
                $cs = new CifradoSimetrico('aes'.$post->blockSize);
                $data = $cs->cifrar($post->texto, $post->llave);
                echo json_encode( array(
                        'code' => 0,
                        'data' => base64_encode($data),
```

```
                        'tipo' => 'aes-'.$post->blockSize,
                        'llave' => $post->llave,
                ));
                break;

        case 'aes-dec':
                $cs = new CifradoSimetrico('aes'.$post->blockSize);
                $texto = ($post->tipoInput == 'binario') ?
file_get_contents($_FILES['binario']['tmp_name']) : base64_decode($post->texto);
                $data = $cs->descifrar($texto, $post->llave);
                echo json_encode( array(
                        'code' => 0,
                        'data' => $data
                ));
                break;

        case 'dsa-enc':
                openssl_public_encrypt($post->texto, $data, DSA_PUBLIC_KEY);
                echo json_encode( array(
                        'code' => 0,
                        'data' => base64_encode($data),
                        'tipo' => 'dsa',
                        'llave' => DSA_PUBLIC_KEY,
                ));
```

```
                    break;



            case 'dsa-dec':

                    $texto     =     ($post->tipoInput     ==     'binario')     ?
file_get_contents($_FILES['binario']['tmp_name']) : base64_decode($post->texto);

                    openssl_private_decrypt($texto, $data, DSA_PRIVATE_KEY);

                    echo json_encode( array(

                           'code' => 0,

                           'data' => $data

                    ));

                    break;



            case 'rsa-enc':

                    $rsa = new Crypt_RSA();

                    $rsa->loadKey(RSA_PUBLIC_KEY);

                    $rsa->setEncryptionMode(CRYPT_RSA_ENCRYPTION_PKCS1);

                    $data = $rsa->encrypt($post->texto);

                    echo json_encode( array(

                           'code' => 0,

                           'data' => base64_encode($data),

                           'tipo' => 'rsa',

                           'llave' => RSA_PUBLIC_KEY,

                    ));

                    break;
```

```
            case 'rsa-dec':
                  $rsa = new Crypt_RSA();
                  $rsa->loadKey(RSA_PRIVATE_KEY);
                  $rsa->setEncryptionMode(CRYPT_RSA_ENCRYPTION_PKCS1);
                  $texto     =     ($post->tipoInput     ==     'binario')     ?
file_get_contents($_FILES['binario']['tmp_name']) : base64_decode($post->texto);
                  $data = $rsa->decrypt($texto);
                  echo json_encode( array(
                        'code' => 0,
                        'data' => $data
                  ));
                  break;

            case 'openpgp-enc':
                  $key = OpenPGP_Message::parse(base64_decode(OPENPGP_KEY));
                  $dataToEncrypt = new OpenPGP_LiteralDataPacket($post->texto);
                  $encrypted   =   OpenPGP_Crypt_Symmetric::encrypt($key,   new
OpenPGP_Message(array($dataToEncrypt)));
                  $data = $encrypted->to_bytes();
                  echo json_encode( array(
                        'code' => 0,
                        'data' => base64_encode($data),
                        'tipo' => 'openpgp',
```

```
                        'llave' => OPENPGP_KEY,

                ));

                break;



        case 'openpgp-dec':

                $key = OpenPGP_Message::parse(base64_decode(OPENPGP_KEY));

                $texto     =     ($post->tipoInput     ==     'binario')     ?
file_get_contents($_FILES['binario']['tmp_name']) : base64_decode($post->texto);

                $decryptor = new OpenPGP_Crypt_RSA($key);

                $decrypted = $decryptor->decrypt( $texto );

                $data = $decrypted->packets[0]->data;

                echo json_encode( array(

                        'code' => 0,

                        'data' => $data

                ));

                break;

    }
```

Como podemos ver, cada acción que el usuario solicite, tiene una entrada en la capa de servicios, la cual llama a las clases y librerías correspondientes y devuelve el resultado en JSON.

D. Código Algoritmo Simétrico

Para los algoritmos simétricos de la aplicación, se utilizó la extensión Mcrypt, la cual viene nativa en PHP desde su versión 5. Mcrypt admite una gran variedad de

algoritmos, las cuáles todas las usadas en esta aplicación, se ejecutan desde esta extensión.

Como todos los algoritmos utilizan la misma extensión, se unificó todo en una misma clase: CifradoSimetrico. Ésta clase provee los métodos para cifrar y descifrar los distintos textos.

```
class CifradoSimetrico {

            private $cipher = MCRYPT_DES;

            private $mode = MCRYPT_MODE_ECB;

            private $iv = NULL;



            public function __construct($cipher) {

                  $this->setCipher($cipher);

            }



            private function setCipher($cipher) {

                  switch ($cipher) {

                        case 'des':

                              $this->cipher = MCRYPT_DES;

                              break;

                        case '3des':

                              $this->cipher = MCRYPT_3DES;

                              break;

                        case 'blowfish':
```

```
                                $this->cipher = MCRYPT_BLOWFISH;
                                break;
                        case 'aes128':
                                $this->cipher = MCRYPT_RIJNDAEL_128;
                                break;
                        case 'aes192':
                                $this->cipher = MCRYPT_RIJNDAEL_192;
                                break;
                        case 'aes256':
                                $this->cipher = MCRYPT_RIJNDAEL_256;
                                break;
                }
        }

        private function getIV() {
                if ($this->iv == NULL) {
                        $this->iv = mcrypt_create_iv(mcrypt_get_iv_size($this-
>cipher, $this->mode), MCRYPT_RAND);
                }
                return $this->iv;
        }

        private function pkc5pad($texto) {
```

```
                $blocksize = mcrypt_get_block_size($this->cipher, $this->mode);

                $pad = $blocksize - (strlen($texto) % $blocksize);

                return $texto . str_repeat(chr($pad), $pad);

        }



        public function cifrar($texto, $llave) {

                $texto = $this->pkc5pad($texto);

                return mcrypt_encrypt($this->cipher, $llave, $texto, $this->mode, $this->getIV());

        }



        public function descifrar($texto, $llave) {

                $decrypted = mcrypt_decrypt($this->cipher, $llave, $texto, $this->mode, $this->getIV());

                $dec_s = strlen($decrypted);

                $padding = ord($decrypted[$dec_s-1]);

                $decrypted = substr($decrypted, 0, -$padding);

                return $decrypted;

        }

    }
```

Como podemos observar, la clase CifradoSimétrico es sencilla y abarca todos los algoritmos utilizados. Para cambiar de un algoritmo a otro, simplemente se inicializa en el constructor con el identificador del algoritmo correspondiente. El constructor llama al método setCipher para inicializar el tipo de algoritmo a utilizar.

El método getIV genera un vector de inicialización, el cual es utilizado por los distintos algoritmos para cifrar y descifrar.

El método pkc5pad retorna el texto con el padding (relleno) adecuado para el algoritmo utilizado.

Los métodos cifrar y descifrar, realizan como sus nombres lo dicen, las tareas de cifrar y descifrar los textos en los algoritmos correspondientes.

E. Código Algoritmo Asimétrico

A continuación se listan los algoritmos asimétricos utilizados. Para ver ejemplos de cómo se usan, puede ir al apartado **Capa de Servicios**.

1. RSA

1.1. Librería

PHP Secure Communications Library (phpseclib).

1.2. Clase

La clase utlizada fue Crypt_RSA, de la librería phpseclib.

1.3. Métodos

Los métodos utilizados fueron:

- loadKey: utilizado para cargar la llave, tanto pública como privada.
- setEncryptionMode: define el modo de encriptación. PKCS1 en esta aplicación.
- encrypt: cifra un texto con la llave pública cargada.
- decrypt: descifra un texto con la llave privada proporcionada.

2. OpenPGP

2.1. Librería

OpenPGP for PHP (openPGP.php)

2.2. Clase

OpenPGP, perteneciente a la librería OpenPGP for PHP.

2.3. Métodos

- parse: se encarga de convertir tipos de datos internos de la clase OpenPGP a llaves públicas y privadas.
- to_bytes: convierte las cadenas cifradas a bytes.
- encrypt: cifra un texto utilizando la llave pública.
- decrypt: descifra un texto cifrado utilizando la llave pública.

3. DSA

3.1. Librería

Se utilizó la librería OpenSSL que viene incorporada en PHP desde la versión 5.

3.2. Funciones

- openssl_public_encrypt: cifra un texto utilizando la clave pública correspondiente.
- openssl_private_decrypt: descifra un texto cifrado utilizando la clave privada proporcionada.

Conclusión

Criptografía Simétrica:

- El beneficio más importante de las criptografía de clave simétrica es su velocidad lo cual hace que éste tipo de algoritmos sean los más apropiados para el cifrado de grandes cantidades de datos.
- El problema que presenta, es la necesidad de distribuir la clave que se emplea para el cifrado por lo que si alguien consigue hacerse tanto con el mensaje como con la clave utilizada, podrá descifrar el mensaje.

Criptografía Asimétrica:

- El beneficio consiste en la supresión de la necesidad del envío de la clave, siendo por lo tanto un sistema más seguro.
- El inconveniente es la lentitud de la operación. Para solventar dicho inconveniente, el procedimiento que suele seguirse para realizar el cifrado de un mensaje es utilizar un algoritmo de clave pública junto a uno de clave simétrica.

Bibliografía

- Criptografía y Sistemas Criptográficos. Recuperado el 13 de julio de 2014, de: zonatic.usatudni.es/ca/aprenentatge/desenvolupa-sobre-el-dnie/57-aspectos-tecnicos/196-criptografia-y-esquemas-de-clave-publica.html
- La criptografía. Recuperado el 13 de julio de 2014, de: www.cert.fnmt.es/curso-de-criptografia/introduccion

Anexos

Tabla comparativa entre criptografía simétrica y asimétrica

	Ventajas	Desventajas	Garantías de seguridad	Uso	Algoritmos más usados
Simétrica	• Sistema eficiente en grupos muy reducidos, ya que sólo es necesaria una única clave • No es necesario disponer de una tercera parte confiable • Infraestructura sencilla	• Es necesario compartir la clave entre emisor y receptor por medios que pueden no ser seguros • Si se compromete la clave, se compromete toda la comunicación • No permite autenticar al emisor ya que una misma clave la utilizan dos personas • Se necesita un elevado número de claves: n*(n-1)/2; siendo n el número de personas implicadas en una comunicación cifrada	• Confidencialidad • Integridad	• Cifrado de mensajes	• DES con tamaño de clave de 56 bits • Triple-Des con tamaño de clave de 128 bits a 256 bits • Blowfish con tamaño de clave de 128 bits a 256 bits • AES con tamaños de clave de 128, 192 o 256 bits
Asimétrica	• Número de claves reducido, ya que cada individuo necesitará únicamente un par de claves • Computacionalmente es complicado encontrar la clave privada a partir de la pública • No es necesario transmitir la clave privada entre emisor y receptor • Permite autenticar a quien utilice la clave privada	• Alto coste computacional en el proceso de generación de claves • La necesidad de un tercero (Autoridad de Certificación) en el proceso • Necesidad de una gran infraestructura independientemente del número de individuos. Se precisa mayor tiempo de proceso y claves más grandes	• Confidencialidad • Integridad • Autenticidad de Origen • No Repudio	• Cifrado de mensajes • Firma digital • Intercambio de claves	• RSA con tamaño de clave mayor o igual a 1024 bits • DSA con tamaño de clave de 512 bits a 1024 bits • ElGamal con tamaño de clave comprendida entre los 1024 bits y los 2048 bits