**Practica Criptografia**

**Profesor : Felipe Rodriguez Fonte**

**Alumno : Sebastian Nahuel Flores**

**Quiero dar un agradecimiento a Felipe por su vocación a enseñar, despertó gozo en mi deseo de aprender.**

**LOS CODIGOS PYTHON ESTAN NOMBRADOS INICIALMENTE POR EL NUMERO DEL EJERCICIO QUE LES CORRESPONDE Y LUEGO DEL ALGORITMO UTILIZADO**

EJERCICIO 1)  
Tenemos un sistema que usa claves de 16 bytes. Por razones de seguridad vamos a proteger la clave de tal forma que ninguna persona tenga acceso directamente a la clave. Por ello, vamos a realizar un proceso de disociación de la misma, en el cuál tendremos, una clave fija en código, la cual, sólo el desarrollador tendrá acceso, y otra parte en un fichero de propiedades que rellenará el Key Manager. La clave final se generará por código, realizando un XOR entre la que se encuentra en el properties y en el código.

La clave fija en código es B1EF2ACFE2BAEEFF, mientras que en desarrollo sabemos que la clave final (en memoria) es 91BA13BA21AABB12. ¿Qué valor ha puesto el Key Manager en properties para forzar dicha clave final?

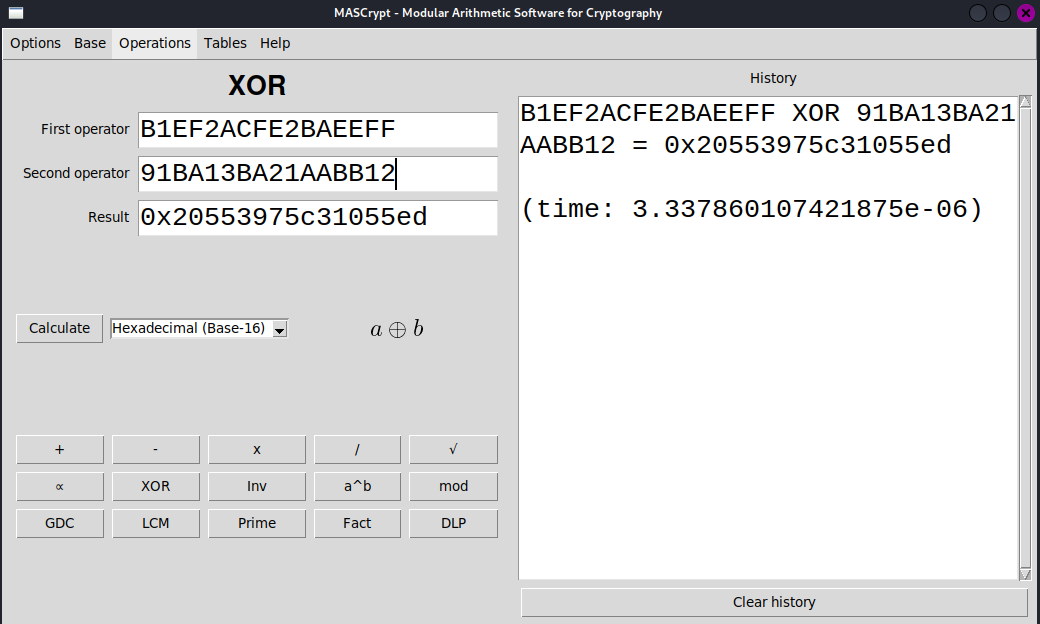
La clave fija, recordemos es B1EF2ACFE2BAEEFF, mientras que en producción sabemos que la parte dinámica que se modifica en los ficheros de propiedades es B98A15BA31AEBB3F. ¿Qué clave será con la que se trabaje en memoria?

RESULTADO  
Primero para calcular la clave que se encuentra en properties de KeyStore debemos entender que la CLAVE FINAL resulta de un XOR entre la clave KeyStore y la CLAVE FIJA, entonces si realizamos la operación XOR entre las dos claves que tenemos nos resultara la faltante.

Clave final = clave fija XOR clave properties

Entonces

Clave final XOR clave fija = clave properties

Para realizar este calculo utilice el programa MASCrypt proporcionado en clase:  


**El resultado de este calculo XOR es “20553975c31055ed”**

**Luego realizamos un XOR entre la clave fija y la dinámica y el resultado es “8653f75d31455c0”  
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**EJERCICIO 2**

Dada la clave con etiqueta “cifrado-sim-aes-256” que contiene el keystore. El iv estará compuesto por el hexadecimal correspondiente a ceros binarios (“00”). Se requiere obtener el dato en claro correspondiente al siguiente dato cifrado:

TQ9SOMKc6aFS9SlxhfK9wT18UXpPCd505Xf5J/5nLI7Of/o0QKIWXg3nu1RRz4QWElezdrLAD5LO4USt3aB/i50nvvJbBiG+le1ZhpR84oI=

Para este caso, se ha usado un AES/CBC/PKCS7. Si lo desciframos, ¿qué obtenemos?

¿Qué ocurre si decidimos cambiar el padding a x923 en el descifrado?

¿Cuánto padding se ha añadido en el cifrado?

Se valorará positivamente, obtener el dato de la clave desde el keystore mediante codificación en Python (u otro lenguaje).

**Cuando desciframos obtenemos los siguientes datos**

**La clave es: a2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72**

**El texto en claro es: Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.**

**Cuando cambiamos el estilo en padding de pkcs7 a x923 lo que sucede es que el formato como se representa el padding cambia cuando lo vemos cifrado.  
x923 4 padding = 00000004  
pkcs7 4 padding = 04040404**

**En este ejercicio especifico si quitamos el unpad, el tamaño del bloque y el estilo de padding veríamos que tiene 1 de padding = 01  
Texto

Descripción generada automáticamente**

**Pero si desciframos ya sea estilo de padding con x923 o pkcs7 veremos que el resultado es el mismo en el descifrado.**

**EJERCICIO 3:**

Se requiere cifrar el texto “KeepCoding te enseña a codificar y a cifrar”. La clave para ello, tiene la etiqueta en el Keystore “cifrado-sim-chacha-256”. El nonce “9Yccn/f5nJJhAt2S”. El algoritmo que se debe usar es un Chacha20.

¿Cómo podríamos mejorar de forma sencilla el sistema, de tal forma, que no sólo garanticemos la confidencialidad sino, además, la integridad del mismo? Se requiere obtener el dato cifrado, demuestra, tu propuesta por código, así como añadir los datos necesarios para evaluar tu propuesta de mejora.

RESPUESTA  
Para garantizar la confidencialidad e integridad del código que cree usaría el algoritmo actualizado chacha Poly1305 que contiene un autenticador que permitiría al receptor verificar si el contenido del mensaje no ha sido alterado y si proviene de la entidad esperada.

Adicional recomendaría que el nonce sea aleatorio ya que eso agregaría aleatoriedad a la clave en el cifrado y mejorar la seguridad para evitar vulnerabilidades. Esto es fundamental en el uso de un nonce

**EJERCICIO 4:**

Tenemos el siguiente jwt, cuya clave es “Con KeepCoding aprendemos”.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE

¿Qué algoritmo de firma hemos realizado?

¿Cuál es el body del jwt?

Un hacker está enviando a nuestro sistema el siguiente jwt:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI

¿Qué está intentando realizar?

¿Qué ocurre si intentamos validarlo con pyjwt?

**RESPUESTA:  
Han realizado un HMAC-SHA256 que es un HMAC con un SHA256. Esto quiere decir que se creo un HASH de 256 bits (32 bytes) y luego se combina con la clave secreta del HMAC para producir un nuevo HASH. El resultado es un hash con un autenticado a través de clave secreta.   
Su body es:  
{'usuario': 'Don Pepito de los palotes', 'rol': 'isNormal', 'iat': 1667933533}**

**El hacker envio un HMAC-SHA256 con una firma falsificada errónea intentando darse permisos de administrador. Lo podemos observar a través del mensaje del body o payload   
{**

**"usuario": "Don Pepito de los palotes",**

**"rol": "isAdmin",**

**"iat": 1667933533**

**}**

**Proporcionado por** [www.jwt.io](http://www.jwt.io)

**EJERCICIO 5:**

El siguiente hash se corresponde con un SHA3 Keccak del texto “En KeepCoding aprendemos cómo protegernos con criptografía”.

bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe

¿Qué tipo de SHA3 hemos generado?

Y si hacemos un SHA2, y obtenemos el siguiente resultado:

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?

Genera ahora un SHA3 Keccak de 256 bits con el siguiente texto: “En KeepCoding aprendemos cómo protegernos con criptografía.”

¿Qué propiedad destacarías del hash, atendiendo a los resultados anteriores?

**RESPUESTA:**

**Usted ha generado un SHA3 de 256 bits, como 1 byte son 8 bytes esto equivale a 32 bytes. Cada byte tiene espacio para 2 caracteres por lo que este SHA3-256 produce un HASH de 64 caracteres como el proporcionado en el ejercicio.**

**Generamos el HASH de** “En KeepCoding aprendemos cómo protegernos con criptografía.” con **el código proporcionado por el profesor en la carpeta código fuente keccak.py**

**Su resultado fue:  
“**302be507113222694d8c63f9813727a85fef61a152176ca90edf1cfb952b19bf”

La propiedad a la que refiere la segunda pregunta de la consigna se llama “efecto avalancha”. Esto se produce gracias a la función matemática algorítmica que actua como una esponja absorbiendo los datos de entrada y procesándolos mediante rondas, exprimiendo resultados hasta obtener el hash. Produce un cambio significativo entre hashes aunque solo tengan un “.” de diferencia.

EJERCICIO 6:

Calcula el hmac-256 (usando la clave contenida en el Keystore) del siguiente texto:

Siempre existe más de una forma de hacerlo, y más de una solución válida.

Se debe evidenciar la respuesta. Cuidado si se usan herramientas fuera de los lenguajes de programación, por las codificaciones es mejor trabajar en hexadecimal.

RESPUESTA:

“857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550”

EJERCICIO 7:  
Trabajamos en una empresa de desarrollo que tiene una aplicación web, la cual requiere un login y trabajar con passwords. Nos preguntan qué mecanismo de almacenamiento de las mismas proponemos.

Tras realizar un análisis, el analista de seguridad propone un hash SHA-1. Su responsable, le indica que es una mala opción. ¿Por qué crees que es una mala opción?

Después de meditarlo, propone almacenarlo con un SHA-256, y su responsable le pregunta si no lo va a fortalecer de alguna forma. ¿Qué se te ocurre?

Parece que el responsable se ha quedado conforme, tras mejorar la propuesta del SHA-256, no obstante, hay margen de mejora. ¿Qué propondrías?

RESPUESTA:  
SHA1 es altamente vulnerable a ataques porque es posible encontrar dos mensajes distintos que produzcan el mismo HASH al tener una longitud tan corta. Con poca capacidad de procesamiento se pueden encontrar colisiones entre hash lo que representa una amenaza a la seguridad.

SHA-256 es mejor propuesta ya que produce un hash de 256 bits (32 bytes), esto dificulta ataques de fuerza bruta y aumenta resistencia a colisiones entre HASH.

Podríamos mejorar la seguridad del SHA-256 agregando un Salt y un Pepper a la mezcla de código.  
Salt es un valor aleatorio y único que se agrega a los datos antes de aplicar el hash.   
Pepper es similar a Salt pero con diferencia que debe ser secreto en su aplicación

EJERCICIO 8:

{

"idUsuario": 1,

"movTarjeta": [{

"id": 1,

"comercio": "Comercio Juan",

"importe": 5000

}, {

"id": 2,

"comercio": "Rest Paquito",

"importe": 6000

}],

"Moneda": "EUR",

"Saldo": 23400

}

Como se puede ver en el API, tenemos ciertos parámetros que deben mantenerse confidenciales. Así mismo, nos gustaría que nadie nos modificase el mensaje sin que nos enterásemos. Se requiere una redefinición de dicha API para garantizar la integridad y la confidencialidad de los mensajes. Se debe asumir que el sistema end to end no usa TLS entre todos los puntos.

¿Qué algoritmos usarías?

RESPUESTA:

Una Api Rest envia mensajes constantes con los mismos datos y claves por lo que utilizaría un AES GCM que es el mejor algoritmo de cifrado en bloque.

Aes Gcm fue ampliamente probado, seguro y confiable porque brinda un autenticador (GCM) que asegura que los datos recibidos no han sido alterados o modificados.  
Además es rápido y eficiente que se adecua con el alto volumen de datos que se envían en una API REST.

EJERCICIO 9:

Se requiere calcular el KCV de las siguiente clave AES:

A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72

Para lo cual, vamos a requerir el KCV(SHA-256) así como el KCV(AES). El KCV(SHA-256) se corresponderá con los 3 primeros bytes del SHA-256.

Mientras que el KCV(AES) se corresponderá con cifrar un texto del tamaño del bloque AES (16 bytes) compuesto con ceros binarios (00), así como un iv igualmente compuesto de ceros binarios. Obviamente, la clave usada será la que queremos obtener su valor de control.

**RESPUESTA:**

**Cifrado AES: 5244dbd02d57d56ae08e064c56c7ca74a35eccad6db31f05841bde3d4e3ada4a**

**KCV AES: 5244db**

**Cifrado SHA256: db7df2f62c6f7d5fd13ebbc6c4fd79e95115c7c38d2a9fce8370eaac6cfd0631**

**KCV SHA256: db7df2**

**EJERCICIO 10:**

El responsable de Raúl, Pedro, ha enviado este mensaje a RRHH:   
“Se debe ascender inmediatamente a Raúl. Es necesario mejorarle sus condiciones económicas un 20% para que se quede con nosotros.”

Lo acompaña del siguiente fichero de firma PGP (MensajeRespoDeRaulARRHH.txt.sig).   
Nosotros, que pertenecemos a RRHH vamos al directorio a recuperar la clave para verificarlo. Tendremos los ficheros Pedro-priv.txt y Pedro-publ.txt, con las claves privada y pública.

Las claves de los ficheros de RRHH son RRHH-priv.txt y RRHH-publ.txt que también se tendrán disponibles.

Se requiere verificar la misma, y evidenciar dicha prueba.

Así mismo, se requiere firmar el siguiente mensaje con la clave correspondiente de las anteriores, simulando que eres personal de RRHH.  
”Viendo su perfil en el mercado, hemos decidido ascenderle y mejorarle un 25% su salario. Saludos.”

Por último, cifra el siguiente mensaje tanto con la clave pública de RRHH como la de Pedro y adjunta el fichero con la práctica.

“Estamos todos de acuerdo, el ascenso será el mes que viene, agosto, si no hay sorpresas.”

RESPUESTA:

A continuacion con prints de pantalla y descripciones explicativas expondre los pasos realizados para conseguir los archivos cifrados y firmados en la carpeta del Ejercicio 10

Descargue la aplicación Kleopatra PGP

Pegue los archivos con claves publicas y privadas junto con el .sig donde se encuentra el cifrado  
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Importo las claves publicas y privadas de pedro y RRHH  
Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Abro el .sig con Kleopatra  
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Nos reconoce que la firma es valida y la validez del certificado es absoluta  
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Abro con lector de notas el archivo generado con el mensaje  
Texto

Descripción generada automáticamente con confianza baja

Aquí el archivo MensajeRespoDeRaulARRHH  
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

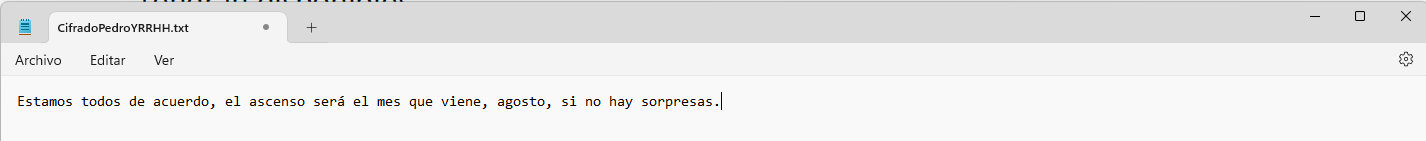
Creo un archivo txt FirmaRecursosHumanos.txt  


Y pego el texto a firmar  
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ahora voy a firmar como RRHH este txt con contraseña 123456  
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente  
Firmado  


Creo un nuevo txt para cifrar con la clave de pedro y de RRHH  


Cifro el txt para Pedro y para RRHH  
  
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Aquí la carpeta completa con el gpg final  
Interfaz de usuario gráfica

Descripción generada automáticamente

**EJERCICIO 11:**

Nuestra compañía tiene un contrato con una empresa que nos da un servicio de almacenamiento de información de videollamadas. Para lo cual, la misma nos envía la clave simétrica de cada videollamada cifrada usando un RSA-OAEP. El hash que usa el algoritmo interno es un SHA-256.

El texto cifrado es el siguiente:



Las claves pública y privada las tenemos en los ficheros clave-rsa-oaep-publ.pem y clave-rsa-oaep-priv.pem.

Si has recuperado la clave, vuelve a cifrarla con el mismo algoritmo. ¿Por qué son diferentes los textos cifrados?

**RESPUESTA:**

Desciframos el texto con nuestra clave privada (clave-rsa-oaep-priv.pem) recuperada por código y el cifrado en flujo SHA256. Nos dio el siguiente resultado:

“e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72”

Cuando volvemos a cifrar encontramos que el texto cifrado es diferente al inicial.  
Esto se produce gracias a un proceso de relleno de bits aleatorios al mensaje antes del cifrado. Sirve para proporcionar mayor seguridad a ataques y asegurarse de que no se produzcan mensajes idénticos.

EJERCICIO 12:

Nos debemos comunicar con una empresa, para lo cual, hemos decidido usar un algoritmo como el AES/GCM en la comunicación. Nuestro sistema, usa los siguientes datos en cada comunicación con el tercero:

Key:E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A426DB74

Nonce:9Yccn/f5nJJhAt2S

¿Qué estamos haciendo mal?

Cifra el siguiente texto:

He descubierto el error y no volveré a hacerlo mal

Usando para ello, la clave, y el nonce indicados. El texto cifrado presentalo en hexadecimal y en base64.

**RESPUESTAS:**

**El código esta generado correctamente pero el error se debe a que el nonce debe ser único e irrepetible. El nonce proporcionado es una variable constante, esto lo vuelve inútil ya que su utilidad es ser aleatorio. Esto lo solucionamos con la función get\_random\_bytes(12) para que la seguridad del cifrado sea eficiente.**

**EJERCICIO 13:**

Se desea calcular una firma con el algoritmo PKCS#1 v1.5 usando las claves contenidas en los ficheros clave-rsa-oaep-priv y clave-rsa-oaep-publ.pem del mensaje siguiente:

“El equipo está preparado para seguir con el proceso, necesitaremos más recursos.”

¿Cuál es el valor de la firma en hexadecimal?

Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519-priv y ed25519-publ.

RESPUESTA:  
En la carpeta “Ejercicio 11” encontre que las firmas en hexadecimal del algoritmo PKCS#1 v1.5 y ed25519.

Ambos están diseñados para firmas, pero con enfoques diferentes.   
RSA es robusto a ataques por su extensa longitud  
ED25519 es veloz y liviano por su peso en bytes

EJERCICIO 14:

Necesitamos generar una nueva clave AES, usando para ello una HKDF (HMAC-based Extract-and-Expand key derivation function) con un hash SHA-512. La clave maestra requerida se encuentra en el keystore con la etiqueta “cifrado-sim-aes-256”.   
La clave obtenida dependerá de un identificador de dispositivo, en este caso tendrá el valor en hexadecimal:

e43bb4067cbcfab3bec54437b84bef4623e345682d89de9948fbb0afedc461a3

¿Qué clave se ha obtenido?

RESPUESTA:  
La utilidad de este algoritmo radica en almacenar variadas claves de un mismo usuario en una misma clave secreta (master key).   
En el ejercicio se nos indicio que solo obtengamos la primera, su valor es:

“e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a”

Pero debemos saber que la finalidad del algoritmo es diversificar claves a partir de una sola para no almacenarlas en una base de datos.

EJERCICIO 15:

Nos envían un bloque TR31:

D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDBE6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E03CD857FD37018E111B

Donde la clave de transporte para desenvolver (unwrap) el bloque es:

A1A10101010101010101010101010102

¿Con qué algoritmo se ha protegido el bloque de clave?

¿Para qué algoritmo se ha definido la clave?

¿Para qué modo de uso se ha generado?

¿Es exportable?

¿Para qué se puede usar la clave?

¿Qué valor tiene la clave?

RESPUESTAS:

AES es el algoritmo con el que se protege y para el cual ha sido definida su clave, su uso es para cifrar y descifrar. Es exportable solo bajo una clave no confiable. La clave se puede usar para cifrar y descifrar datos de forma simétrica y tiene el valor c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1