Informe de Práctica de Ciberseguridad

Ejercicios de evaluación “Criptografía”

Nombre del Auditor: Kacper Mariusz Koper Mielczarek

Fecha: 20/12/2024

Índice

0. Introducción

En este informe de práctica sobre la criptografía nos encontraremos diversos ejercicios detallados en el PDF de la propia práctica, sin embargo, hemos detallado los enunciados en este mismo documento.

La teoría junto a los resultados que se piden en cada enunciado está detallada en este mismo documento, aunque la demostración estará expuesta en documentos aparte que están adjuntados en este directorio.

Por lo tanto, el código que demuestre los resultados obtenidos está en archivos independientes que están enumerados con el numero correspondiente del ejercicio y su apartado.

1. Ejercicio Primero

Parte 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?

Solución:

- Para conseguir sacar la clave fija 2 que ha puesto el Key Manager hemos de utilizar un script que calcula a partir de una clave_fija_1 y la clave final la clave_fija_2. Después hemos utilizado otro script para comprobar el resultado, en el que hemos

metido la clave_fija_1 y la clave_fija_2 que nos devolvió el script, y en conclusión el resultado fue certero.

Por lo tanto, el resultado de la clave_fuja_2 será "**20553975C31055ED"**

Parte 2:

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?

Solución:

- Para conseguir la clave final tuvimos que volver a utilizar un script para que nos devuelva el resultado esperado.

  Por lo tanto, la clave final con la que se trabajará en memoria será **08653F75D31455C0**

2. Ejercicio Segundo

Parte 1:

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?

Solución:

- Texto descifrado: “Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.”

  Lo que se realizado fue crear un script paso a paso para entender su funcionamiento con detalle, toda información o código que se utilizo fue sacado de los apuntes o de los scripts que fueron dados en el módulo.

Parte 2:

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

Solución:

- Si cambiamos el padding de pkcs7 a x923 el único cambio que sucede es que la secuencia del padding debe ser por ejemplo en pkcs7 “030303” en x923 sería “000003”. Por lo que podemos observar que la secuencia sigue siendo la misma con el pequeño cambio que en el x923 muestra los bytes añadidos solamente en el último byte. Solamente existe este pequeño cambio, por lo demás los dos muestran el número de bytes que se añade en el padding solamente de dos formas distintas.

Parte 3:

¿Cuánto padding se ha añadido en el cifrado?

Solución:

- Se ha añadido un byte de padding en el cifrado, que se expresa con un 01 al final del código cifrado tanto en pkcs7 como x923, ya que como solo se añade un byte los dos muestran la misma secuencia; sin embargo, si se llega a añadir más bytes tanto pkcs7 como x923 imprimirían el mismo resultado solamente con una breve diferencia explicada en el apartado anterior.

3. Ejercicio Tercero

Parte 1:

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-chacha20-256”. El nonce “9Yccn/f5nJJhAt2S”. El algoritmo que se debe usar es un Chacha20.

Solución 1:

- La solución es:

  “TslZIcqLdX4jNmBcfbq49NQLW00iDmaql490DT5ZsM1w4yFyQpkcwUC7Hho=”

Parte 2:

¿Cómo podríamos mejorar de forma sencilla el sistema, de tal forma, que no sólo garanticemos la confidencialidad si no, 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.

Solución 2:

- Podemos mejorar el cifrado de varias maneras; primero cambiamos el modelo de cifrado del “ChaCha20” al “ChaCha20-Poly1305” que proporciona tanto confidencialidad como integridad. Esto lo consigue gracias a la implementación del “tag” cuya función es verificar si el mensaje no ha sido alterado por el canal. Por lo tanto, lo que realiza es la comprobación de que el mensaje que envía el emisor sea el mismo que le llega al receptor; por lo tanto, cuando el receptor verifica el mensaje, el tag que le ha enviado el emisor tiene que coincidir con el suyo.

  Por otro lado, otra mejora que se realizo fue añadir “datos a asociados” que ayudan a mejorar el proceso de la integridad.

  Por último, se ha modificado el valor del nonce, que ahora mismo se crea de manera aleatoria para que cada vez que se intente enviar otro secreto este sea cifrado con unos patrones diferentes que el anterior mensaje.

  En conclusión, si decidimos encriptar un mensaje dos veces y dejamos el mismo nonce entonces el código encriptado será el mismo, en cambio sí modificamos el nonce el mensaje encriptado variará. Y en el último proceso, cuando el mensaje llega al receptor y se ha detectado algún mínimo cambio, el tag también cambia; por lo tanto, ya sabemos que o el nonce, o los datos asociados o el mismo mensaje ha sido manipulado por un tercero.

4. Ejercicio Cuarto

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

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSB

sb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0

dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE

Parte 1:

¿Qué algoritmo de firma hemos realizado? Y ¿Cuál es el body del jwt?

Solución 1:

- Hemos realizado un algoritmo de firma “HMAC-SHA256”, y lo sabemos porque lo hemos comprobado con un script de Python. Lo que ha realizado este script ha sido descodificar la primera parte del token (Header) que es el encargado de guardar la información de que tipo de algoritmo de firma se está usando.

  Y el body (Payload) del jwt es la segunda parte del token que está separado entre puntos:
  “eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ”

Parte 2:

Un hacker está enviando a nuestro sistema el siguiente jwt:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBk

ZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9

.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI

¿Qué está intentando realizar?

¿Qué ocurre si intentamos validarlo con pyjwt?

Solución 2:

- El hacker está intentando obtener en este caso derechos de administrador, al cambiar el BODY en el token ha intentado ponerse como administrador. Esto es posible porque el header y payload son accesibles sin necesidad de validar la “key”. Esto se debe a que está cifrado en base64. Pero el header y payload no están cifrados ni codificados hasta que no se verifique la firma.

En este caso si intentamos validar el token con pyjwt nos sale un aviso de que la clave no es correcta, y esto se debe a que el hacker ha modificado y manipulado el token. De esta manera el receptor se asegura de la integridad del mensaje.

5. Ejercicio Quinto

Parte 1:

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?

Solución 1:

- El hash es de 64 caracteres, donde cada carácter representa 4 bits; ergo 64 caracteres multiplicado por 4 bits da 256 bits. Entonces el hash que hemos generado es SHA3-256.
  Además, también lo hemos comprobado con un script en Python.

Parte 2:

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

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f

6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?

Solución 2:

- Siguiendo la misma lógica del apartado anterior como este hash está formado por 128 caracteres y cada uno de ellos representa 4 bits, entonces estamos antes un SHA512.

Parte 3:

Genera ahora un SHA3 de 256 bits con el siguiente texto: "En KeepCoding

aprendemos cómo protegernos con criptografía." ¿Qué propiedad destacarías del

hash, atendiendo a los resultados anteriores?

Solución 3:

- Con añadir al texto cualquier tipo minúsculo de variación, en este caso un mero punto hace que el hash cambie completamente, por lo tanto, a un tercero le sería imposible relacionar un hash con otro, aunque el secreto sea prácticamente idéntico.

6. Ejercicio Sexto

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.

Solución:

- El HMAC-SHA256 es:
  "857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550"

  Lo que hemos utilizado para crear el hmac-256 ha sido un script en Python, el cual recoge la clave y el texto a encriptar y los convierte en bytes, ya que hmac trabaja solamente en bytes. Después encripta el texto y verifica si el hmac se ha creado correctamente.

7. Ejercicio Séptimo

Parte 1:

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?

Solución 1:

- Propondríamos que encriptasen las contraseñas utilizando argon2, ya que almacenarlas en texto plano sería muy peligroso e inadecuado. Además de añadir a la encriptación "salt" y "pepper" para implementar su seguridad. Les aconsejaríamos que actualizasen periódicamente las contraseñas y que añadiesen protección adicional a la web utilizando HTTPS en el momento que el usuario intente acceder a la plataforma. Y, por último, les venderíamos la idea de los beneficios que tiene utilizar contraseñas de autentificación de doble factor.

  En este caso, la idea de utilizar SHA-1 es muy mala opción porque ya se han encontrado ataques de colisiones en 2017. Lo que significa que un "hacker" encontró dos entradas diferentes que producen el mismo hash.

Parte 2:

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?

Solución 2:

- Si, hubiera que añadir "salt" y "pepper" para mejorar tanto su confidencialidad como su integridad.

  Si tuviese que mejorarlo sería cambiar el algoritmo SHA-256 por el de Argon2 que es una solución más segura y aceptada por la comunidad.

8. Ejercicio Octavo

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?

Solución:

Primero usaría **AES-256-GCM** para cifrar los datos confidenciales, asegurando privacidad y autenticidad. Después, aplicaría **HMAC-SHA256** para firmar el mensaje y detectar modificaciones. Opcionalmente, implementaría **tokenización** del número de tarjeta para mayor seguridad. Además, incluiría un campo **nonce** o **timestamp** para evitar ataques de repetición. Por último, añadiría firmas digitales **RSA** para mayor seguridad.

Si llegásemos a utilizar **TLS 1.2 o 1.3** (recomendado) todo sería más seguro, ya que TLS garantiza la confidencialidad e integridad en la comunicación.

9. Ejercicio Noveno

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.

Solución:

- Para calcular el KCV tanto en SHA-256 como en AES, utilizamos un script en Python.

  Primero, calculamos el hash SHA-256 de la clave AES y extraemos los 3 primeros bytes del resultado, obteniendo así el KCV (SHA-256) que es: "DB7DF2".

  Luego, para calcular el KCV en AES, ciframos un bloque de 16 bytes usando solamente el cero, usando la clave AES en modo GCM (Porque es más segura), con un IV igualmente compuesto de ceros, y tomamos los 3 primeros bytes del texto cifrado que es: "9F5446"

  Finalmente, ambos KCV permiten verificar la integridad de la clave de forma segura.

10. Ejercicio Décimo

Parte 1:

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.

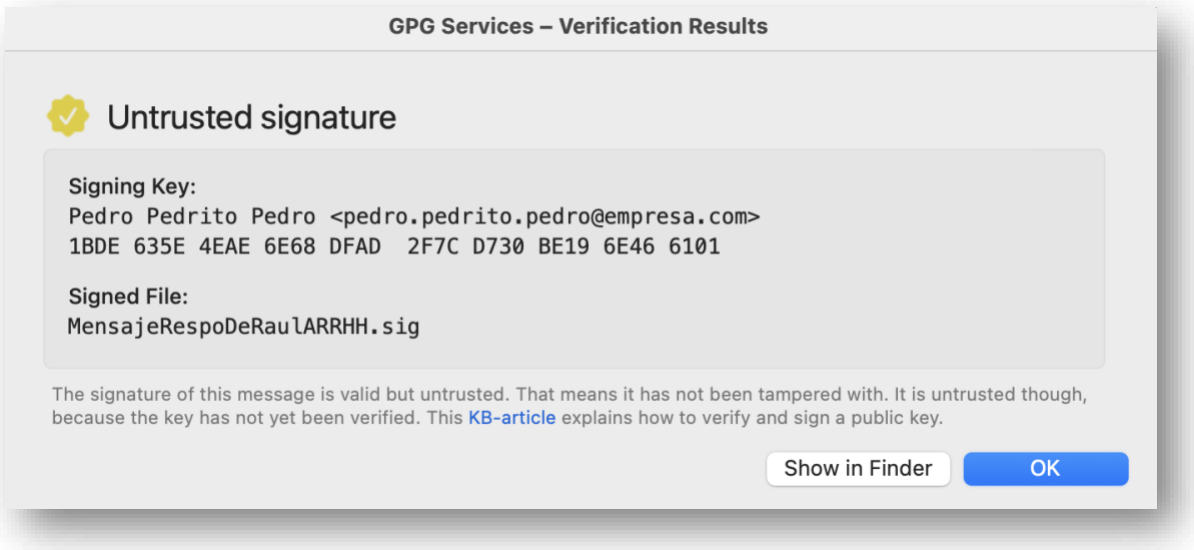
Se requiere verificar la misma, y evidenciar dicha prueba.

Solución 1:

- La clave es la correcta. Aunque cuando fue verificada obtuve una advertencia en el mensaje de verificación de la firma de que no hay pruebas de que la clave pertenezca realmente a "Pedro Pedrito Pedro". Esto se debe a que no está firmada por una entidad confiable.

  Se comprobó la veracidad de la clave por dos métodos; uno mediante la terminal utilizando comandos y la otra mediante software (GPT KEYCHAIN).

```
kk ~ $ gpg --verify MensajeRespoDeRaulARRHH.sig
gpg: Signature made Sun Jun 26 13:47:01 2022 CEST
gpg:                using EDDSA key 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101
gpg:                issuer "pedro.pedrito.pedro@empresa.com"
gpg: Good signature from "Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 1BDE 635E 4EAE 6E68 DFAD  2F7C D730 BE19 6E46 6101
```



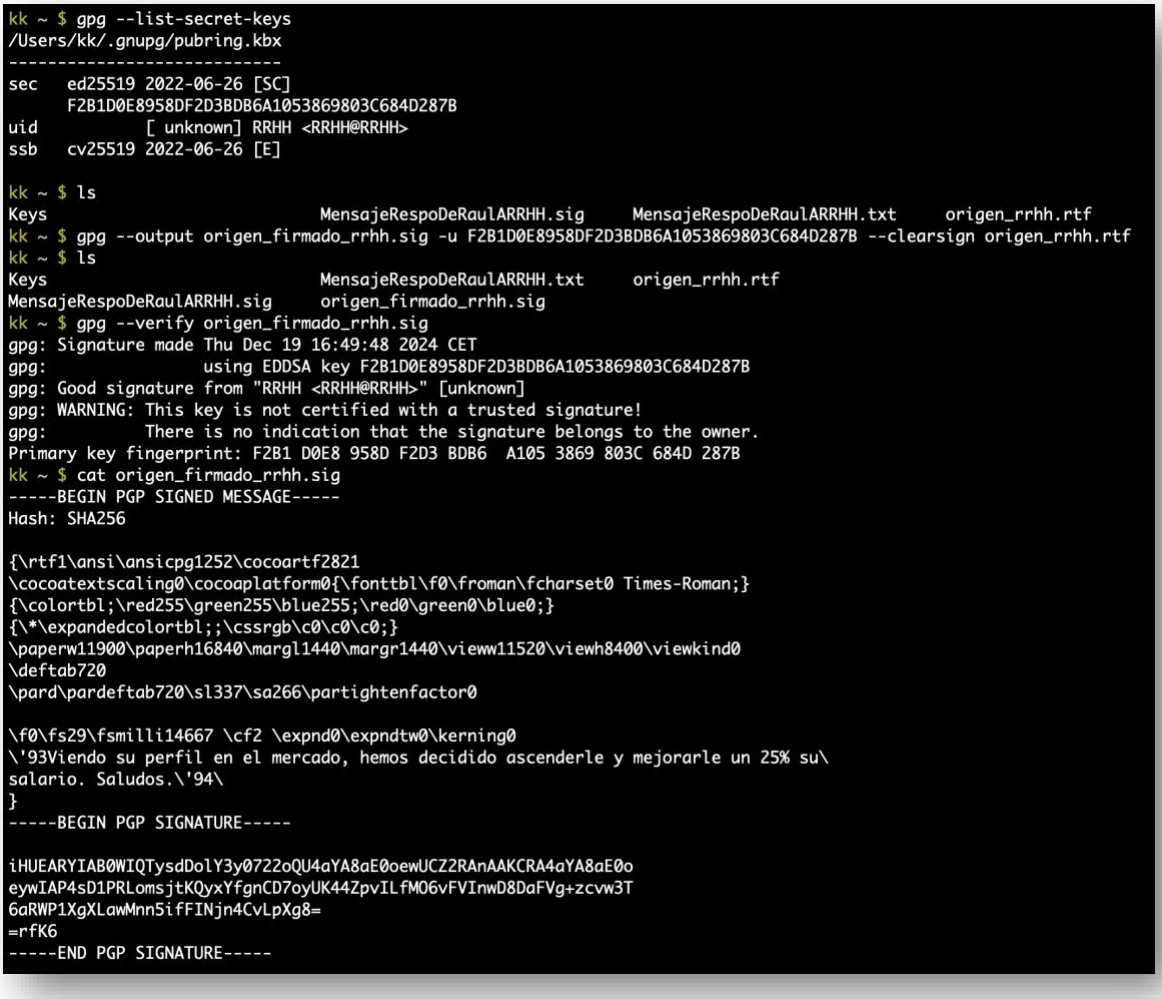
Parte 2:

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."

Solución 2:

- Para lograrlo tuvimos primero que crear un fichero y agregarle el texto correspondiente. Después importamos la clave privada de RRHH para firmar el fichero. Y, por último, comprobamos si la firma se hizo correctamente.



Parte 3:

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."

Solución 3:

- Para lograrlo teníamos que importar las claves públicas de Pedro y de RRHH. Para ello hemos utilizado el siguiente comando: "gpg --output cifrado_origen.gpg --armor --encrypt --recipient F2B1D0E8958DF2D3BDB6A1053869803C684D287B --recipient 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101 Origen_conjunto.rtf"

11. Ejercicio Undécimo

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:

“b72e6fd48155f565dd2684df3ffa8746d649b11f0ed4637fc4c99d18283b32e1709b30c

96b4a8a20d5dbc639e9d83a53681e6d96f76a0e4c279f0dffa76a329d04e3d3d4ad629

793eb00cc76d10fc00475eb76bfbc1273303882609957c4c0ae2c4f5ba670a4126f2f14

a9f4b6f41aa2edba01b4bd586624659fca82f5b4970186502de8624071be78ccef573d

896b8eac86f5d43ca7b10b59be4acf8f8e0498a455da04f67d3f98b4cd907f27639f4b1

df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f
177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b0372

2b21a526a6e447cb8ee"

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?

Solución:

- Esto se debe a la particular función de RSA-OAEP de generar un padding aleatorio cada vez que se intenta encriptar de nuevo la clave simétrica. Gracias a ello aumenta el nivel de seguridad del cifrado.

  Por lo tanto, cada vez que intentemos cifrar la misma clave simétrica nos aparecerá un texto cifrado diferente gracias a la aleatoriedad del padding. Esto también evita que dos mensajes idénticos creados con la misma clave pública tengan el mismo texto cifrado.

12. Ejercicio Duodécima

Parte 1:

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?

Solución 1:

- Estamos definiendo un once, el cual debería ser siempre un dato aleatorio y no fijo; por lo tanto, en cada cifrado debería de haber un nonce diferente.

Parte 2:

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 preséntalo en hexadecimal y en base64.

Solución 2:

- Hemos utilizado un script en Python.

  La solución es:

  Texto cifrado en Hexadecimal:
  "5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6eb8307095f5078fbfc16256d"

  Texto cifrado en Base64:
  "Xcu2Jh0PuinOOUMemgE7NMvKKk4Euy2QFJ1h9K/QTWXiq92dhLum64MHCV9QePv8FiVt"

13. Ejercicio Treceavo

Parte 1:

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?

Solución 1:

- Para conseguirlo hemos utilizado un script en Python. Por lo tanto, el valor de la firma en hexadecimal al usar la clave privada es:

  "a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23
  885c6dece92aa3d6eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f94
  1ea998ef08b2cb3a925c959bcaae2ca9e6e60f95b989c709b9a0b90a0c69d9eaccd86
  3bc924e70450ebbbb87369d721a9ec798fe66308e045417d0a56b86d84b305c555a
  0e766190d1ad0934a1befbbe031853277569f8383846d971d0daf05d023545d274f
  1bdd4b00e8954ba39dacc4a0875208f36d3c9207af096ea0f0d3baa752b48545a5d7
  9cce0c2ebb6ff601d92978a33c1a8a707c1ae1470a09663acb6b9519391b61891bf5
  e06699aa0a0dbae21f0aaaa6f9b9d59f41928d"

  Después hemos utilizado otro script en Python para comprobar si la firma era válida al descifrarla con la clave pública.

Parte 2:

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

Solución 2:

- Firma Generada (Hexadecimal) Sin Hash:

  "bf32592dc235a26e31e231063a1984bb75ffd9dc5550cf30105911ca4560dab52ab
  b40e4f7e2d3af828abac1467d95d668a80395e0a71c51798bd54469b7360d"

  Firma Generada (Hexadecimal) Con Hash:

  "cfa44c31375c4725fcea8e4bd45599eeeecdc41dea219566d412e0f5b9cc65694458
  d3c0bd9cf25e07de9f7b0137f89ab15592ec5310572205f7ec678c09e700"

14. Ejercicio Catorceavo

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" que es:

“A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72”. La clave obtenida dependerá de un identificador de dispositivo, en este caso tendrá el valor en hexadecimal:

å“e43bb4067cbcfab3bec54437b84bef4623e345682d89de9948fbb0afedc461a3”

¿Qué clave se ha obtenido?

Solución:

- Tras haber utilizado un script en Python hemos obtenido la siguiente clave:

  “e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a”

15. Ejercicio Quinceavo

Nos envían un bloque TR31:

“D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDB
E6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E0
3CD857FD37018E111B”

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

“A1A10101010101010101010101010102”

Parte 1:

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

Solución 1:

- Se ha protegido con el algoritmo basado en KBPK (Key Block Protection Key).

Parte 2:

¿Para qué algoritmo se ha definido la clave?

Solución 2:

- Se ha definido para el algoritmo A (AES)

Parte 3:

¿Para qué modo de uso se ha generado?

Solución 3:

- Se ha generado para el modo de uso B (CBC - Cipher Block Chaining)

Parte 4:

¿Es exportable?

Solución 4:

- Si que es exportable.

Parte 5:

¿Para qué se puede usar la clave?

Solución 5:

- El uso de la clave es D0 (Cifrado de datos)

Parte 6:

¿Qué valor tiene la clave?

Solución 6:

- El valor de la clave en hexadecimal es "c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1".