Práctica: Criptografía

Objetivo:

Contestar un conjunto de preguntas / ejercicios relacionados con la materia aprendida en el curso.

Detalles:

En esta práctica el alumno aplicará las técnicas y utilizará las diferentes herramientas vistas durante el módulo. Para aquellos ejercicios para los que se haya usado código, se entregará el código junto con el documento de respuestas.

Evaluación

Es obligatorio la entrega de un informe para considerar como APTA la práctica. Este informe ha de contener:

- Los enunciados seguido de las respuestas justificadas.
- En el caso de que se hayan usado comandos / herramientas también se deben nombrar y explicar los pasos realizados.

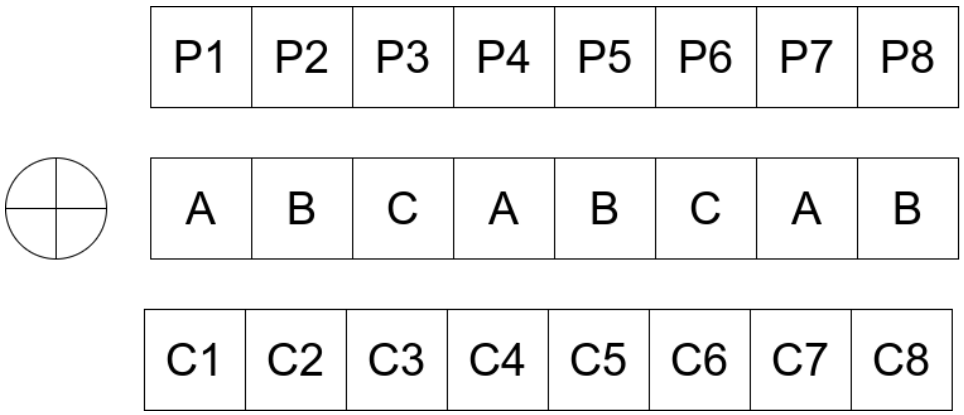
El código escrito para la resolución de los problemas se entrega en archivos separados junto al informe.

Se va a valorar el proceso de razonamiento aunque no se llegue a resolver completamente los problemas. Si el código no funciona, pero se explica detalladamente la intención se valorará positivamente.

Ejercicios:

1. **Junto a este documento se adjunta un archivo ej1.txt que contiene ciphertext en base64. Sabemos que el método para encriptarlo ha sido el siguiente:**

   La clave es de 10 caracteres. Frodo ha pensado que era poco práctico implementar one time pad, ya que requiere claves muy largas (tan largas como el mensaje). Por lo que ha decidido usar una clave corta, y repetirla indefinidamente. Es decir el primer byte de cipher es el primer byte del plaintext XOR el primer byte de la clave. Como la clave es de 10 caracteres, sabemos que el carácter 11 del ciphertext también se le ha hecho XOR con el mismo primer carácter de la clave. Debajo vemos un ejemplo donde la clave usada es "ABC".



   ¿Por qué no ha sido una buena idea la que ha tenido Frodo? Explica cómo se puede encontrar la clave y desencriptar el texto a partir del archivo.

   Programa el código y encuentra la clave junto con el texto.

   Pista: no debería ser muy costoso, ya que se puede reutilizar mucho código de un ejercicio hecho en clase ;)

No ha sido una buena idea, porque en OTP, la clave debe ser aleatoria, y debe ser del mismo
tamaño que el mensaje. Al reutilizar la misma clave, aunque sea en un mismo mensaje, podemos probar diferentes claves y analizar la frecuencia de las letras de los resultados, tal como se puede hacer cuando se reutiliza una misma clave con diferentes mensajes. Si partimos el mensaje en grupos de 10 bytes, básicamente tenemos una lista de mensajes que han sido encriptados con la misma clave de 10 bytes. La solución en código del problema consiste en identificar esta pequeña diferencia, y convertir este problema en el problema ya resuelto de claves reutilizadas en OTP. Si no supiéramos el tamaño de la clave, existen también técnicas para descubrirlo, sin tener que probar muchos tamaños.

2. **En el archivo ej2.py encontramos dos funciones. Una de ellas obtiene una string cualquiera y genera una cookie encriptada con AES CTR, la función genera cuentas con rol user, y se encarga de verificar que la string no contiene ni ‘;’ ni ‘=’. La otra función desencripta la cookie y busca la string ‘;admin=true;’. Realizando llamadas a estas dos funciones, genera una cookie y modifica el ciphertext tal que al llamar la función de check te devuelva permisos de admin. (Obviamente se puede acceder a la clave de encriptación de la clase, pero el ejercicio consiste en modificar el sistema tal y como si el código se ejecutara en un servidor remoto al que no tenemos acceso, y solo tuvieramos acceso al código).**

   Pista: Piensa en qué efecto tiene sobre el plaintext en CTR cuando cambia un bit del ciphertext

Realizar bit-flipping en CTR es muy fácil. Ya que al ser un modo de stream, la encriptación se basa en un simple XOR del plaintext con el keystream. Por lo tanto, cuando modificamos un bit del ciphertext, al desencriptarlo el mismo bit estará cambiado en el plaintext. Por lo tanto, si sabemos lo que hay en el ciphertext, y podemos modificarlo, podemos hacer cualquier cambio que queramos.

3. **Explica cómo modificarías el código del ejercicio 2 para protegerte de ataques en los que se modifica el ciphertext para conseguir una cookie admin=true.**

El propósito de los códigos de autenticación es evitar que se pueda modificar el mensaje encriptado. CTR es particularmente vulnerable a modificaciones en el ciphertext, por lo que es inseguro usarlo sin un MAC de algún tipo. Se debería añadir al mensaje enviado, un código de autenticación. Podría ser un HMAC, o podríamos cambiar el algoritmo usado, y usar GCM, que se trata de CTR con un MAC incorporado. De esta forma, si un atacante modifica el ciphertext, la MAC no será válida, y no vamos a aceptar el mensaje.

4. **Existe otro error en la implementación del Server que le puede hacer vulnerable a otro tipo de ataques, concretamente en los parámetros que usa para CTR. ¿Sabrías decir cuál es? ¿Cómo lo solucionarías?**

En la creación del servidor se genera un nonce, que se usa para cada mensaje que se encripta. Cuando se reutiliza el nonce para encriptar diferentes mensajes, lo que ocurre es que estamos encriptando varios mensajes con XOR, con el mismo keystream. Básicamente se da el mismo error que usando OTP con la misma clave reutilizada, y es vulnerable al mismo ataque de análisis de frecuencias. La forma correcta de usar CTR, es generando un nuevo nonce para cada mensaje que se encripta, y mandandolo junto con el mensaje encriptado.

5. **En el archivo ej5.py encontrarás una implementación de un sistema de autenticación de un servidor usando JWT. El objetivo del ejercicio es encontrar posibles vulnerabilidades en la implementación del servidor, y explicar brevemente cómo podrían ser atacadas.**
    - **La función register crea un nuevo usuario en el sistema (en este caso guardado localmente en el objeto AuthServer en la lista users).**
    - **La función login verifica que el password es correcto para el usuario, y si es así, devuelve un JWT donde sub = user y con expiración 6h después de la creación del token.**
    - **La función verify verifica que la firma es correcta y devuelve el usuario que está autenticado por el token.**

    Para JWT usamos la librería PyJWT: https://pyjwt.readthedocs.io/en/latest/

    Para funciones criptográficas, usamos la librería PyCryptoDome usada anteriormente en clase.

- **Se genera una llave de 4 bytes. No es un tamaño seguro ya que las posibles combinaciones no son muchas, y sería vulnerable a un ataque de fuerza bruta.**
- **Se usa un hash sha1 para guardar los passwords, eso es un problema ya que sha1 es muy inseguro, y al no usarse tampoco salt, es muy rápido conseguir los passwords a partir de los hashes, usando por ejemplo has tables.**
- **La expiración de los JWT es de 6 horas, un poco demasiado larga. Si alguien capturara un jwt, podría suplantar una sesión durante demasiado tiempo.**
- **En la verificación del JWT se acepta el algoritmo 'none', y eso da lugar a un posible ataque de jwt sin firmas.**

6. **Modifica el código para solucionar las vulnerabilidades encotradas.**

- **Soluciones en ej5.py, con comentarios.**

7. **¿Qué problema hay con la implementación de AuthServer si el cliente se conecta a él usando http? ¿Cómo lo solucionarías?**

Si un cliente se conecta al Auth Server con http, entonces las llamadas de registro y login NO serán encriptadas. Y por lo tanto cualquier persona con acceso a la network podría leer los paquetes que contienen la contraseña. Si no tuviera acceso a las llamadas de signup y login, con la llamada de verify podría obtener el jwt y enviarlo para suplantar la identidad del usuario hasta que el jwt caduque. La solución es forzar la conexión https. Un servidor de autenticación no es seguro si usa http.

8. **Sabemos que un hacker ha entrado en el ordenador de Gandalf en el pasado. Gandalf entra en internet, a la página de Facebook. En la barra del navegador puede ver que tiene un certificado SSL válido. ¿Corre algún riesgo si sigue navegando?**

Aunque el navegador muestre que el certificado es válido, si sabemos que ha entrado alguien en el ordenador de Gandalf, no podemos fiarnos completamente. Sería necesario comprobar cuál CA ha firmado el certificado y que su clave pública es la correcta, ya que se podrían haber introducido CAs maliciosos en nuestro sistema.

9. **En el archivo sergi-pub.asc mi clave pública PGP:**

    - **Comprueba que la fingerprint de mi clave es: 4010179CB16FF3B47F7D09C4751DD875E2FBBF59**
    - **Genera un par de claves, y añade tu clave pública a los archivos entregados, el archivo con tu clave pública tiene que estar encriptado para que lo pueda ver sólo con mi clave privada.**
    - **Usa tu clave privada para firmar el archivo de entrega, y adjunta la firma en un documento firma.sig (Recuerda que deberás hacer la firma con la versión final que entregues, ya que si realizas algún cambio, la firma no será válida).**

    Explica los pasos que has seguido.

Para este ejercicio básicamente se tenían que seguir los comandos de PGP vistos en clase. Primero importar mi clave del archivo a vuestro keyring, generar claves, o usar las que ya teníais generadas de clase, exportar la clave pública en un archivo, y encriptarlo para mi clave pública.
Luego con todo el documento de la práctica terminado, generar una firma con vuestra clave privada. Si se genera un cambio en el documento después de firmar, mi validación no va a funcionar. Pero si los comandos son correctos, no voy a tenerlo muy en cuenta.