PRÁCTICA 03

TEORÍA DE CÓDIGOS Y CRIPTOGRAFÍA

Autores
Adrián Jiménez Benítez
Antonio Miguel Almagro Valles
David Montoya Segura
Javier Diaz Vilchez

Asignatura
Teoría de Códigos y Criptografía

Titulación
Grado en Ingeniería Informática

1. Enunciado.

Esta tarea requiere implementar algoritmos de clave pública para guardar la clave secreta de los algoritmos simétricos de la tarea 2.

Por un lado, cada grupo de trabajo entregará un archivo pdf explicativo de la implementación, así como un manual de usuario. Los detalles de la implementación deben incluir una breve explicación de los métodos implementados o utilizados (algunos métodos pueden estar incluidos en una biblioteca, citar esto si es el caso) así como su relación con otros métodos (es recomendable un diagrama de bloques de esto). Estos detalles también deben mencionar el criptosistema(s) seleccionado(s) así como las características principales: longitud de clave, modo de operación, relleno (si existe), etc.

Incluir una tabla con las funciones o procedimientos usados en la tarea 2 y 3 donde se indique de qué librería dependen.

Por otra parte, la aplicación debe probarse durante la sesión de clase que se ha diseñado como fecha límite. Un único representante de cada grupo de trabajo realizará una presentación sobre el trabajo desarrollado, así como una demostración en vivo del funcionamiento de la aplicación. Durante la presentación, el profesor podrá realizar preguntas sobre los detalles de implementación y funcionamiento a cualquier miembro del grupo.

2. Introducción.

El cifrado es esencial para proteger la información que se quiere que sea privada o no accesible, sobre todo en sistemas donde los datos viajan en redes no seguras. Existen dos tipos de cifrado: simétrico (una misma clave para cifrar y descifrar) y asimétrico (un par de claves, pública y privada). En el proyecto se usa AES(simétrico) para cifrar archivos de manera eficiente y RSA (asimétrico) para proteger las claves de cifrado AES, logrando una combinación segura.

3. Tipos de cifrado asimétrico

3.1. RSA (Rivest-Shamir-Adleman

RSA es un algoritmo de criptografía asimétrica, lo que significa que utiliza dos claves distintas: una clave pública para cifrar y una clave privada para descifrar. Es ampliamente utilizado en la seguridad de la información, en protocolos como SSL/TLS para proteger las comunicaciones en Internet.

Pasos de funcionamiento del RSA

1. **Generación de las claves**:

Se eligen dos números primos grandes, **n** y **p**.

Se calcula $n = p \times q$, que será parte de las claves y el módulo sobre el cual se realizarán las operaciones.

Se calcula el valor de Euler $\varphi(n) = (p-1) \times (q-1)$ necesario para determinar los valores de las claves.

Se elige un número **e** tal que $1 < e < \varphi(n)$ y que sea coprimo con $\varphi(n)$ (es decir, que no tengan factores comunes).

Se calcula el valor de **d**, que es el inverso modular de **e** módulo $\varphi(n)$, y que cumple con la ecuación $d \times e \equiv 1(mod \varphi(n))$.

Al final de este proceso, se obtienen:

- **Clave pública**: (e, n), que se usa para cifrar los datos y que puede ser compartida públicamente.
- **Clave privada**: (d, n), que se mantiene en secreto y se usa para descifrar los datos.

2. **Cifrado**:
   - Supongamos que se quiere cifrar un mensaje **M** (representado como un número entero menor que **n**).
   - El mensaje cifrado **C** se calcula como: $C = M^e mod\, n$
3. **Descifrado**:
   - Para descifrar el mensaje, se usa la clave privada **d** para calcular el mensaje original: $M = C^d mod\ n$

Seguridad del RSA

La seguridad de RSA se basa en la dificultad de factorizar números grandes, ya que conocer los factores primos de nnn (que son **p** y **q**) permitiría calcular **d** y romper el cifrado

3.2. ECC (Elliptic Curve Cryptography)

La criptografía de curvas elípticas (ECC) es un sistema de criptografía asimétrica que, al igual que RSA, utiliza un par de claves pública y privada, pero se basa en las propiedades matemáticas de las curvas elípticas en lugar de la factorización de números grandes. ECC proporciona un alto nivel de seguridad con claves mucho más cortas y consume menos recursos, lo cual es especialmente útil para dispositivos con limitaciones en procesamiento y energía, como teléfonos móviles y otros dispositivos IoT.

Conceptos básicos de ECC

I. **Curva elíptica**:
   - Una curva elíptica es una curva algebraica definida por una ecuación de la forma:
     $$y^2 = x^3 + ax + b$$
     donde **a** y **b** son constantes que determinan la forma de la curva, y que deben cumplir ciertas propiedades para que la curva sea adecuada para criptografía.

II. **Punto en la curva**:

- ECC utiliza puntos sobre la curva elíptica (pares de coordenadas (x, y)) como base para sus operaciones.
- Dado un punto inicial **G** (llamado punto generador), es posible generar otros puntos mediante una operación matemática conocida como **suma de puntos**. Repetir esta operación sucesivamente crea un grupo de puntos que forma la base de la criptografía en ECC.

III. **Problema del Logaritmo Discreto en Curvas Elípticas**:

- La seguridad de ECC se basa en la dificultad de resolver el problema del logaritmo discreto en el contexto de una curva elíptica. Dado un punto **G** y un punto **P** (que es el resultado de multiplicar **G** por un número secreto **k**), es muy difícil calcular **k** incluso si se conocen **G** y **P**.

Funcionamiento de ECC

I. **Generación de las claves**:

- Se elige un número aleatorio **k** (clave privada), que será el número secreto.
- Se calcula el punto $P = k \cdot GP$ (clave pública), donde **G** es el punto generador preestablecido en la curva elíptica.
- La clave pública **P** puede compartirse abiertamente, mientras que la clave privada **k** se mantiene en secreto.

II. **Cifrado y Descifrado**:

- ECC se usa frecuentemente en esquemas de intercambio de claves y firmas digitales, y a menudo se combina con otros algoritmos (como ECDSA para firmas y ECDH para intercambio de claves).
- En el intercambio de claves (ECDH), ambas partes acuerdan un punto generador **G** y generan sus propias claves privadas y públicas. Usan sus claves privadas y la clave pública del otro para calcular un valor compartido, que luego puede usarse como clave para cifrar y descifrar datos.

3.3. DSA (Digital Signature Algorithm)

Fue adoptado como estándar por el gobierno de los EE. UU. para firmas digitales. Aunque no es un algoritmo de cifrado general, garantiza la autenticidad de los datos al verificar firmas digitales.

Ventajas de DSA

- **Autenticidad**: DSA permite verificar que un mensaje fue creado y firmado por el titular de la clave privada, proporcionando autenticación.
- **Integridad**: Si la firma es válida, garantiza que el mensaje no fue modificado desde que fue firmado.

- **Estándar gubernamental**: Es un estándar en seguridad, adoptado y avalado por el gobierno de los Estados Unidos (NIST), lo que le otorga confiabilidad en el ámbito público.

Limitaciones de DSA

- **No es un sistema de cifrado**: DSA solo es adecuado para la firma de mensajes, no para cifrar o descifrar información.
- **Sensibilidad en la generación de firmas**: La seguridad de DSA depende de la selección correcta del valor aleatorio **k** en cada firma. Si **k** es reutilizado o elegido incorrectamente, puede comprometer la clave privada.

3.4. Diffie-Hellman

El **algoritmo de intercambio de claves de Diffie-Hellman (DH)** es un método criptográfico que permite a dos partes establecer una **clave secreta compartida** sobre un canal público sin necesidad de haber tenido un contacto previo. Esta clave compartida, una vez generada, puede ser utilizada para cifrar y descifrar mensajes de manera segura. Aunque DH no cifra directamente los datos, es crucial para crear un canal seguro de comunicación.

Funcionamiento de Diffie-Hellman

El algoritmo se basa en el **problema del logaritmo discreto**, que asegura que, aunque una tercera persona escuche la comunicación, no podrá calcular la clave compartida en un tiempo razonable.

Pasos de Diffie-Hellman

1. **Elección de parámetros públicos**:
    - Ambos participantes (a menudo denominados Alice y Bob) acuerdan de antemano dos números públicos:
        - Un número primo grande **p**.
        - Un número **g**, llamado **generador** o **base**, que es un número menor que **p** y que cumple ciertas propiedades matemáticas con respecto a **p**.
    - Estos valores **p** y **g** son conocidos públicamente y no necesitan ser secretos.
2. **Generación de claves privadas y públicas**:
    - **Alice** elige un número aleatorio privado **a** (clave privada) y calcula su clave pública como: $A = g^a mod\ p$
    - **Bob** elige un número aleatorio privado **b** (clave privada) y calcula su clave pública como: $B = g^b mod\ p$
    - Alice y Bob se envían mutuamente sus claves públicas **A** y **B** a través del canal público.

3. **Cálculo de la clave compartida**:
    - **Alice** usa la clave pública de **Bob B** y su propia clave privada **a** para calcular la clave compartida: Clave compartida = $B^a mod\ p$
    - **Bob** usa la clave pública de **Alice A** y su propia clave privada **b** para calcular la misma clave compartida: Clave compartida = $A^b mod\ p$
    - Gracias a las propiedades matemáticas de los exponentes, ambos resultados serán idénticos, es decir, $B^a mod\ p = A^b$ obteniendo así la misma clave secreta compartida sin que ninguna otra persona pueda calcularla.
4. **Uso de la clave compartida**:
    - Una vez establecida, esta clave compartida puede usarse como clave de cifrado en otros algoritmos de cifrado simétrico (por ejemplo, AES) para proteger la comunicación entre Alice y Bob.

3.5. ElGamal

El **algoritmo de ElGamal** es un sistema de criptografía asimétrica que se utiliza tanto para el **cifrado de mensajes** como para la **generación de firmas digitales**. Basado en el **problema del logaritmo discreto**, ElGamal proporciona seguridad, pero es menos común que otros algoritmos como RSA y ECC debido a que requiere claves y firmas de mayor tamaño para alcanzar un nivel de seguridad similar.

Funcionamiento de ElGamal

ElGamal utiliza un par de claves (pública y privada) para cifrar y descifrar, y se basa en operaciones de exponenciación en un grupo de números enteros. A continuación, se describen los pasos generales para el cifrado, descifrado y firma digital en ElGamal.

Pasos de ElGamal

1. **Generación de claves**:
    - Se elige un número primo grande **p** y un generador **g** que genera un grupo cíclico.
    - Se selecciona un número aleatorio privado **x** tal que $1 \leq x \leq p - 2$.
    - Se calcula la clave pública **y** como: $y = g^x mod\ p$
    - La **clave pública** es el conjunto (p, g, y), y la **clave privada** es **x**.
2. **Cifrado de un mensaje**:
    - Supongamos que Alice quiere enviar un mensaje **M** a Bob, quien tiene una clave pública (p, g, y).
    - Alice representa el mensaje **M** como un número entero que sea menor que **p**.
    - Elige un valor aleatorio **k** (conocido como valor efímero) que no se reutiliza en otros mensajes.

- Alice calcula dos valores para el mensaje cifrado:
  - $c_1 = g^k mod\ p$
  - $c_2 = M \cdot y^k mod\ p$
- El mensaje cifrado es el par (c_1, c_2), que Alice envía a Bob.

3. **Descifrado del mensaje**:
   - Bob recibe el mensaje cifrado (c_1, c_2).
   - Usando su clave privada **x**, calcula el valor: $s = c_1^x mod\ p$
   - Luego, recupera el mensaje **M** como: $M = c_2 \cdot s^{-1} mod\ p$
   - Aquí, s^{-1} es el inverso multiplicativo de **s** módulo **p**, y permite a Bob descifrar el mensaje.
4. **Firma digital en ElGamal**:
   - Para firmar un mensaje, el remitente elige un número aleatorio **k** tal que $\gcd(k, p-1) = 1$.
   - Luego, calcula:
     - $r = g^k mod\ p$
     - $s = k^{-1} \cdot (H(M) - x \cdot r) mod\ (p-1)$
     - La firma del mensaje es el par (r, s).
   - Para verificar la firma, el receptor utiliza la clave pública y el mensaje firmado, verificando que se cumple una relación derivada de las propiedades de los exponentes en grupos modulares.

3.6. Tabla comparativa

Algoritmo	Seguridad	Velocidad	Longitud de Clave Requerida	Aplicaciones Típicas	Ventajas	Desventajas
RSA	Alta, basada en factorización	Moderada	Larga (2048-4096 bits)	SSL/TLS, firmas digitales, VPN	Amplia adopción, soporte para cifrado y firmas digitales	Requiere claves larga para alta seguridad
ECC	Alta, basada en curvas elípticas	Alta	Corta (256-521 bits)	Dispositivos móviles, IoT, criptomonedas	Eficiencia en dispositivos con recursos limitados	Matemática complejas, menor adopción
DSA	Alta, basado en logaritmos	Alta para firmas	Similar a RSA	Firmas digitales	Eficiente en creación de firmas	Solo para firmas digitales, no para cifrado
Diffie-Hellman	Alta, intercambio de claves	Alta	Variable (2048 bits o más)	Intercambio de claves, VPN	Permite establecer claves compartidas en canales inseguros	Vulnerable sin autenticación adicional
ElGamal	ALta, basado en logaritmos	Más lento que RSA	Larga(2048-4096 bits)	Cifrado y firmas digitales	Seguridad basada en logaritmos discretos	Firmas y claves grandes, menos eficiente

3.7. Conclusión

Elegir **RSA** como algoritmo de desarrollo es ideal en aplicaciones que requieren compatibilidad y flexibilidad, pues RSA se utiliza ampliamente en protocolos de seguridad (como TLS/SSL) y soporta tanto el cifrado como las firmas digitales. Su base matemática en la factorización de números grandes le otorga una seguridad probada y bien estudiada, con una gran cantidad de bibliotecas optimizadas para facilitar su implementación. Aunque ECC es más eficiente en sistemas con recursos limitados, RSA sigue siendo preferible por su simplicidad y menor complejidad. Además, a diferencia de otros como DSA o ElGamal, RSA no depende de valores efímeros únicos, reduciendo riesgos de seguridad adicionales. En general, RSA es una opción sólida, versátil y confiable en entornos de seguridad robusta.

4. Manual de Usuario.

4.1. Cargar el archivo de claves

- Al iniciar el programa por primera vez, no tendrás un archivo de claves existente. Selecciona "Cancelar" al intentar cargar el archivo.
- El sistema detectará que no tienes un archivo de claves y te dará la opción de crear uno en blanco. Este archivo se irá actualizando automáticamente con las claves de los archivos que cifres en el futuro.

4.2. Cifrar archivos

- Ahora se puede seleccionar los archivos que desees cifrar. Cada archivo tendrá su propia clave asociada.
- El programa generará una clave única para cada archivo y la almacenará en el archivo “claves.bin”, que creaste en el paso anterior.
- En “claves.bin” se guardará:
    - El nombre del archivo cifrado.
    - La clave generada.
    - La extensión de archivo.
- Si se cifran múltiples archivos, cada uno tendrá su clave única en “claves.bin”, que se irá actualizando con el nombre y la extensión de cada nuevo archivo.

4.3. Proteger el archivo de claves

- Como las claves almacenadas en “claves.bin” son vulnerables en este momento, es importante encriptar este archivo.
- Para proteger el archivo de claves, genera una clave pública y una clave privada, que se guardarán en archivos separados con extensión **“.pem”**.
    - La **clave pública** se usará para encriptar el archivo de claves.
    - La **clave privada** te permitirá descifrar el archivo de claves cuando lo necesites.

4.4. Acceder al archivo de claves encriptado

- Cuando vuelvas a abrir el archivo “claves.bin”, el sistema te preguntará si está encriptado con una clave pública. Selecciona “Sí” para confirmar.
- Luego, se abrirá un cuadro de diálogo donde deberás proporcionar la **clave privada** para desbloquear y acceder a “claves.bin”.

4.5. Desencriptar archivos

- Una vez que hayas accedido a “claves.bin” usando la clave privada, tendrás acceso a todas las claves de los archivos cifrados.

- Esto permite desencriptar cualquier archivo sin tener que ingresar otra clave adicional, ya que el programa detecta automáticamente que tienes acceso total a “claves.bin”.

5. Descripción de los métodos específicos en el código

- **crear_RSA:**
    - Genera un par de claves RSA ((pública y privada) de 2048 bits. Las claves se almacenan en archivos **.pem** para su uso posterior.
    - Es un método necesario para iniciar el proceso de cifrado, ya que los archivos de claves generados aquí son necesarios para proteger la clave AES en otros métodos.
- **cifrar_archivo:**
    - Cifra un archivo utilizando una clave AES generada aleatoriamente. La función genera también un IV, que se almacena junto con el archivo cifrado para su posterior descifrado.
    - La clave AES generada es luego protegida mediante RSA en el método **encriptar_claves**.
- **descifrar_archivo:**
    - Descifra un archivo utilizando la clave AES y el IV correspondiente. Este método permite recuperar el contenido original del archivo cifrado.
    - Depende de que **habilitar_claves** haya descifrado correctamente la clave AES.
- **encriptar_claves:**
    - Cifra la clave AES generada utilizando la clave pública RSA, lo que permite guardar la clave de forma segura y protegerla contra accesos no autorizados.
    - Asegura la clave AES de cada archivo, que luego será descifrada mediante **habilitar_claves**.
- **habilitar_claves:**
    - Descifra la clave AES cifrada utilizando la clave privada RSA, permitiendo recuperar la clave necesaria para descifrar un archivo.
    - Es el paso previo a la función **descifrar-archivo**, proporcionando la clave AES necesaria para acceder a los datos originales.

6. Diagramas de Bloques

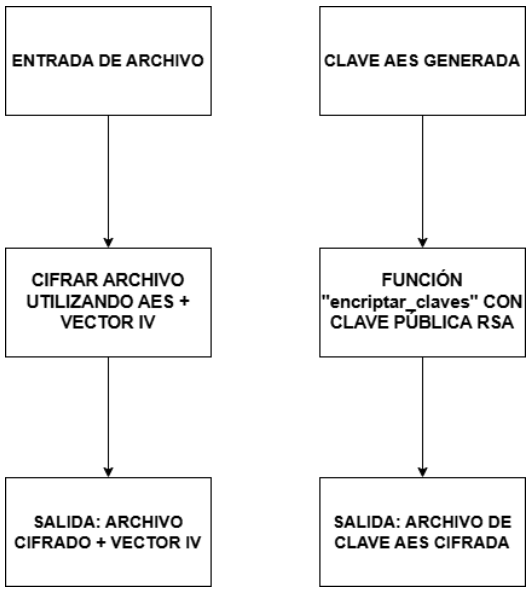
6.1. Generación de claves RSA

1. **Inicio**: Este es el punto de partida del proceso. Indica que se va a iniciar la generación del par de claves.

2. **Generar par de claves RSA con la función "crear_RSA"**: En esta etapa, se ejecuta una función llamada crear_RSA. Esta función es responsable de generar las claves RSA. El algoritmo RSA crea un par de claves (una clave pública y una clave privada) a partir de números primos grandes. La seguridad del algoritmo se basa en la dificultad de factorizar el producto de estos números primos.

3. **Salida: Archivos de clave pública y privada .PEM**: Finalmente, la salida del proceso son dos archivos en formato .PEM, que contienen las claves generadas. El archivo de clave pública puede ser compartido libremente y se utiliza para cifrar datos o verificar firmas digitales, mientras que el archivo de clave privada debe mantenerse seguro, ya que se usa para descifrar datos o firmar digitalmente.

6.2. Cifrado del Archivo



Este diagrama de bloques representa el proceso de cifrado de un archivo mediante una combinación de cifrado simétrico (AES) y asimétrico (RSA) para proteger tanto el contenido como la clave de cifrado. Aquí tienes el desglose del proceso:

1. **Entrada de archivo**: Este bloque indica el archivo que se desea cifrar. Este archivo será la entrada para el proceso de cifrado simétrico.

2. **Clave AES generada**: En paralelo, se genera una clave AES, que será utilizada para cifrar el archivo. AES es un algoritmo de cifrado simétrico, lo cual significa que la misma clave AES se usará tanto para cifrar como para descifrar el archivo.

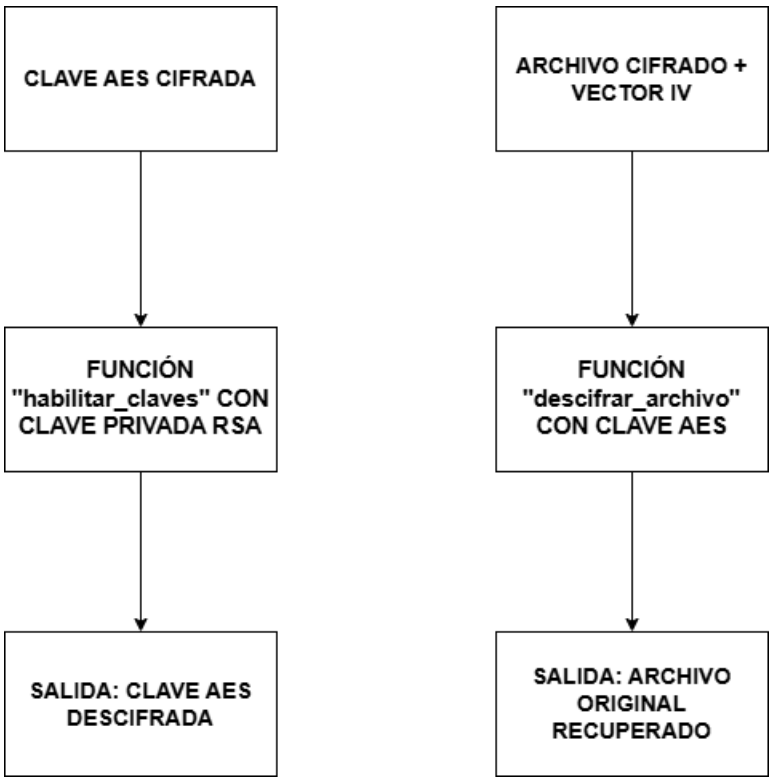
3. **Cifrar archivo utilizando AES + Vector IV**: Con la clave AES generada y un vector de inicialización (IV), el archivo es cifrado. El vector IV ayuda a garantizar que el cifrado sea más seguro, al evitar que se generen patrones predecibles en el texto cifrado.

4. **Salida: Archivo cifrado + Vector IV**: La salida de esta etapa es el archivo cifrado junto con el vector IV, necesarios para el posterior descifrado del archivo.

5. **Función "encriptar_claves" con clave pública RSA**: La clave AES generada es cifrada usando la clave pública RSA mediante la función encriptar_claves. Esto asegura que solo el propietario de la clave privada correspondiente podrá descifrar la clave AES y, por lo tanto, acceder al archivo cifrado.

6. **Salida: Archivo de clave AES cifrada**: El resultado final es un archivo que contiene la clave AES cifrada. Este archivo permite que la clave de cifrado (AES) esté protegida, de manera que solo el destinatario con la clave privada RSA adecuada pueda acceder a ella.

6.3. Descifrado del Archivo



Este diagrama de bloques representa el proceso de descifrado para recuperar un archivo que ha sido cifrado con una combinación de cifrado simétrico (AES) y asimétrico (RSA). A continuación, te explico cada parte del proceso:

1. **Clave AES cifrada**: Este bloque contiene la clave AES en su forma cifrada. Como la clave AES se cifró inicialmente con la clave pública RSA, necesita ser descifrada con la clave privada RSA para poder usarse.

2. **Función "habilitar_claves" con clave privada RSA**: En este paso, se utiliza la función habilitar_claves junto con la clave privada RSA para descifrar la clave AES. Este proceso permite recuperar la clave AES original, que luego será utilizada para descifrar el archivo.

3. **Salida: Clave AES descifrada**: La salida de este paso es la clave AES en su forma original, ya lista para ser utilizada en el descifrado del archivo.

4. **Archivo cifrado + Vector IV**: Este bloque representa el archivo cifrado junto con el vector de inicialización (IV) que se usó durante el cifrado AES. Ambos elementos son necesarios para realizar el descifrado adecuado del archivo.

5. **Función "descifrar_archivo" con clave AES**: Usando la clave AES descifrada y el vector IV, se aplica la función descifrar_archivo para revertir el proceso de cifrado, devolviendo el archivo a su estado original.

6. **Salida: Archivo original recuperado**: El resultado final de este proceso es el archivo original, que ha sido descifrado y recuperado con éxito.

7. Bibliografía.

[1] Cryptography - enlace (03-10-2024)

[2] pyAesCrypt - enlace (03-10-2024)

[3] Fernet - enlace (03-10-2024)

[4] pyCrypto - enlace (03-10-2024)

[5] pyCryptodome - enlace (03-10-2024)

[6] Jupyter lab - enlace (04-10-2024)

[7] IEEE Xplore - enlace (04-11-2024).

[8] Practical Networking - enlace (04-11-2024).