Документација за CCMP Протоколот - Објаснување на кодот

Оваа Java програма го симулира CCMP (Counter Mode with CBC-MAC Protocol), што е протокол за безбедност кој се користи во безжични комуникации, како што е WPA2 (Wi-Fi Protected Access II). CCMP се користи за шифрирање и потврдување на интегритетот на податоци во мрежи. Програмата ја користи AES (Advanced Encryption Standard) во CTR (Counter) режим за шифрирање на податоците и MIC (Message Integrity Code) за потврда на интегритетот на податоците.
Програмата користи **AES-128** алгоритам за шифрирање и генерира **nonce** (случаен број), шифрира податоци, пресметува MIC, ги дешифрира податоците и потврдува дали MIC е ист како при примањето.

Клучни Функции на Програмата:

1. **Шифрирање/Дешифрирање со AES** во **CTR режим**.
2. **Пресметување MIC** на базата на шифрираните податоци.
3. **Генерирање на nonce** за секоја сесија на шифрирање.
4. **Креирање блокови** од податоците кои се шифрираат по блокови од 128 битови.

Класа: `CCMPProtocol`

Објаснување на методите:

1. generateKey()

Цел: Генерира 128-битен AES клуч.

Објаснување:

-Методот користи класата KeyGenerator за да генерира нов AES клуч од 128 бита (16 бајти) со алгоритмот AES.

-Клучот се иницијализира со методата keyGen.init(128) и потоа се враќа.

Код:

```
public static SecretKey generateKey() throws Exception {
    KeyGenerator keyGen = KeyGenerator.getInstance("AES");
    keyGen.init(128);
    return keyGen.generateKey();
}
```

Излез: Враќа случајно генериран 128-битен AES таен клуч.

2. encrypt(SecretKey key, byte[] plaintext, byte[] nonce)

Цел: Шифрира влезни податоци (plaintext) со користење на AES во **CTR режим** со дадениот nonce.

Објаснување:

-Методот користи Cipher.getInstance("AES/CTR/NoPadding") за да создаде инстанца на шифрата во **AES-CTR** режим.

-Создава IvParameterSpec (иницијален вектор) користејќи го дадениот nonce (16 бајти). Овој nonce ќе се користи како IV за шифрирање.

-Податоците се шифрираат со помош на doFinal() методот и резултирачкиот шифриран текст се враќа.

Код:

```
public static byte[] encrypt(SecretKey key, byte[] plaintext, byte[] nonce) throws
Exception {
    Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding");
    IvParameterSpec ivSpec = new IvParameterSpec(nonce);
    cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
    return cipher.doFinal(plaintext);}
```

Влезни параметри:

-key: 128-битен AES таен клуч.

-plaintext: Податоците кои треба да се шифрираат.

-nonce: Случаен број кој ќе се користи како IV (иницијален вектор) во CTR режимот (16 бајти).

Излез: Враќа шифрирани податоци (ciphertext) како низа од бајти.

3. decrypt(SecretKey key, byte[] ciphertext, byte[] nonce)

Цел: Дешифрира шифрирани податоци (`ciphertext`) користејќи AES во **CTR режим** со дадениот `nonce`.

Објаснување:

-Овој метод е сличен на методот за шифрирање, но работи во **режим за дешифрирање** (`Cipher.DECRYPT_MODE`).

`-nonce` се користи како IV (иницијален вектор) и се дешифрираат шифрираните податоци назад во оригиналниот текст.

Код:

```
public static byte[] decrypt(SecretKey key, byte[] ciphertext, byte[] nonce) throws Exception {
    Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding");
    IvParameterSpec ivSpec = new IvParameterSpec(nonce);
    cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
    return cipher.doFinal(ciphertext);
}
```

Влезни параметри:

-key: 128-битен AES таен клуч.

-ciphertext: Шифрирани податоци кои треба да се дешифрираат.

-nonce: Случаен број кој ќе се користи како IV во CTR режимот (16 бајти).

Излез: Враќа дешифрирани податоци (plaintext) како низа од бајти.

4. calculateMIC(byte[] key, byte[] data)

Цел: Пресметува MIC (Message Integrity Code) користејќи **SHA-256**.

Објаснување:

-Методот користи **SHA-256** за да пресмета хеш на податоците (`data`).

-Прво се додава клучот како додаток за хеширање и потоа се додаваат податоците.

-Резултатниот хеш се скратува на **64 бита** (8 бајти) за MIC.

Код:

```
public static byte[] calculateMIC(byte[] key, byte[] data) throws Exception {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    digest.update(key);
    digest.update(data);
    return Arrays.copyOf(digest.digest(), 8); // MIC: 64 bits
}
```

Влезни параметри:

-key: Таен AES клуч (или друг клуч за HMAC).

-data: Податоци за кои треба да се пресмета MIC.

Излез: Враќа MIC како низа од 8 бајти (64 бита).

5. createBlocks(byte[] packet)

Цел: Креира 128-битни блокови од податоците (пакетот), со додавање на пополнување ако е потребно.

Објаснување:

-Методот го дели влезниот пакет на блокови од 128 бита (16 бајти).

-Ако последниот блок е помал од 128 бита, се пополнува со нули.

Код:

```
public static byte[][] createBlocks(byte[] packet) {
    int blockSize = 16; // 128 bits = 16 bytes
    int numberOfBlocks = (packet.length + blockSize - 1) / blockSize;
    byte[][] blocks = new byte[numberOfBlocks][blockSize];

    for (int i = 0; i < numberOfBlocks; i++) {
        int offset = i * blockSize;
        int length = Math.min(packet.length - offset, blockSize);
        System.arraycopy(packet, offset, blocks[i], 0, length);
        // Ако последниот блок е помал од 128 бита, го пополнуваме со 0
        if (length < blockSize) {
            Arrays.fill(blocks[i], length, blockSize, (byte) 0);
        }
    }
    return blocks;
}
```

Влезни параметри:

-packet: Податоците кои треба да се делат на блокови.

Излез: Враќа низата од блокови.

6. generateNonce()

Цел: Генерира случаен **nonce** од 12 бајти и го проширува до 16 бајти, што е потребно за AES-CTR режим.

Објаснување:

-Методот создава **nonce** од 12 бајти со користење на **SecureRandom** кој обезбедува криптографски сигурни случајни бајтови.

-За да се користи во **AES-CTR** режимот, nonce се проширува до 16 бајти, каде што последните 4 бајти се пополнуваат со нули (или можат да се заменат со нешто друго, ако е потребно).

Код:

```
public static byte[] generateNonce() {

    byte[] nonce = new byte[12]; // 12 bytes for the nonce

    new SecureRandom().nextBytes(nonce); // Fill with random bytes



    // Padding the nonce to 16 bytes for AES-CTR mode

    byte[] nonce16Bytes = new byte[16];

    System.arraycopy(nonce, 0, nonce16Bytes, 0, 12);  // Copy 12 bytes of the original nonce

    // The last 4 bytes will remain 0 (they can be replaced with something else if needed)

    return nonce16Bytes;
```

}

Излез: Враќа **nonce** од 16 бајти што ќе се користи како IV за шифрирање во AES-CTR режим.

87 bytesToHex(byte[] bytes)

Цел: Конвертира низа од бајтови во хексадецимален формат.

Објаснување:

-Овој метод користи **StringBuilder** за да ги претвори бајтовите во хексадецимален стринг.

-Секој бајт се конвертира во двоцифрен хексадецимален број и се додава на резултантниот стринг.

Код:

```
public static String bytesToHex(byte[] bytes) {
    StringBuilder sb = new StringBuilder();
    for (byte b : bytes) {
        sb.append(String.format("%02x", b));
    }
    return sb.toString();
}
```

Излез: Враќа хексадецимален стринг кој го претставува влезниот низ од бајтови.

8. Главен метод main()

Цел: Главната логика на програмата, која ги комбинира сите претходно објаснети методи.

Објаснување:

-Програмата започнува со **генерирање на AES клуч** и **генерирање на nonce** (случаен број).

-Врши **шифрирање на IP пакет** во **CTR режим** со користење на AES.

-Пресметува MIC за шифрираниот податок и го испечатува.

-Дешифрира податоците и ги **враќа оригиналните податоци**.

-На крај, се прави **проверка на MIC** за да се потврди дали податоците не биле изменети.

Код:

```
public static void main(String[] args) {

    try (Scanner scanner = new Scanner(System.in)) {

        // Генерирање AES клуч

        SecretKey key = generateKey();



        // Внесете IP пакет

        System.out.print("Enter the IP packet: ");

        String input = scanner.nextLine();



        // Претворање на влезниот податок во бајтови

        byte[] packet = input.getBytes(StandardCharsets.UTF_8);



        // Генерирање на nonce

        byte[] nonce = generateNonce();
```

```
System.out.println("Generated Nonce: " + bytesToHex(nonce));

// Креирање 128-битни блокови од пакетот
byte[][] blocks = createBlocks(packet);

// Шифрирање на секој блок
byte[] ciphertext = new byte[blocks.length * 16];
for (int i = 0; i < blocks.length; i++) {
    byte[] encryptedBlock = encrypt(key, blocks[i], nonce);
    System.arraycopy(encryptedBlock, 0, ciphertext, i * 16, 16);
}
System.out.println("Ciphertext: " + bytesToHex(ciphertext));

// Пресметување на MIC за шифрираниот податок
byte[] mic = calculateMIC(key.getEncoded(), ciphertext);
System.out.println("MIC: " + bytesToHex(mic));

// Симулирање на примање на рамката (ciphertext + MIC)
byte[] receivedCiphertext = ciphertext; // примени шифриран текст
byte[] receivedMic = mic; // примени MIC

// Дешифрирање на секој блок и реконструкција на оригиналниот пакет
byte[] decryptedPacket = new byte[packet.length];
```

```
        int decryptedLength = 0; // Следење на должината на декриптираните податоци

        for (int i = 0; i < blocks.length; i++) {

            byte[] decryptedBlock = decrypt(key, Arrays.copyOfRange(receivedCiphertext, i
* 16, (i + 1) * 16), nonce);

            // Пресметување на вистинската должина за копирање

            int lengthToCopy = (i == blocks.length - 1) ? packet.length % 16 : 16;

            System.arraycopy(decryptedBlock, 0, decryptedPacket, decryptedLength,
lengthToCopy);

            decryptedLength += lengthToCopy;

        }

        System.out.println("Decrypted packet: " + new String(decryptedPacket, 0,
decryptedLength, StandardCharsets.UTF_8));



        // Проверка на MIC

        boolean isMicValid = Arrays.equals(calculateMIC(key.getEncoded(),
receivedCiphertext), receivedMic);

        System.out.println("MIC valid: " + isMicValid);

    } catch (Exception e) {

        e.printStackTrace();

    }

}
```

Основни активности:

-Генерирање на AES клуч за шифрирање.

-Внесување на IP пакет и претворање во бајтови.

-Генерирање на nonce за користење во AES-CTR режим.

-Шифрирање на податоците по блокови.

-Пресметување на MIC за шифрираниот податок.

-Дешифрирање на податоците и реконструирање на оригиналниот IP пакет.

-Проверка на MIC за потврдување на интегритетот на применият податок.

Заклучок:

Програмата демонстрира основни операции на CCMP протоколот, користејќи AES во **CTR режим** за шифрирање и **SHA-256** за пресметување на MIC. Таа вклучува процеси за шифрирање и дешифрирање на податоци, заедно со механизам за потврдување на интегритетот на податоците преку MIC.

Изработил 221078, Верица Чочоровска