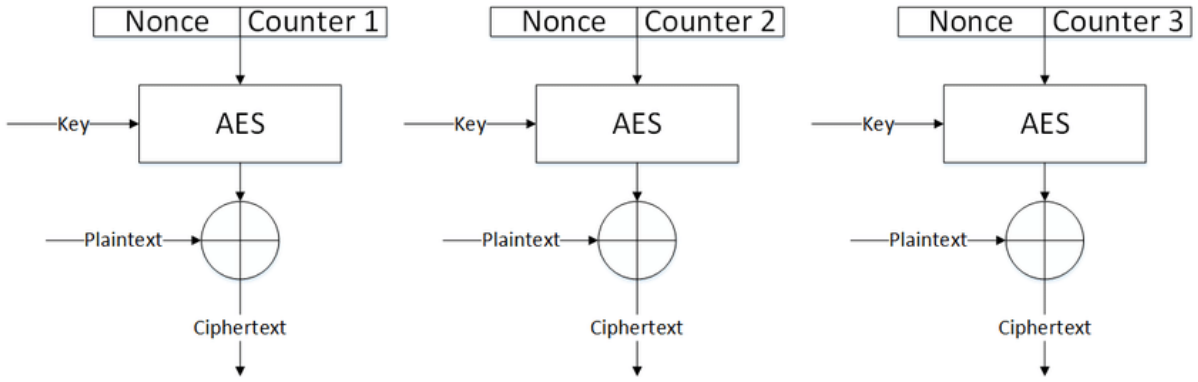
Kajetan Harla 148369

Krzysztof Surażyński 162839

AES to symetryczny algorytm szyfrowania blokowego, który szyfruje dane w blokach o długości 128 bitów (16 bajtów). Obsługuje trzy długości klucza: 128, 192 lub 256 bitów, co odpowiada odpowiednio 10, 12 lub 14 rundom szyfrowania. Każda runda składa się z czterech głównych operacji:

1. SubBytes – podstawienie każdego bajtu przez inny według ustalonej tablicy (S-Box), zapewniające nieliniowość.
2. ShiftRows – przestawienie bajtów w wierszach stanu (macierzy 4x4) w celu rozproszenia danych.
3. MixColumns – przemieszanie danych kolumnami, wykorzystując mnożenie w ciele Galois, co zapewnia dyfuzję.
4. AddRoundKey – operacja XOR pomiędzy stanem a podkluczem rundy wygenerowanym z klucza głównego (Key Expansion).



Rysunek 1 Schemat szyfrowania AES w trybie CTR

AES w trybie **CTR** przekształca szyfrowanie blokowe w szyfrowanie strumieniowe. Zamiast szyfrować bezpośrednio dane, szyfrowany jest licznik (counter) — unikalna wartość dla każdego bloku. Wynik szyfrowania licznika jest następnie XOR-owany z danymi wejściowymi (tekstem jawnym lub zaszyfrowanym). Licznik zazwyczaj składa się z wektora inicjalizującego (IV) i rosnącego numeru bloku. Dzięki temu CTR pozwala na równoległe szyfrowanie i deszyfrowanie oraz nie wymaga paddingu, co czyni go szybkim i elastycznym.

Kod programu:

```
import * as fs from "fs";
import * as crypto from "crypto";
import * as path from "path";

/**
 * File encryption utility using AES in CTR mode
 * This script reads a file, encrypts its contents using AES-256-CTR,
 * and writes the encrypted data to a new file.
 */

// Function to encrypt a file using AES in CTR mode
function encryptFile(
```

```
  inputFilePath: string,
  outputFilePath: string,
  key: Buffer,
  iv: Buffer
): Promise<void> {
  return new Promise((resolve, reject) => {
    try {
      const readStream = fs.createReadStream(inputFilePath);
      const writeStream = fs.createWriteStream(outputFilePath);

      const cipher = crypto.createCipheriv("aes-256-ctr", key, iv);

      readStream
        .pipe(cipher)
        .pipe(writeStream)
        .on("finish", () => {
          console.log(`File encrypted successfully: ${outputFilePath}`);
          resolve();
        })
        .on("error", (error) => {
          reject(error);
        });
    } catch (error) {
      reject(error);
    }
  });
}

// Function to decrypt a file using AES in CTR mode
function decryptFile(
  inputFilePath: string,
  outputFilePath: string,
  key: Buffer,
  iv: Buffer
): Promise<void> {
  return new Promise((resolve, reject) => {
    try {
      const readStream = fs.createReadStream(inputFilePath);
      const writeStream = fs.createWriteStream(outputFilePath);

      const decipher = crypto.createDecipheriv("aes-256-ctr", key, iv);

      readStream
        .pipe(decipher)
        .pipe(writeStream)
        .on("finish", () => {
          console.log(`File decrypted successfully: ${outputFilePath}`);
          resolve();
        })
        .on("error", (error) => {
          reject(error);
        });
```

```
    } catch (error) {
      reject(error);
    }
  });
}

// Generate a random encryption key and IV
function generateKeyAndIV(): { key: Buffer; iv: Buffer } {
  // AES-256 requires a 32-byte key and 16-byte IV for CTR mode
  const key = crypto.randomBytes(32); // 256 bits
  const iv = crypto.randomBytes(16); // 128 bits
  return { key, iv };
}

async function main() {
  try {
    // Get file paths from command line arguments or use defaults
    const inputFileName = process.argv[2] || "large-file.txt";
    const outputFileName = process.argv[3] || `${inputFileName}.encrypted.enc`;
    const decryptedFileName =
      process.argv[4] || `${inputFileName}.decrypted.txt`;

    const inputFilePath = path.join(__dirname, "..", "assets", inputFileName);
    const outputFilePath = path.join(__dirname, "..", "assets", outputFileName);
    const decryptedFilePath = path.join(
      __dirname,
      "..",
      "assets",
      decryptedFileName
    );

    try {
      const fileStats = fs.statSync(inputFilePath);
      console.log(
        `Input file size: ${fileStats.size / 1024 / 1024 / 1024} Gigabytes`
      );
    } catch (error) {
      console.error(`Error getting file size: ${(error as Error).message}`);
    }

    // Check if input file exists
    if (!fs.existsSync(inputFilePath)) {
      console.error(`Error: Input file '${inputFileName}' not found`);
      process.exit(1);
    }

    // Generate encryption key and IV
    const { key, iv } = generateKeyAndIV();

    // Save the key and IV to files for later decryption
    const keyFileName = `${outputFileName}.key`;
    const ivFileName = `${outputFileName}.iv`;
```

```
    const keyFilePath = path.join(__dirname, "..", "assets", keyFileName);
    const ivFilePath = path.join(__dirname, "..", "assets", ivFileName);

    fs.writeFileSync(keyFilePath, key);
    console.log(`Encryption key saved to: ${keyFilePath}`);

    fs.writeFileSync(ivFilePath, iv);
    console.log(`Initialization vector saved to: ${ivFilePath}`);

    // Measure encryption time
    console.time("Encryption Time");
    await encryptFile(inputFilePath, outputFilePath, key, iv);
    console.timeEnd("Encryption Time");

    // Measure decryption time
    console.time("Decryption Time");
    await decryptFile(outputFilePath, decryptedFilePath, key, iv);
    console.timeEnd("Decryption Time");
  } catch (error) {
    console.error("Error:", (error as any).message);
  }
}

// Run the main function
main();
```

Program składa się z następujących funkcji:

- main() – główna funkcja programu, w której następuje szyfrowanie i deszyfrowanie pliku oraz wyświetlenie wyników
- generateKeyAndIV() – generuje losowy klucz i wektor inicjalizujący IV
- encryptFile() – szyfruje plik za pomocą AES w trybie CTR i zapisuje do innego pliku; mierzy czas potrzebny na przeprowadzenie operacji
- decryptFile() – odszyfrowuje plik i zapisuje wynik do pliku; mierzy czas potrzebny na przeprowadzenie operacji

Istotną cechą programu jest strumieniowe przetwarzanie danych, które pozwala na efektywne szyfrowanie i deszyfrowanie nawet bardzo dużych plików bez nadmiernego zużycia pamięci RAM. Wygenerowane wartości klucza i wektora IV są zapisywane do osobnych plików w celu umożliwienia późniejszego odszyfrowania. Program wykorzystuje bibliotekę crypto dostępną w środowisku Node.js.

Wyniki

Szyfrowanie trwało 7,579 sekund

Odszyfrowanie trwało 7,620 sekund

Rozmiar pliku: 10 GB

Format wygenerowanego pliku: txt

Konfiguracja sprzętowa:

- CPU: Apple M1 Max
- RAM: 32 GB

Podsumowanie i wnioski

Szyfrowanie i odszyfrowanie pliku trwało ok. 7,5 sekundy w obu przypadkach co świadczy o podobnej złożoności obliczeniowej tych dwóch operacji. Istotne jest również to, że jest to znacznie krótszy czas niż w przypadku szyfrowanie 3DES w trybie OFB, co świadczy o tym, że algorytm AES jest znacznie bardziej wydajny. Tak dobry wynik możne być spowodowany również użytym językiem programowania – TypeScript zamiast Pythona.