Decrittazione AES: Calcolo Parallelo per la Ricerca della Chiave

Luca Timpano 240236

Contents

1. Algoritmo di criptazione e decriptazione AES

L'AES (*Advanced Encryption Standard*) è un algoritmo di crittografia a blocchi che permette di proteggere dati sensibili tramite la **cifratura**. Nasce nel 1997 per sostituire il *Data Encription Standard (DES)*. L'algoritmo utilizza una serie di operazioni "meccaniche" che prevedono fasi di trasposiozione, sostituzione e mescolatura.

1.1. Funzionamento

L'operazione di crittografia prevede sei passaggi:

1. **Divisione**: Il testo viene suddiviso in blocchi di bit, viene aggiunta la **round key**.
2. **Sostituzione**: iL testo viene sostituito con il testo crittografato attraverso la tabella *Rijndaek S-box*.
3. **Shifting**: Tutte le righe vengono spostate esclusa la prima.
4. **Mescolatura**: le righe vengono mescolate
5. **Round Key**: IL testo viene nuovamente crittografato attraverso la round key
6. **Ripetizione**: Questo processo viene ripetuto in base al tipo di AES

La decrittazione è il processo inverso della crittoggrafia. Consiste nel trasformare i dati crittografati nuovamente in dati leggibili attraverso la stessa chiave utilizzata per la crittografia.

2. Logica utilizzata

Il programma utilizza una logica di ricerca esaustiva per testare tutte le possibili chiavi di crittografia all'interno di un intervallo prefissato. Ogni thread esamina un intervallo diverso, consentendo una parallelizzazione del processo e accelerando la ricerca della chiave corretta.

Librerie

Il progetto fa uso delle librerie standard Java per gestire l'input/output dei file, come java.nio.file.Files e java.nio.file.Path, oltre alle librerie Java per la crittografia AES, come javax.crypto.Cipher e javax.crypto.KeyGenerator.

3. L'algoritmo di decrittazione

L'implementazione dell'algoritmo AES per la decrittazione di un file crittografato è suddivisa in due classi: **SearchKeyConc** e **Decripta**.

3.1. La classe SearchKeyConc

Il metodo pricipale della seguente classe è **startSearch()**. Questo metodo inserisce in un *ThreadGroup* un numero di Thread pari alla variabile *num_Threads*. Al singolo Thread viene assegnato un range in cui lavorare, che viene calcolato dividendo *Integer.MAX_VALUE* per il numero di thread scelto. Una volta effettuata questa operazione, verrà assegnato al primo thread il range (0,Integer.MAX_VALUE/num_Threads), al secondo(Integer.MAX_VALUE/num_Threads, Integer.MAX_VALUE/num_Threads * 2), al terzo (Integer.MAX_VALUE/num_Threads * 2, Integer.MAX_VALUE/num_Threads * 3) e così via. Per automatizzare questo processo è bastato moltiplcare la varibile i del ciclo per Integer.MAX_VALUE/num_Threads per il limite inferiore e $i + 1$ per Integer.MAX_VALUE/num_Threads per il limite superiore. Successivamente vengono fatti partire i singoli Thread.

```
1  public static void startSearch() throws Exception{
2          int num_Threads = 7;
3          Thread[] threads = new Thread[num_Threads];
4
5          byte[] byteCriptati = leggiFile();
6
7          int rangeSize = Integer.MAX_VALUE / num_Threads
       ;
8
9          for (int i = 0; i < num_Threads; i++) {
10             int startRange = i * rangeSize;
11             int endRange = (i + 1) * rangeSize;
12             threads[i] = new Thread(gruppo, new
       Decripta(startRange, endRange, byteCriptati), "T"
       + (i + 1));
13             threads[i].start();
14         }
15     }
```

Code 1. startSearch()

3.2. La classe Decripta

La classe Decripta rappresenta il singolo thread incaricato di testare una serie di chiavi di crittografia all'interno di un intervallo specifico. Il metodo **run()** di ciascun thread esegue il processo di decrittazione e verifica se il testo decriptato contiene una stringa specifica (SISOPcorsoB) che indica il successo della decrittazione.

3.2.1. Costruttore

```
1  public Decripta(long start, long end, byte[] data){
2          this.start = start;
3          this.end = end;
4          this.data = data;
5      }
```

Code 2. Costruttore

3.2.2. Il metodo run()

Il metodo run() è implementato per eseguire il processo di decrittazione all'interno di un singolo thread. All'interno di questo metodo, viene generata una serie di chiavi di crittografia all'interno dell'intervallo specificato e testata per decifrare il file crittografato. Se una chiave riesce a decifrare con successo il file, viene stampata e il gruppo di thread viene interrotto.

```
1  @Override
2      public void run() {
3          try {
4              Thread currentThread = Thread.currentThread
       ();
5
```

```
            for (long i = start; i < end && !currentThread.isInterrupted(); i++) {
                String keyText = generateKey(i);
                SecretKey key = Key(keyText);

                if (decripta(data, key)) {
                    endTimer();
                    System.out.println("Decriptato con successo");
                    System.out.println("Chiave: " + keyText);
                    SearchKeyConc.gruppo.interrupt();
                    return;
                }

                /*
                ...
                */

                }
            }
        }catch (Exception e){
            e.printStackTrace();
        }
    }
```

Code 3. run()

3.2.3. Il metodo decripta()

Questo metodo si occupa di decrittare i byte crittografati utilizzando una chiave di crittografia specifica. Utilizza l'oggetto Cipher per inizializzare la modalità di decrittazione AES e tenta di decifrare i byte crittografati utilizzando la chiave fornita. Se la decrittazione ha successo e il testo decriptato contiene una stringa specifica (SISOP-corsoB), restituisce true, altrimenti restituisce false.

```
public static boolean decripta(byte[] byteCriptati, SecretKey secretKey) throws Exception{
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] byteDecriptati = null;
        try{
            byteDecriptati = cipher.doFinal(byteCriptati);
        }catch (BadPaddingException e){
            return false;
        }

        String decryptedText = new String(byteDecriptati);
        if(decryptedText.contains("SISOP-corsoB")){
            System.out.println("Testo decriptato: " + decryptedText);
            return true;
        }
        return false;
    }
```

Code 4. decripta()

3.2.4. I metodi generateKey() e Key()

I metodi generateKey e Key lavorano insieme per gestire la generazione e la conversione delle chiavi di crittografia utilizzate nel processo di decrittazione. Il metodo generateKey(long key) genera una chiave a partire da un numero intero key. Inizialmente, converte il numero in una stringa utilizzando il metodo String.valueOf(). Quindi, calcola la lunghezza della stringa e determina quante cifre mancano per raggiungere la lunghezza desiderata di 16 caratteri per una chiave AES. Infine, aggiunge le cifre mancanti come zeri all'inizio della stringa e restituisce la chiave generata. Il metodo key(String customKey) converte una stringa rappresentante una chiave di crittografia in un oggetto SecretKey utilizzato dall'algoritmo AES. Innanzitutto, ottiene i byte della stringa chiave utilizzando il metodo getBytes(). Quindi, inizializza un generatore di chiavi AES (KeyGenerator) per generare una chiave a 128 bit. Poiché la chiave AES deve essere esattamente di 16 byte, viene utilizzata la classe SecretKeySpec per creare una nuova chiave segreta utilizzando i byte della stringa come dati chiave. Infine, restituisce l'oggetto SecretKey appena creato.

```
private static String generateKey(long key){
        int keyLength = 16;
        StringBuilder sb = new StringBuilder();
        String x = String.valueOf(key);
        int lenght = x.length();
        while(sb.length() < keyLength - lenght){
            sb.append("0");
        }
        sb.append(x);
        return sb.toString();
    }

    private static SecretKey Key(String customKey) throws NoSuchAlgorithmException {
        byte[] customKeyBytes = customKey.getBytes();
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(128);
        SecretKey secretKey = new SecretKeySpec(customKeyBytes, "AES");
        return secretKey;
    }
```

Code 5. Generazione chiave

4. SpeedUp

L'efficienza dell'applicazione in parallelo dipende dallo speedup, che indica quanto velocemente il programma riesce ad eseguire un compito rispetto all'esecuzione sequenziale. Aumentando il numero di thread, in genere si ottiene uno speedup maggiore. Tuttavia, oltre una certa soglia, l'overhead dell'allocazione e gestione dei thread potrebbe superare i benefici della parallelizzazione, portando a un'efficienza diminuita.