MeshHash
Implementacja w języku Java

Maciej Lipski Kacper Średnicki

13 czerwca 2023

Spis treści

Charakterystyka MeshHash

- Funkcja skrótu
- Kandydat na SHA-3
- Nacisk na bezpieczeństwo
- Szybkość na poziomie SHA-2

Ogólny opis algorytmu

- Działanie blokowe, stan wewnętrzny składa się z P rur
- Każdy blok przetwarzany w P standardowych rundach, po których zmienia się stan wewnętrzny
- Po rundach standardowych następuje finalna runda bloku - przetwarzany jest klucz i stan wewnętrzny
- Po przetworzeniu wszystkich bloków wykonują się rundy finalne całego algorytmu
- Wartość skrótu obliczana poprzez tzw. wyciskanie gąbki

Klasa MeshHash - atrybuty

```
private int P;
private long [] pipes;
private int block_round_counter;
private int key_counter;
private int key_length;
private Counter bit_counter;
private Counter block_counter;
private int hash_bit_length;
private byte[] message;
private long[] key;
private long[] dataStream;
```

Klasa Counter

Opis

- Reprezentuje licznik
- Licznik składa się z czterech słów
- Obiekty tej klasy: *bit_counter*, *block_counter*

Fragmenty kodu

```
public long[] counterArray;
public Counter() { counterArray = new long[4]; }
public void increment()
public long getValue()
```

Klasa MathOperations

Opis

- Obsługa działań matematycznych
- Metody statyczne
- Zastosowany overriding metod

Fragmenty kodu

```
public static long addModulo264(long a, String b)
public static long multiplyModulo264(long a, String b)
public static long rotRi(long word, int i)
```

Klasa MeshHash - główna metoda

Opis

- Przetwarzane wszystkie bloki *dataStream*, każdy z nich przetwarzany przez P standardowych rund
- Finalna runda bloku i finalne rundy całego algorytmu

Metoda *private byte[] computeMeshHash()*

```
private byte[] computeMeshHash() {
    for (int i = 0; i < dataStream.length; i++) {
        long data = dataStream[i];
        while (block_round_counter < P) {
            normalRound(data, block_round_counter); }
        finalBlockRound(); }
    final_rounds();
    byte[] hashValue = computeHashValue();
    return hashValue; }
```

Inicjalizacja algorytmu

Inicjalizacja stanu wewnętrznego algorytmu ma miejsce w konstruktorze klasy MeshHash.

Fragmenty konstruktora

```
private MeshHash(byte[] message, byte[] key, int hash_length) {
    this.P = computeNumberOfPipes(hash_length);
    this.pipes = new long[P];
    int messBits = message.length*8;
    for (int i = 0; i < messBits; i++) {
        bit_counter.increment();
    }

    this.key = convertBytesToWords(key);
    this.key_length = key.length / 8;
    this.dataStream = prepareDataStream();
}
```

Standardowa runda

Opis

- Wykonywana P razy dla bloku
- Inkrementacja *block_round_counter*
- Wykorzystanie metody *SBox()*

Metoda normalRound()

```
private void normalRound(long data, int index) {
    long sBoxInput = pipes[index] ^
    ↪  (MathOperations.multiplyModulo264(index,
    ↪  "0101010101010101") ^ data);
    sBoxInput = MathOperations.rotRi(sBoxInput, 37 * index);
    pipes[index] = MathOperations.addModulo264(SBox(sBoxInput),
    ↪  pipes[(index + 1) % P]);
    block_round_counter++;
}
```

SBox

SBox to zbiór podstawowych operacji matematycznych wykonywanych wielokrotnie w trakcie działania algorytmu.

Metoda SBox()

```
private void normalRound(long data, int index) {
private long SBox(long input) {
    input = MathOperations.multiplyModulo264(input,
    ↪  "9e3779b97f4a7bb9");
    input = MathOperations.addModulo264(input,
    ↪  "5e2d58d8b3bcdef7");
    input = MathOperations.rotRi(input, 37);
    input = MathOperations.multiplyModulo264(input,
    ↪  "9e3779b97f4a7bb9");
    input = MathOperations.addModulo264(input,
    ↪  "5e2d58d8b3bcdef7");
    input = MathOperations.rotRi(input, 37);
    return input; }
```

Finalna runda dla bloku

Opis

- Wykonywana po P normalnych rundach
- Pierwsza czynność to przetworzenie licznika *block_counter*
- Druga czynność to przetworzenie klucza, jeśli został podany

Metoda finalBlockRound()

```
private void finalBlockRound() {
    processBlockCounter();
    if (key_length > 0) {
        processKey();
    }
}
```

Finalna runda dla bloku

Przetwarzanie *block_counter*

- Wyzerowanie *block_round_counter*
- Wykonanie SBox P razy
- Inkrementacja *block_counter*

Metoda processBlockCounter()

```
private void processBlockCounter() {
    block_round_counter = 0;
    for (int i = 0; i < P; i++) {
        pipes[i] = SBox(pipes[i] ^ block_counter.counterArray[i %
        ↪  4]);
    }
    block_counter.increment();
}
```

Finalna runda dla bloku

Przetwarzanie klucza

- Modyfikacja wartości rur za pomocą SBox
- Aktualizacja licznika klucza

Metoda processKey()

```
private void processKey() {
    for (int i = key_counter; i < key_length + key_counter; i +=
    ↪  P) {
        for (int j = 0; j < P; j++) {
            pipes[j] = SBox(pipes[j] ^ key[(i + j) %
            ↪  key_length]); } }
    key_counter = (key_counter + 1) % key_length;
    for (int i = 0; i < P; i++) {
        pipes[i] = SBox(pipes[i] ^ key_length ^
        ↪  (Long.parseUnsignedLong("0101010101010101", 16) *
        ↪  i)); } }
```

Rundy finalne

Wykonywane po przetworzeniu wszystkich bloków. P-krotnie wykonane działania SBox.

Metoda final_rounds()

```
private void final_rounds() {
    for (int i = 0; i < bit_counter.counterArray.length; i++) {
        for (int j = 0; j < P; j++) {
            pipes[j] = SBox(pipes[j] ^
            ↪  bit_counter.counterArray[i] ^
            ↪  (Long.parseUnsignedLong("0101010101010101", 16) *
            ↪  j)); } }
    for (int i = 0; i < P; i++) { {
            pipes[i] = SBox(pipes[i] ^ hash_bit_length ^
            ↪  (Long.parseUnsignedLong("0101010101010101", 16) *
            ↪  i)); } } }
```

Obliczanie skrótu

Metoda computeHashValue()

- Tzw. wyciskanie gąbki
- Dla każdego bajtu z wynikowej tablicy wykonywana jest standardowa runda z danymi wejściowymi równymi 0
- Wykonywana jest ona P razy dla każdego bajtu wynikowego skrótu
- Następnie ma miejsce działanie XOR łączące rury oraz ich AND z odpowiednią stałą
- Jeśli indeks danego bajtu w skrócie modulo P jest równy P-1, należy dla tego bajtu wykonać finalną rundę blokową

Obliczanie skrótu

Metoda computeHashValue()

```
private byte[] computeHashValue() {
    byte[] hashValue = new byte[hash_bit_length / 8];
    for (int i = 0; i < hashValue.length; i++) {
        for (int j = 0; j < P; j++) {
            normalRound(0, j);
        }
        byte temp = 0;
        for (int j = 0; j < P; j += 2)
            temp ^= pipes[j];
        hashValue[i] = (byte) (temp & 255);

        if (i % P == P - 1) finalBlockRound();
    }
    return hashValue;
}
```

Sposób użycia implementacji

Metoda przygotowana dla użytkownika

```
public static String computeMeshHash(String message, String key,
↪  int digestLength)
```

Opis

- Użytkownik nie ma dostępu do konstruktora, ani do metod używanych bezpośrednio przy obliczaniu skrótu
- Wiadomość i klucz przetwarzane do tablicy bajtów, z wykorzystaniem metody z klasy String *getBytes()*
- Tworzony obiekt klasy MeshHash, wywoływana jest dla niego metoda *private byte[] computeMeshHash()*
- Zwraca wartość skrótu w postaci hexadecymalnego Stringa (konwersja z wykorzystaniem *StringBuilder*)

Testy

Opis

- Skróty generowane pomyślnie (różnica z wektorami testowymi)
- Bezkolizyjność
- Powtarzalność dla tej samej wiadomości
- Przy niewielkich zmianach wiadomości, wynikowe skróty znacząco się różnią

Przykład użycia z kluczem „bdan”

```
Ala ma koty:
1a56c1562981ec886cee3526
Ala ma kota:
2ddd1ff3a6f440d1c4782c85
Ala ma koty:
1a56c1562981ec886cee3526
```

Bibliografia

[1] Björn Fay. *MeshHash*. Submission to NIST. 2008. URL: http://ehash.iaik.tugraz.at/uploads/5/5a/Specification_DIN-A4.pdf.

[2] Oracle. *The Java Tutorials - Primitive Data Types*. URL: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.htmll (term. wiz. 08. 05. 2023).