Лунгол О.С.

КН-М922а

Лабораторна робота 1

Хешування

Мета

Дослідити принципи роботи гешування.

Завдання

Дослідити існуючі механізми гешування. Реалізувати алгоритм гешування SHA (будь-якої версії).

Довести коректність роботи реалізованого алгоритму шляхом порівняння результатів з існуючими реалізаціями (напр. утилітою sha1sum).

Хеш-функції сімейства SHA-2 побудовані на основі структури Меркла - Дамгора .

Вихідне повідомлення після доповнення розбивається на блоки, кожен блок – на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл із 64 або 80 ітераціями (раундами). На кожній ітерації 2 слова перетворюються, функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума є значенням хеш-функції. Тим не менш, ініціалізація внутрішнього стану здійснюється результатом обробки попереднього блоку. Тому незалежно обробляти блоки та складати результати не можна.

Текст програми

```
let sha256 = function sha256(word) {
  function rightRotate(value, amount) {
    return (value >>> amount) | (value << (32 - amount));
```

```
}



let mathPow = Math.pow;
let maxWord = mathPow(2, 32);
let lengthProperty = 'length';
let i, j;
let result = '';



let words = [];
let wordBitLength = word[lengthProperty] * 8;
```

Початкове хеш-значення: перші 32 біти дробових частин квадратного кореня з перших 8 простих чисел

```
let hash = (sha256.h = sha256.h || []);
```

Округлення констант: перші 32 біти дробових частин кубічних коренів перших 64 простих чисел

```
let k = (sha256.k = sha256.k || []);
let primeCounter = k[lengthProperty];



let isComposite = {};
for (let cnd = 2; primeCounter < 64; cnd++) {
  if (!isComposite[cnd]) {
    for (i = 0; i < 313; i += cnd) {
      isComposite[i] = cnd;
```

```
      }
      hash[primeCounter] = (mathPow(cnd, 0.5) * maxWord) | 0;
      k[primeCounter++] = (mathPow(cnd, 1 / 3) * maxWord) | 0;
    }
  }

  word += '\x80'; додавання біта
  while ((word[lengthProperty] % 64) - 56) word += '\x00';
  for (i = 0; i < word[lengthProperty]; i++) {
    j = word.charCodeAt(i);
    if (j >> 8) return;
приймати лише символи в діапазоні 0-255
    words[i >> 2] |= j << (((3 - i) % 4) * 8);
  }
  words[words[lengthProperty]] = (wordBitLength / maxWord) | 0;
  words[words[lengthProperty]] = wordBitLength;

  for (j = 0; j < words[lengthProperty]; ) {
    let w = words.slice(j, (j += 16));
    let oldHash = hash;
    hash = hash.slice(0, 8);

    for (i = 0; i < 64; i++) {
      let i2 = i + j;
```

```
  let w15 = w[i - 15],
    w2 = w[i - 2];
  let a = hash[0],
    e = hash[4];
  let temp1 =
    hash[7] +
    (rightRotate(e, 6) ^ rightRotate(e, 11) ^ rightRotate(e, 25)) +
    ((e & hash[5]) ^ (~e & hash[6])) + // ch
    k[i] +
    (w[i] =
      i < 16
        ? w[i]
        : (w[i - 16] +
            (rightRotate(w15, 7) ^ rightRotate(w15, 18) ^ (w15 >>> 3)) +
            w[i - 7] +
            (rightRotate(w2, 17) ^ rightRotate(w2, 19) ^ (w2 >>> 10))) |
          0);


  let temp2 =
    (rightRotate(a, 2) ^ rightRotate(a, 13) ^ rightRotate(a, 22)) +
    ((a & hash[1]) ^ (a & hash[2]) ^ (hash[1] & hash[2]));


  hash = [(temp1 + temp2) | 0].concat(hash);
  hash[4] = (hash[4] + temp1) | 0;
}
```

```
    for (i = 0; i < 8; i++) {
      hash[i] = (hash[i] + oldHash[i]) | 0;
    }
  }

  for (i = 0; i < 8; i++) {
    for (j = 3; j + 1; j--) {
      let b = (hash[i] >> (j * 8)) & 255;
      result += (b < 16 ? 0 : '') + b.toString(16);
    }
  }
  return result;
};
```

Порівняємо результат з вбудованою бібліотекою crypto

```
const { createHash } = require('crypto');

function hash(string) {
  return createHash('sha256').update(string).digest('hex');
}
console.log('Library : ', hash('hello'));
```

```
[Running] node "/Users/alexandralunhol/Desktop/STBP/lab01/src/tempCodeRunnerFile.js"
Library :  2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

[Done] exited with code=0 in 0.146 seconds

[Running] node "/Users/alexandralunhol/Desktop/STBP/lab01/src/main.js"
Function:  2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

Висновок

При порівнянні виявлено що написана функція працює так само як і вбудована.