1 Цель работы

Реализовать алгоритм хэширования SIMD.

2 Описание алгоритма

2.1 Поле F_{257}

Т.к. 257 — простое число, то кольцо Z_{257}по модулю целого числа 257 — поле F_{257}. Операции в этом поле обозначаются как (mod 257). Данное поле было выбрано потому, что можно легко сопоставить байт в элемент поля, а операции могут быть выполнены эффективно в программном обеспечении и аппаратных средствах.

2.2 The Number-Theoretic Transform (NTT)

NTT размера n в поле F_{257}определено как

$$NTT_n: F_{257}^n \to F_{257}^n$$

$$(x_j)_{\text{j=0}}^{\text{n-1}} \to (y_i)_{\text{i=0}}^{\text{n-1}}: y_i = \sum_{j=0}^{n-1} x_j w^{ij}$$

где $n \leq 256$и w — n-ый корень из единицы в поле F_{257}.

2.3 Кольца $Z_{2^{16}}$и $Z_{2^{32}}$

$Z_{2^{16}}$обозначает кольцо по модулю целого числа 2^{16}, и $Z_{2^{32}}$— кольцо по модулю целого числа2^{32}.

Сложение по модулю производится в кольце $Z_{2^{32}}$(32-битные слова), а умножение — в кольце $Z_{2^{16}}$(16-битные слова).

Также, определен циклический сдвиг влево 32-битного слова на *s* бит как $x^{<<<\,s}$.

2.4 Булевы функции

Следующие булевы функции будут использоваться для 32-битных слов.

$$IF(A, B, C) = (A \wedge B) \vee (\neg A \wedge C)$$

$$MAJ(A, B, C) = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)$$

где ∧— логическое «И», ∨— логическое «ИЛИ», ¬— логическое «НЕ».

2.5 Еще некоторые обозначения

Т.к. функция сжатия состоит из повторяющихся простых функций раунда, будет использоваться $X^{(i)}$, чтобы обозначить переменную *X*, связанную с *i*-ым раундом.

Многие переменные могут быть представлены в виде вектора, поэтому $X_{[0..k]}$будет обозначать $[X_0, X_1, \ldots, X_k]$.

2.6 SIMD

Размер выходного блока n, бит	Размер входного блока m, бит	Размер внутреннего состояния p, бит
256	512	512

SIMD — итеративная хэш функция, довольно похожая на семейство функций MD/SHA. Она основана на структуре Меркла-Дамгарда. Основной компонентой является функция сжатия $C: \{0,1\}^p \times \{0,1\}^m \to \{0,1\}^p$. Чтобы вычислить *h(M)*, сообщение *M*, разбивается на k частей по m бит. Функция сжатия построена на основе сети Фейстеля в режиме Девиса-Мейера (рис. 2).

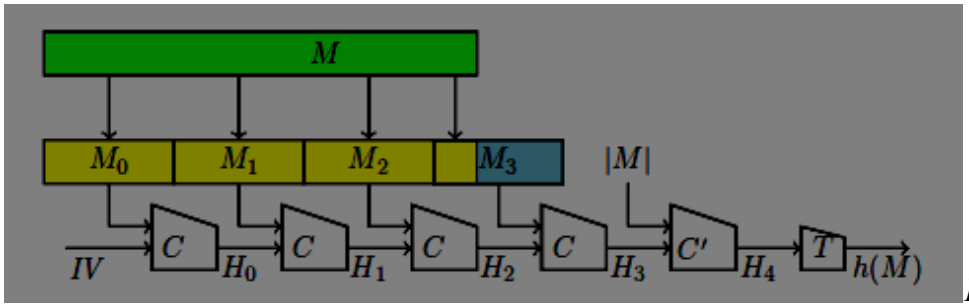

Рис. 1 Общий вид структуры алгоритма SIMD

Внутреннее состояние представляется матрицей из 32-битных слов. Для SIMD-256 это матрица 4 × 4:

$$S_{256} = \begin{bmatrix} A_0 B_0 C_0 D_0 \\ A_1 B_1 C_1 D_1 \\ A_2 B_2 C_2 D_2 \\ A_3 B_3 C_3 D_3 \end{bmatrix}$$

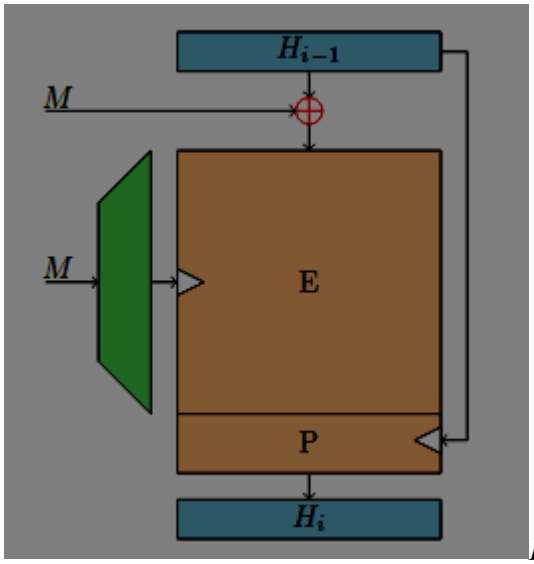

Рис. 2 Структура Девиса-Мейера

2.7 Расширение сообщения

Данная процедура является очень важной частью алгоритма. Расширение сообщения составлено из трех стадий.

2.7.1 Первая стадия: Number-Theoretic Transform

Первая стадия является вычислительно сложной, но это необходимо сделать.

$(Z_{2^8})^{64} \to (F_{257})^{128}$

$(x_j)_{j=0}^{63} \to (y_i)_{i=0}^{127}: y_i = \sum_{j=0}^{63} \quad x_j a^{ij} + a^{127i} (mod 257)$

где a = 139 — это корень 128-ой степени из единицы в поле F_{257}.

2.7.2 Вторая стадия

$I_C \to C \times x (mod 2^{16})$

$I_{185}: x \to 185 \times \tilde{x} (mod 2^{16})$

$I_{233}: x \to 233 \times \tilde{x} (mod 2^{16})$

где $-128 \le \tilde{x} \le 128$и $\tilde{x} = x (mod 257)$

2.7.3 Третья стадия: перестановка

Здесь два 16-битных слова объединяются в одно 32-битное слово. $I_C(x, y) = I_C(x) + 2^{16} I_C(y)$отражает данную процедуру.

Для того, чтобы усилить расширение сообщения, необходимо выполнить перестановку.

$$Z_j^{(i)} = \begin{cases} I_{185}(y[8i + 2j], y[8i + 2j + 1]), & 0 \le i \le 15 \\ I_{233}(y[8i + 2j - 128], y[8i + 2j - 64]), & 16 \le i \le 23 \\ I_{233}(y[8i + 2j - 191], y[8i + 2j - 127]), & 24 \le i \le 31 \end{cases}$$

Наконец, сделаем перестановку строк матрицы Z с помощью следующей функции:

$$W_j^{(i)} = Z_j^{(P(i))}$$

где *P(i)* — фиксированное, и берется из таблицы 1.

Таблица 1

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	4	6	0	2	7	5	3	1	15	11	12	8	9	13	10	14

i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	17	18	23	20	22	21	16	19	30	24	25	31	27	29	28	26

Полное расширение сообщения можно увидеть в таблице 2.

Таблица 2

i	$W_0^{(i)}$	$W_1^{(i)}$	$W_2^{(i)}$	$W_3^{(i)}$
0	$I_{185}(y_{32}, y_{33})$	$I_{185}(y_{34}, y_{35})$	$I_{185}(y_{36}, y_{37})$	$I_{185}(y_{38}, y_{39})$
1	$I_{185}(y_{48}, y_{49})$	$I_{185}(y_{50}, y_{51})$	$I_{185}(y_{52}, y_{53})$	$I_{185}(y_{54}, y_{55})$
2	$I_{185}(y_0, y_1)$	$I_{185}(y_2, y_3)$	$I_{185}(y_4, y_5)$	$I_{185}(y_6, y_7)$
3	$I_{185}(y_{16}, y_{17})$	$I_{185}(y_{18}, y_{19})$	$I_{185}(y_{20}, y_{21})$	$I_{185}(y_{22}, y_{23})$
4	$I_{185}(y_{56}, y_{57})$	$I_{185}(y_{58}, y_{59})$	$I_{185}(y_{60}, y_{61})$	$I_{185}(y_{62}, y_{63})$
5	$I_{185}(y_{40}, y_{41})$	$I_{185}(y_{42}, y_{43})$	$I_{185}(y_{44}, y_{45})$	$I_{185}(y_{46}, y_{47})$
6	$I_{185}(y_{24}, y_{25})$	$I_{185}(y_{26}, y_{27})$	$I_{185}(y_{28}, y_{29})$	$I_{185}(y_{30}, y_{31})$
7	$I_{185}(y_8, y_9)$	$I_{185}(y_{10}, y_{11})$	$I_{185}(y_{12}, y_{13})$	$I_{185}(y_{14}, y_{15})$
8	$I_{185}(y_{120}, y_{121})$	$I_{185}(y_{122}, y_{123})$	$I_{185}(y_{124}, y_{125})$	$I_{185}(y_{126}, y_{127})$
9	$I_{185}(y_{88}, y_{89})$	$I_{185}(y_{90}, y_{91})$	$I_{185}(y_{92}, y_{93})$	$I_{185}(y_{94}, y_{95})$
10	$I_{185}(y_{96}, y_{97})$	$I_{185}(y_{98}, y_{99})$	$I_{185}(y_{100}, y_{101})$	$I_{185}(y_{102}, y_{103})$
11	$I_{185}(y_{64}, y_{65})$	$I_{185}(y_{66}, y_{67})$	$I_{185}(y_{68}, y_{69})$	$I_{185}(y_{70}, y_{71})$
12	$I_{185}(y_{72}, y_{73})$	$I_{185}(y_{74}, y_{75})$	$I_{185}(y_{76}, y_{77})$	$I_{185}(y_{78}, y_{79})$
13	$I_{185}(y_{104}, y_{105})$	$I_{185}(y_{106}, y_{107})$	$I_{185}(y_{108}, y_{109})$	$I_{185}(y_{110}, y_{111})$
14	$I_{185}(y_{80}, y_{81})$	$I_{185}(y_{82}, y_{83})$	$I_{185}(y_{84}, y_{85})$	$I_{185}(y_{86}, y_{87})$
15	$I_{185}(y_{112}, y_{113})$	$I_{185}(y_{114}, y_{115})$	$I_{185}(y_{116}, y_{117})$	$I_{185}(y_{118}, y_{119})$
16	$I_{233}(y_8, y_{72})$	$I_{233}(y_{10}, y_{74})$	$I_{233}(y_{12}, y_{76})$	$I_{233}(y_{14}, y_{78})$
17	$I_{233}(y_{16}, y_{80})$	$I_{233}(y_{18}, y_{82})$	$I_{233}(y_{20}, y_{84})$	$I_{233}(y_{22}, y_{86})$
18	$I_{233}(y_{56}, y_{120})$	$I_{233}(y_{58}, y_{122})$	$I_{233}(y_{60}, y_{124})$	$I_{233}(y_{62}, y_{126})$
19	$I_{233}(y_{32}, y_{96})$	$I_{233}(y_{34}, y_{98})$	$I_{233}(y_{36}, y_{100})$	$I_{233}(y_{38}, y_{102})$
20	$I_{233}(y_{48}, y_{112})$	$I_{233}(y_{50}, y_{114})$	$I_{233}(y_{52}, y_{116})$	$I_{233}(y_{54}, y_{118})$
21	$I_{233}(y_{40}, y_{104})$	$I_{233}(y_{42}, y_{106})$	$I_{233}(y_{44}, y_{108})$	$I_{233}(y_{46}, y_{110})$
22	$I_{233}(y_0, y_{64})$	$I_{233}(y_2, y_{66})$	$I_{233}(y_4, y_{68})$	$I_{233}(y_6, y_{70})$
23	$I_{233}(y_{24}, y_{88})$	$I_{233}(y_{26}, y_{90})$	$I_{233}(y_{28}, y_{92})$	$I_{233}(y_{30}, y_{94})$
24	$I_{233}(y_{49}, y_{113})$	$I_{233}(y_{51}, y_{115})$	$I_{233}(y_{53}, y_{117})$	$I_{233}(y_{55}, y_{119})$
25	$I_{233}(y_1, y_{65})$	$I_{233}(y_3, y_{67})$	$I_{233}(y_5, y_{69})$	$I_{233}(y_7, y_{71})$
26	$I_{233}(y_9, y_{73})$	$I_{233}(y_{11}, y_{75})$	$I_{233}(y_{13}, y_{77})$	$I_{233}(y_{15}, y_{79})$
27	$I_{233}(y_{57}, y_{121})$	$I_{233}(y_{59}, y_{123})$	$I_{233}(y_{61}, y_{125})$	$I_{233}(y_{63}, y_{127})$
28	$I_{233}(y_{25}, y_{89})$	$I_{233}(y_{27}, y_{91})$	$I_{233}(y_{29}, y_{93})$	$I_{233}(y_{31}, y_{95})$
29	$I_{233}(y_{41}, y_{105})$	$I_{233}(y_{43}, y_{107})$	$I_{233}(y_{45}, y_{109})$	$I_{233}(y_{47}, y_{111})$
30	$I_{233}(y_{33}, y_{97})$	$I_{233}(y_{35}, y_{99})$	$I_{233}(y_{37}, y_{101})$	$I_{233}(y_{39}, y_{103})$
31	$I_{233}(y_{17}, y_{81})$	$I_{233}(y_{19}, y_{83})$	$I_{233}(y_{21}, y_{85})$	$I_{233}(y_{23}, y_{87})$

2.8 Лестница Фейстеля

Функция сжатия основана на структуре Фейстеля, раунд которой представлен на рисунке 3.

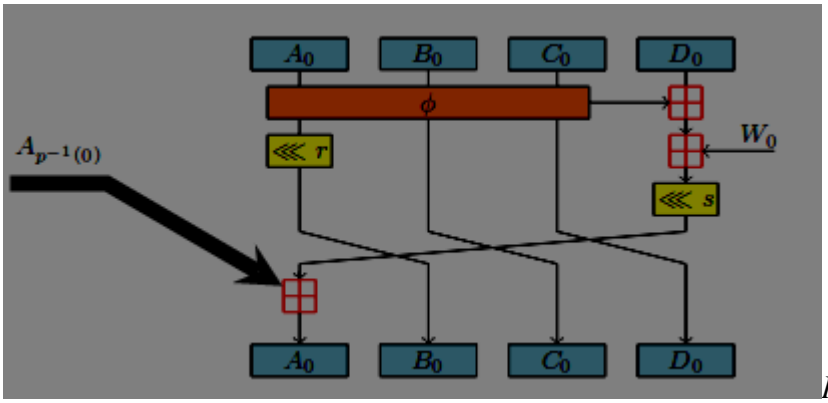

Рис. 3 Схема одного раунда

$$A_j^{(i)} = (D_j^{(i-1)} + W_j^{(i)} + \varphi^{(i)}(A_j^{(i-1)}, B_j^{(i-1)}, C_j^{(i-1)}))^{<<<s^{(i)}} + A_{p^{(i)}(j)}^{(i-1)\ <<<r^{(i)}} (mod 2^{32})$$

где $\varphi^{(i)}$— одна из булевых функций, описанных в п.2.4.

$$B_j^{(i)} = A_j^{(i-1)^{<<<r^{(i)}}}$$

$$C_j^{(i)} = B_j^{(i-1)}$$

$$D_j^{(i)} = C_j^{(i-1)}$$

Для i-го шага: $p^{(imod3)}$

$p^{(0)}(j) = j \oplus 1$
$p^{(1)}(j) = j \oplus 2$
$p^{(2)}(j) = j \oplus 3$

j	0	1	2	3
$p^{(0)}(j)$	1	0	3	2
$p^{(1)}(j)$	2	3	0	1
$p^{(2)}(j)$	3	2	1	0

Один шаг функции можно представить в виде:

$$\text{Step}\left(\begin{bmatrix} A_0 & B_0 & C_0 & D_0 \\ A_1 & B_1 & C_1 & D_1 \\ A_2 & B_2 & C_2 & D_2 \\ A_3 & B_3 & C_3 & D_3 \end{bmatrix}, \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix}, \phi, r, s, p\right)$$
$$= \begin{bmatrix} (D_0 \boxplus W_0 \boxplus \phi(A_0, B_0, C_0))^{\lll s} \boxplus A_{p(0)}^{\lll r} & A_0^{\lll r} & B_0 & C_0 \\ (D_1 \boxplus W_1 \boxplus \phi(A_1, B_1, C_1))^{\lll s} \boxplus A_{p(1)}^{\lll r} & A_1^{\lll r} & B_1 & C_1 \\ (D_2 \boxplus W_2 \boxplus \phi(A_2, B_2, C_2))^{\lll s} \boxplus A_{p(2)}^{\lll r} & A_2^{\lll r} & B_2 & C_2 \\ (D_3 \boxplus W_3 \boxplus \phi(A_3, B_3, C_3))^{\lll s} \boxplus A_{p(3)}^{\lll r} & A_3^{\lll r} & B_3 & C_3 \end{bmatrix}$$

Блок из 8 шагов называется раундом и он имеет набор фиксированных параметров $\pi_{[0..3]}$:

$\phi^{(i)}$	$r^{(i)}$	$s^{(i)}$
IF	π_0	π_1
IF	π_1	π_2
IF	π_2	π_3
IF	π_3	π_0
MAJ	π_0	π_1
MAJ	π_1	π_2
MAJ	π_2	π_3
MAJ	π_3	π_0

Round	π_0	π_1	π_2	π_3
0	3	23	17	27
1	28	19	22	7
2	29	9	15	5
3	4	13	10	25

2.9 Финальная функция сжатия

После того, как все блоки сообщения были сжаты, необходимо провести аналогичную процедуру. В качестве входных данных будет являться вектор, содержащий длину сообщения, а остальные байты равны нулю. Еще отличие в том, что для блока сообщения выполнялось: $O(M) = NTT_{128}(M + X^{127})$, а для финальной функции будет выполнено: $O'(M) = NTT_{128}(M + X^{127} + X^{125})$.

2.10 Инициализирующий вектор

В качестве инициализирующего вектора берутся следующие 64 байта:

SIMD-256 IV				
$A_{0..3}$	4d567983	07190ba9	8474577b	39d726e9
$B_{0..3}$	aaf3d925	3ee20b03	afd5e751	c96006d3
$C_{0..3}$	c2c2ba14	49b3bcb4	f67caf46	668626c9
$D_{0..3}$	e2eaa8d2	1ff47833	d0c661a5	55693de1

3 Описание реализации

3.1 Класс Reader

В конструктор передается имя файла, с которым будет производиться работа.

Обеспечивает чтение файла с помощью метода:

```
byte[] getBytes(int byteCount)
```

В качестве параметра указывается число байт, необходимых для считывания. Возвращает массив считанных байт. Если количество байт в файле меньше, указанного параметра, считывает то, что осталось, а остальное заполняет нулевыми байтами.

3.2 Класс Constants

Данный класс содержит все необходимые константы и таблицы.

3.3 Класс Converter

Содержит статические методы:

```
int combine(byte[] bytes)
```

Объединяет массив из четырех байт в целое число.

```
int[] byte2int(byte[] x)
```

Преобразует массив байт в массив целых чисел, объединяя четыре байта в одно число. Размер полученного массива, естественно, меньше.

```
byte[] int2byte(int[] x)
```

Аналогично предыдущему методу, преобразует массив целых чисел в массив байт.

3.4 Класс HelpFunctions

Здесь присутствуют различные вспомогательные функции разного рода, не относящиеся напрямую к алгоритму.

```
byte[] getCapacityBytes(long x)
```

Метод преобразует размер сообщения в необходимые входные данные, т. е. в массив байт, заполняя пустые нулями.

```
int[] xor(int[] a, int[] b)
```

Выполняет операцию сложения по модулю 2 для массивов целых чисел.

```
void writeHashIntoFile(String filename, int[] s)
```

Выполняет запись результата хеширования в указанный файл.

```
int modulPow(int a, int b, int modul)
```

Возводит число a в степень b по модулю целого числа.

3.5 Класс Simd

Является главным классом. Основной метод:

void getHash(String filename, String output)

принимает на вход имя файла, содержащее сообщение, и имя файла, куда необходимо записать результат.

Метод **byte[] getIV()** загружает инициализирующий вектор из фиксированного файла.

Функция сжатия реализована в следующем методе:

```
int[] compression(int[] initVector, byte[] m, int f) {
```

```
    int[][] w = messageExpansion(m, f);
    int[] msg = Converter.byte2int(m);
    int[] s = HelpFunctions.xor(initVector, msg);

    //steps: 0...31
    int[] subPi = new int[4];
    for (int i = 0; i < 4; i++) {
         subPi = Constants.PI[i];
         s = round(s, w, i, subPi);
    }

    //steps: 32...35
    int[] inputVector = new int[4];
    int nStep = Constants.STEPS_COUNT;
    subPi = Constants.PI[3];
    for (int i = 0; i < 4; i++) {
         inputVector[0] = initVector[4*i];
         inputVector[1] = initVector[4*i+1];
         inputVector[2] = initVector[4*i+2];
         inputVector[3] = initVector[4*i+3];
         s = step(s,inputVector,Constants.IF,subPi[i],subPi[(i+1) %4],nStep);
         nStep++;
    }

    return s;
}
```

Метод, реализующий функцию раунда:

```
int[] round(int[] s, int[][] w, int i, int[] pi) {
    s = step(s, w[8*i],   Constants.IF,  pi[0], pi[1], 8*i);
    s = step(s, w[8*i+1], Constants.IF,  pi[1], pi[2], 8*i+1);
    s = step(s, w[8*i+2], Constants.IF,  pi[2], pi[3], 8*i+2);
    s = step(s, w[8*i+3], Constants.IF,  pi[3], pi[0], 8*i+3);
    s = step(s, w[8*i+4], Constants.MAJ, pi[0], pi[1], 8*i+4);
    s = step(s, w[8*i+5], Constants.MAJ, pi[1], pi[2], 8*i+5);
    s = step(s, w[8*i+6], Constants.MAJ, pi[2], pi[3], 8*i+6);
    s = step(s, w[8*i+7], Constants.MAJ, pi[3], pi[0], 8*i+7);
    return s;
}
```

Функция Step, описанная в п.2.8:

```
int[] step(int[] s, int[] w, int mode, int pi1, int pi2, int nStep) {
    int[] res = new int[16];
    long mod = 0x100000000L;
    for (int i = 0; i < 4; i++) {
         if (mode == Constants.IF) {
              long tmp = s[12+i] + w[i] + IF(s[i], s[4+i], s[8+i]);
              res[i] = (int) (tmp % mod);
         } else if (mode == Constants.MAJ) {
```

```
                long tmp = s[12+i] + w[i] + MAJ(s[i], s[4+i], s[8+i]);
                res[i] = (int) (tmp % mod);
            }
            res[i] = Integer.rotateLeft(res[i], pi2);
            int index = -1;
            switch (nStep % 3) {
                case 0:
                    index = i ^ 1;
                    break;
                case 1:
                    index = i ^ 2;
                    break;
                case 2:
                    index = i ^ 3;
                    break;
            }
            res[i] = (int) ((res[i] + Integer.rotateLeft(s[index], pi1)) % mod);
        }
        for (int i = 4; i < 8; i++) {
            res[i] = Integer.rotateLeft(s[i-4], pi1);
        }
        for (int i = 8; i < 16; i++) {
            res[i] = s[i-4];
        }
        return res;
    }
```

Также присутствуют булевы функции:

```
int IF(int a, int b, int c)  {
    return (a & b) | (~a & c);
}

int MAJ (int a, int b, int c) {
    return (a & b) | (a & c) | (b & c);
}
```

Метод, выполняющий расширение сообщения. В качестве параметра *f* указывается 0, если *m* — обычный блок сообщения, и — 1, если *m* — финальный блок, содержащий размер.

```
int[][] messageExpansion(byte[] m, int f) {
    int[] y;
    if (f == 0) {
        y = ntt(m, 0, 1);
    } else {
        y = ntt(m, 1, 1);
    }

    int[][] z = new int[32][4];
    for (int i = 0; i < 32; i++) {
        for (int j = 0; j < 4; j++) {
            z[i][j] = zCalculation(i, j, y);
        }
    }
```

```
        int[][] w = new int[32][4];
        for (int i = 0; i < 32; i++) {
            for (int j = 0; j < 4; j++) {
                w[i][j] = z[Constants.Z_PERMUTATION[i]][j];
            }
        }
        return w;
    }
```

Метод, выполняющий Number-Theoretic Transform (NTT). Параметр *x125* равен 1 в случае обработки финального блока.

```
int[] ntt(byte[] x, int x125, int x127)
```

```
int zCalculation(int i, int j, int[] y)
```

Наконец, методы, необходимые для вычисления индексов, применяемые при перестановках. В качестве параметра *coef* указывается число 185 или 233.

```
int I_coef(int x, int y, int coef)
```

```
int I_coef(int x, int coef)
```

4 Пример

Воспользуемся примером, приведенным автором данного алгоритма.

```
00000000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ................
00000010 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F ................
00000020 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F  !"#$%&'()*+,-./
00000030 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789:;<=>?
```

Рис. 4 Входные данные

```
00000000 83 79 56 4D A9 0B 19 07 7B 57 74 84 E9 26 D7 39 .yVM....{Wt..&.9
00000010 25 D9 F3 AA 03 0B E2 3E 51 E7 D5 AF D3 06 60 C9 %......>Q.....`.
00000020 14 BA C2 C2 B4 BC B3 49 46 AF 7C F6 C9 26 86 66 .......IF.|..&.f
00000030 D2 A8 EA E2 33 78 F4 1F A5 61 C6 D0 E1 3D 69 55 ....3x...a...=iU
```

Рис. 5 Инициализирующий вектор

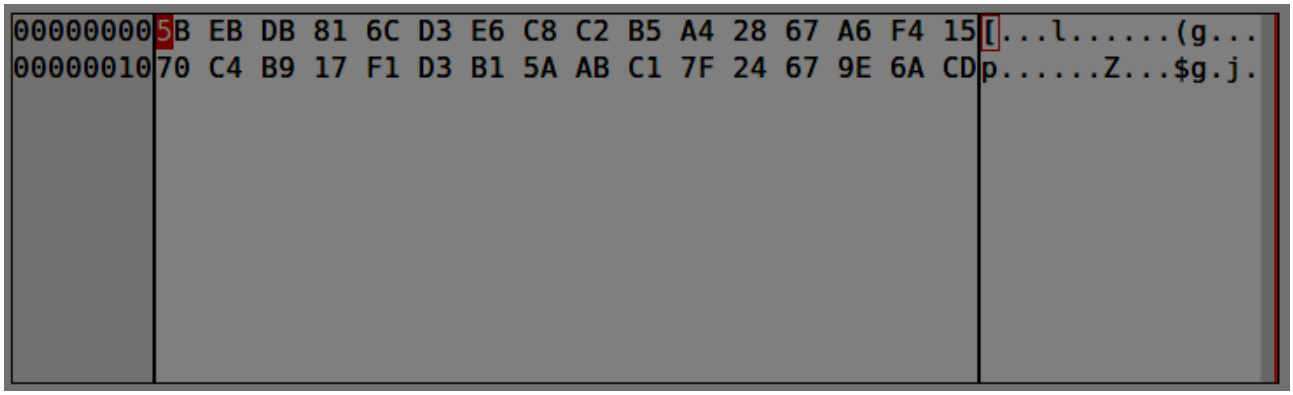

Рис. 6 Результирующая последовательность

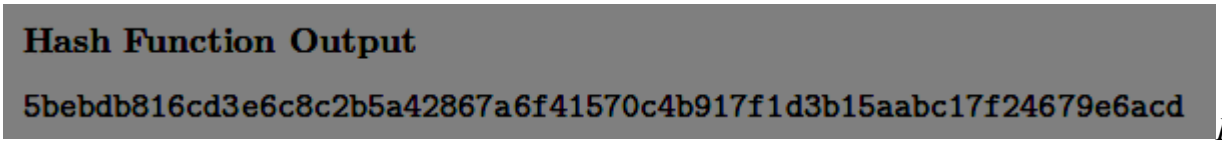

Рис. 7 Результат, приведенный автором теста