VIKTORIA: A NEW PARADIGM FOR HASH FUNCTIONS

Edimar Veríssimo
Cryptographer, Jacareí - São Paulo, Brazil
February 2020
yugi386@yahoo.com.br

Abstract: Viktoria hash is a compression function that generates a set of 512 bits from an arbitrary size input (limit of 2^{480}-1 bytes). This hash function contains some internal routines clearly inspired by AES and RC4 symmetric algorithms [14]. The new paradigm presents two major innovations: a fast preprocessing that initiates an internal state of $256!^{2}$ permutations and a post-processing that guarantees a minimum number of executed rounds of 2^{13}. The pre-processing allows to differentiate very similar messages in the first runs of the algorithm. In the post-processing we have a safety barrier provided by a large number of rounds through a different structure of the main processing. The Viktoria algorithm seems to inaugurate a new design model in the construction of robust hash functions for some reasons, among them we highlight: the customization of the internal state according to each message, the elegance and efficiency of its main function and also a supposed high margin of safety provided by its post-processing function. Viktoria hash can also process bit oriented messages (whose last byte size is not complete) and generate larger hashes (1024, 1536, 2048 or larger) always as multiples of 512.

Key words: viktoria, compression function, collision, hash function, irreversible function, digital mapping of a message.

1. INTRODUCTION

With the increasing advent of electronic transactions, it is necessary to have alternatives of hash functions that allow the generation of reliable summaries of a document, guarantee the confirmation of knowledge between two or more parties, allow the derivation of keys and the generation of pseudo-random numbers [1].

The advancement of computer technology and cryptoanalytic attacks makes it essential to search for constant innovations in this segment of cryptography. Weaknesses are regularly discovered in the better known hash function classes such as MD5 and SHA-1 [2][3].

With this modest work we present a new hash function: Viktoria. It is based on the structure of Merkle-Damgard [5][8] but has two extra functions at the end of the algorithm processing, plus a pre-reading of the message before the main processing which helps to differentiate similar inputs quickly. Viktoria has a very large internal state so it can behave like a pseudo-random number generator with a maximum period of $256!^2 * 2^{512}$, something around $9,86 * 10^{1167}$.

In part 2 we present in detail the Viktoria hash function starting with a more general description and then moving on to the more detailed functions. In this part we present the whole logic of the algorithm highlighting its most important parts.

In part 3 we present a justification for the design of the Viktoria hash function (more precise for the mixword function). It is clear from that description why we choose the internal structure this way. The mixword function works by dividing the whole block (512 bits) into 4 sub blocks of 128 bits and uses 3 of the 4 sub blocks to change the other sub-block. This way a high data diffusion is guaranteed in each round.

In part 4 we present a logical rationale and some tests to justify the design of the three main functions of the algorithm. In this part we present the read_block() function that uses an intelligent mechanism to read the bytes of the message. They are read not as they are but are translated through a dynamic Sbox. We also present the diffusion mechanism of the mixword function validated by statistical tests. Finally we present the permutation_block() function that works with dynamic Pbox's (permutation boxes).

In part 5 some statistical tests are made using the Dieharder battery test tool. These tests try to prove that the outputs of the Viktoria hash function behave in a pseudo-random way. Reduced versions of the algorithm (with a minimum number of rounds) and the full version have been tested.

In part 6 of this work we compared the Viktoria algorithm with the SHA2-512 and SHA3-512 hash functions. The first comparison refers to the diffusion of bits in the three algorithms using the hashes of all possible 16-bit messages. The second comparison is based on a test to check the resistance to differential cryptoanalysis in the three algorithms. The XOR operations between the hashes of 16384 very similar files are analyzed. And the last comparison refers to the performance of the three algorithms.

In part 7 we present a brief description of how to compile the Viktoria algorithm and how to use it. This part shows the parameters that can be used to extract the hash from files and how to use Viktoria hash to process bit oriented files (with incomplete byte at the end of the file).

The conclusion reaffirms what was verified in the tests performed to verify the effectiveness of the Viktoria hash function. Finally we present in Annex XIX the complete source code in C language (optimized but not in its entirety).

2. DESCRIPTION OF THE ALGORITHM

The Viktoria hash algorithm works with three phases of message processing:

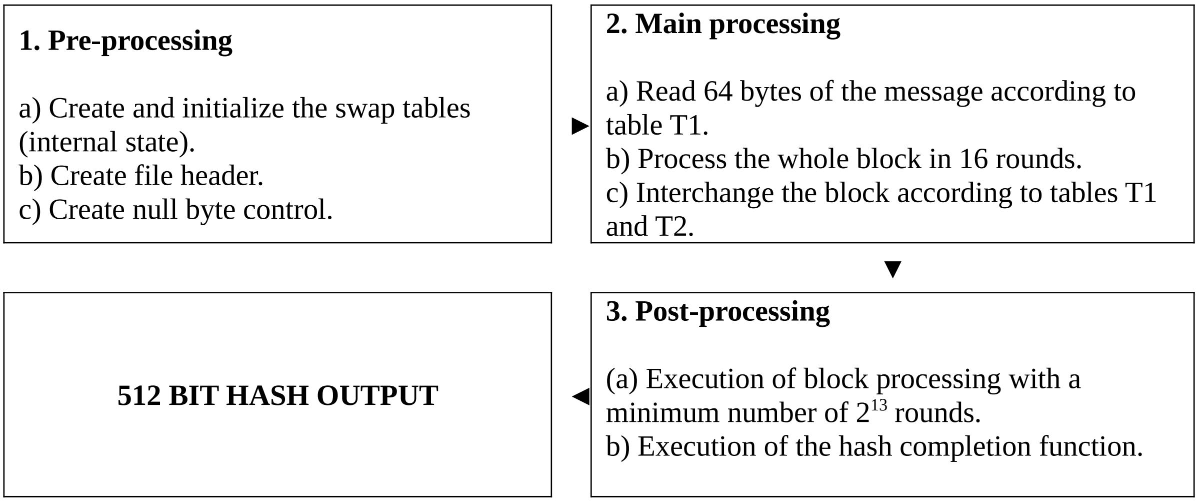
a) Pre-processing: at this stage the internal states of two 256-byte exchange tables1 are exchanged according to the content of the entire message. It is important to note here that these internal states of the algorithm fully affect the reading and processing of the file data so that very similar messages are differentiated more quickly. In addition, the message size management mechanism generates a header that is processed with the mixword() function before reading data from the file. There is also a mechanism to fill the initial block when the message size is not a multiple of 64 and a special control to handle binary messages not byte oriented.

b) Central processing: is executed by three distinct functions: read_block(), mixword() and permutation_block(). These functions read 64 bytes of the message, process the contents of this block in 16 rounds and permute bytes of the whole block, respectively. Each block read from the file passes through a different Sbox[1] and at the end a different permutation is performed over the 64 bytes of the block.

c) Post-processing: this step performs an operation called mixword_final() and a final hash calculation function. The mixword_final() function is similar but more complex than the mixword() function and does not have a certain number of runs to perform the processing. The finalize() function sets the intermediate hash to the final 512-bit output. If required Viktoria hash can generate varied hash sizes with 1024, 1536, 2048 or larger, always as multiples of 512.

Graphically we can represent the entire Viktoria function in the following diagram:

[1] Non-linear and dynamic replacement box.



1. Pre-processing

a) Create and initialize the swap tables (internal state).
b) Create file header.
c) Create null byte control.

2. Main processing

a) Read 64 bytes of the message according to table T1.
b) Process the whole block in 16 rounds.
c) Interchange the block according to tables T1 and T2.

3. Post-processing

(a) Execution of block processing with a minimum number of 2^{13} rounds.
b) Execution of the hash completion function.

512 BIT HASH OUTPUT

Chart 1

The Viktoria function has a very simple macro structure. First we create the interchange tables T1 and T2 (see Annex I). Then we initialize the swap tables according to the content of the message (see Annex II). Then we form the header of the file and check data regarding its size (the process is described in Annex III). At this point the pre-processing is finished and the message is prepared to be processed and generate the hash value.

2.1 The main algorithm

Viktoria hash has a core of 3 functions that together form the heart of the algorithm. At the end of processing a special routine is performed.

ALGORITHM 1

```
From beginning to end of the file
{
        read_block()
        mixword()
        permutation_block()
}
mixword_final()
finalizes()
```

2.2 Read_block() function

The read_block() function reads 64 bytes of the input message from table T1 and makes an XOR operation with the result of processing the previous block.

ALGORITHM 2

```
0 to 64 do:
        BLOCK[ct] = T1[read_block[ct]] XOR BLOCK[ct]
```

2.3 Mixword function()

The mixword() function is the heart of the Viktoria algorithm. The processing of this function can be better understood through a graphical schema. The data block read from the file has 64 bytes and can be represented as follows:

SUB-BLOCK A				SUB-BLOCK B				SUB-BLOCK C				SUB-BLOCK D			
Current				T0 (without SBOX)				T1				T2			
0	1	2	3	16	17	18	19	32	33	34	35	48	49	50	51
4	5	6	7	20	21	22	23	36	37	38	39	52	53	54	55
8	9	10	11	24	25	26	27	40	41	42	43	56	57	58	59
12	13	14	15	28	29	30	31	44	45	46	47	60	61	62	63

Table 1

These 64 bytes divided into these 4 sub-blocks were read from the message. However they do not exactly represent the bytes of the message. These same bytes were changed by the **T1** table that is working in the read_block() function as a SBOX[2]. The logic of the first operation of the mixword() function is to use the sub blocks **B**, **C**, and **D** to change the sub-block. **A**. Then the sub blocks are rotated left in the next round of the mixword() function. This function works with 16 rounds, changing each block 4 times.

Each sub-block has 16 bytes and they will be identified individually by a hexadecimal number according to graph 3. First we will form 4 words of 32 bits as follows:

SUB-BLOCK A				SUB-BLOCK B				SUB-BLOCK C				SUB-BLOCK D			
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	A	B	8	9	A	B	8	9	A	B	8	9	A	B
C	D	E	F	C	D	E	F	C	D	E	F	C	D	E	F

Table 2

$$word[0] = A_0 * 256^3 + A_4 * 256^2 + A_8 * 256^1 + A_C * 256^0$$
$$word[1] = A_1 * 256^3 + A_5 * 256^2 + A_9 * 256^1 + A_D * 256^0$$
$$word[2] = A_2 * 256^3 + A_6 * 256^2 + A_A * 256^1 + A_E * 256^0$$
$$word[3] = A_3 * 256^3 + A_7 * 256^2 + A_B * 256^1 + A_F * 256^0$$

We will use sub-blocks **B**, **C** and **D** to change sub-block **A**. Each word represents a column of sub-block A. These values are used when the number of the processing lap in module 4 is zero. This way we start with the first value of each column. If it were equal to 1 the word[0] would be equal to **$A_4 * 256^3 + A_8 * 256^2 + A_C * 256^1 + A_0 * 256^0$**. If the result of module 4 was equal to 2 the word[0] would be equal to **$A_8 * 256^3 + A_C * 256^2 + A_0 * 256^1 + A_4 * 256^0$**. If the result of module 4 were equal to 3 the word[0] would equal **$A_C * 256^3 + A_0 * 256^2 + A4 * 256^1 + A_8 * 256^0$**. The same applies, analogously, to the other words.

2.3.1 Adding the elements of sub-block B

First let's do a XOR operation with the 4 words and all 16 elements of sub-block **B**:

$$word[0] = word[0] \text{ XOR } (B_0 * 256^3 + B_5 * 256^2 + B_A * 256^1 + B_F * 256^0)$$
$$word[1] = word[1] \text{ XOR } (B_1 * 256^3 + B_6 * 256^2 + B_B * 256^1 + B_C * 256^0)$$
$$word[2] = word[2] \text{ XOR } (B_2 * 256^3 + B_7 * 256^2 + B_8 * 256^1 + B_D * 256^0)$$
$$word[3] = word[3] \text{ XOR } (B_3 * 256^3 + B_4 * 256^2 + B_9 * 256^1 + B_E * 256^0)$$

We can graphically represent this operation in relation to sub-block **B** in this way:

Word 0				Word 1				Word 2				Word 3			
0	1	2	3	0	**1**	2	3	0	1	**2**	3	0	1	2	**3**
4	**5**	6	7	4	5	**6**	7	4	5	6	**7**	**4**	5	6	7
8	9	**A**	B	8	9	A	**B**	**8**	9	A	B	8	**9**	A	B
C	D	E	**F**	**C**	D	E	F	C	**D**	E	F	C	D	**E**	F

Table 3

[2] A SBOX is a non-linear replacement box. It is similar to the substitution box used by the AES algorithm but the T1 table is dynamic, that is, it changes over the time of the algorithm processing.

Note that each word is modified by parts of each of the 4 words in sub-block **B**.

2.3.2 Adding the elements of sub-block C

The next operation is a sum of each of these 4 words with others formed by the elements of sub-block **C**, only this time not a byte of the sub-block, but the mapping of this byte in table **T1** that here works as a SBOX.

$$word[0] = word[0] + (T1[C_0] * 256^3 + T1[C_6] * 256^2 + T1[C_8] * 256^1 + T1[C_E] * 256^0) \text{ MOD } 2^{32}$$
$$word[1] = word[1] + (T1[C_1] * 256^3 + T1[C_7] * 256^2 + T1[C_9] * 256^1 + T1[C_F] * 256^0) \text{ MOD } 2^{32}$$
$$word[2] = word[2] + (T1[C_2] * 256^3 + T1[C_4] * 256^2 + T1[C_A] * 256^1 + T1[C_C] * 256^0) \text{ MOD } 2^{32}$$
$$word[3] = word[3] + (T1[C_3] * 256^3 + T1[C_5] * 256^2 + T1[C_B] * 256^1 + T1[C_D] * 256^0) \text{ MOD } 2^{32}$$

We represent it graphically this way:

Word 0				**Word 1**				**Word 2**				**Word 3**			
0	1	2	3	0	**1**	2	3	0	1	**2**	3	0	1	2	**3**
4	5	**6**	7	4	5	6	**7**	**4**	5	6	7	4	**5**	6	7
8	9	A	B	8	**9**	A	B	8	9	**A**	B	8	9	A	**B**
C	D	**E**	F	C	D	E	**F**	**C**	D	E	F	C	**D**	E	F

Table 4

2.3.3 Adding the elements of sub-block D

Finally we have the interaction with sub-block **D** through an operation with an element of this sub-block mapped in vector **T2**:

$$word[0] = word[0] \text{ XOR } (T2[D_0] * 256^3 + T2[D_7] * 256^2 + T2[D_A] * 256^1 + T2[D_D] * 256^0)$$
$$word[1] = word[1] \text{ XOR } (T2[D_1] * 256^3 + T2[D_4] * 256^2 + T2[D_B] * 256^1 + T2[D_E] * 256^0)$$
$$word[2] = word[2] \text{ XOR } (T2[D_2] * 256^3 + T2[D_5] * 256^2 + T2[D_8] * 256^1 + T2[D_F] * 256^0)$$
$$word[3] = word[3] \text{ XOR } (T2[D_3] * 256^3 + T2[D_6] * 256^2 + T2[D_9] * 256^1 + T2[D_C] * 256^0)$$

Graphically this operation can be represented as such:

Word 0				**Word 1**				**Word 2**				**Word 3**			
0	1	2	3	0	**1**	2	3	0	1	**2**	3	0	1	2	**3**
4	5	6	**7**	**4**	5	6	7	4	**5**	6	7	4	5	**6**	7
8	9	**A**	B	8	9	A	**B**	**8**	9	A	B	8	**9**	A	B
C	**D**	E	F	C	D	**E**	F	C	D	E	**F**	**C**	D	E	F

Table 5

This part of the algorithm was inspired by the shiftRow() function of the AES algorithm. The difference is that here we work with a block of 512 bits divided into 4 parts. The sub blocks **B**, **C** and **D** are used to modify the sub-block. **A**. Each of these sub blocks has 128 bits, totaling a block of 512 bits.

2.3.4 Modifying T1 and T2 tables

Table **T1** is dynamic. At this point 16 of its elements will be changed of position. The data used to change this vector comes from **sub-block D**. Table **T2** is also dynamic. The data used to change this vector comes from **sub-block C**. The changes are made in a simple way through the following operations.

Changes to the interchange table T1:

ALGORITHM 3

```
TMP = T1[T2[BLOCK[48]]]
T1[T2[BLOCK[48]]] = T1[T2[BLOCK[55]]]
T1[T2[BLOCK[55]]] = T1[T2[BLOCK[58]]]
T1[T2[BLOCK[58]]] = T1[T2[BLOCK[61]]]
```

```
T1[T2[BLOCK[61]]] = T1[T2[BLOCK[49]]]
T1[T2[BLOCK[49]]] = T1[T2[BLOCK[52]]]
T1[T2[BLOCK[52]]] = T1[T2[BLOCK[59]]]
T1[T2[BLOCK[59]]] = T1[T2[BLOCK[62]]]
T1[T2[BLOCK[62]]] = T1[T2[BLOCK[50]]]
T1[T2[BLOCK[50]]] = T1[T2[BLOCK[53]]]
T1[T2[BLOCK[53]]] = T1[T2[BLOCK[56]]]
T1[T2[BLOCK[56]]] = T1[T2[BLOCK[63]]]
T1[T2[BLOCK[63]]] = T1[T2[BLOCK[51]]]
T1[T2[BLOCK[51]]] = T1[T2[BLOCK[54]]]
T1[T2[BLOCK[54]]] = T1[T2[BLOCK[57]]]
T1[T2[BLOCK[57]]] = T1[T2[BLOCK[60]]]
T1[T2[BLOCK[60]]] = TMP
```

Changes to the **T2** exchange table:

ALGORITHM 4

```
TMP = T2[T1[BLOCK[32]]]
T2[T1[BLOCK[32]]] = T2[T1[BLOCK[38]]]
T2[T1[BLOCK[38]]] = T2[T1[BLOCK[40]]]
T2[T1[BLOCK[40]]] = T2[T1[BLOCK[46]]]
T2[T1[BLOCK[46]]] = T2[T1[BLOCK[33]]]
T2[T1[BLOCK[33]]] = T2[T1[BLOCK[39]]]
T2[T1[BLOCK[39]]] = T2[T1[BLOCK[41]]]
T2[T1[BLOCK[41]]] = T2[T1[BLOCK[47]]]
T2[T1[BLOCK[47]]] = T2[T1[BLOCK[34]]]
T2[T1[BLOCK[34]]] = T2[T1[BLOCK[36]]]
T2[T1[BLOCK[36]]] = T2[T1[BLOCK[42]]]
T2[T1[BLOCK[42]]] = T2[T1[BLOCK[44]]]
T2[T1[BLOCK[44]]] = T2[T1[BLOCK[35]]]
T2[T1[BLOCK[35]]] = T2[T1[BLOCK[37]]]
T2[T1[BLOCK[37]]] = T2[T1[BLOCK[43]]]
T2[T1[BLOCK[43]]] = T2[T1[BLOCK[45]]]
T2[T1[BLOCK[45]]] = TMP
```

Note that in both cases 16 elements of the swap tables are changed for each round of the mixword() function. The block indexes are fixed but as they are indexed by table **T1** or table **T2** they vary in time according to the contents of the sub blocks **D** and **C**, respectively, for table **T1** and table **T2**.

2.3.5 Mixing the basic words

After generating the words we use bit rotation operations ROTL32 (bit rotation to the left), XOR, ADD and NOT to generate new words. The process will be done as follows:

ALGORITHM 5

```
word[0] = word[0] XOR (ROTL32(NOT(word[1]),13) XOR ROTL32(word[2],3)) + ROTL32(NOT(word[3]),27);
word[1] = word[1] + (ROTL32(word[0],14) XOR ROTL32(NOT(word[2]),11)) + ROTL32(word[3],26);
word[2] = word[2] XOR (ROTL32(NOT(word[0]),9) XOR ROTL32(word[1],20)) + ROTL32(NOT(word[3]),28);
word[3] = word[3] + (ROTL32(word[0],17) XOR ROTL32(NOT(word[1],2)) + ROTL32(word[2],1);

word[0] = word[0] XOR (ROTL32(NOT(word[1]),25) XOR ROTL32(word[2],7)) + ROTL32(NOT(word[3]),18);
word[1] = word[1] + (ROTL32(word[0],10) XOR ROTL32(NOT(word[2]),8)) + ROTL32(word[3],23);
word[2] = word[2] XOR (ROTL32(NOT(word[0]),15) XOR ROTL32(word[1],31)) + ROTL32(NOT(word[3]),29);
word[3] = word[3] + (ROTL32(word[0],30) XOR ROTL32(NOT(word[1]),16)) + ROTL32(word[2],21);

word[0] = word[0] XOR (ROTL32(NOT(word[1]),19) XOR ROTL32(word[2],24)) + ROTL32(NOT(word[3]),12);
word[1] = word[1] + (ROTL32(word[0],22) XOR ROTL32(NOT(word[2]),4)) + ROTL32(word[3],6);
word[2] = word[2] XOR (ROTL32(NOT(word[0]),5) XOR ROTL32(word[1],8)) + ROTL32(NOT(word[3]),13);
word[3] = word[3] + (ROTL32(word[0]),14) XOR ROTL32(NOT(word[1]),24)) + ROTL32(word[2],20);
```

This process causes a diffusion of **sub-block A** with itself. The words are rotated in several bits to the left. In addition, the words are exchanged in the final step preparing for the last step of the round which consists of converting the words again into bytes. These bytes will make up **sub-block D** to be processed by the next round of the mixword() function.

2.3.6 Block rotation

The sub blocks are in positions **A**, **B**, **C** and **D**. After rotation they'll be at positions **B**, **C**, **D** and **A**. See the pseudo code

ALGORITHM 6

```
For ct from 1 to 48 do:
        BLOCK[ct] = BLOCK[ct+16]

position = 0

for ct from 0 to 3 do:
{
        tmp = words[ct]
        tmp1 = tmp DIV 65536
        tmp2 = tmp MOD 65536
        t1 = tmp1 DIV 256
        t2 = tmp1 MOD 256
        t3 = tmp2 DIV 256
        t4 = tmp2 MOD 256

        If (ct MOD 2 == 0)
                BLOCK[48 + position] = T1[(t1+ position) MOD 256]
                BLOCK[49 + position] = T1[(t2+ position+1) MOD 256]
                BLOCK[50 + position] = T1[(t3+ position+2) MOD 256]
                BLOCK[51 + position] = T1[(t4+ position+3) MOD 256]
        if not
                BLOCK[48 + position] = T2[(t1+ position) MOD 256]
                BLOCK[49 + position] = T2[(t2+ position+1) MOD 256]
                BLOCK[50 + position] = T2[(t3+ position+2) MOD 256]
                BLOCK[51 + position] = T2[(t4+ position+3) MOD 256]
        position = position + 4
 }
```

Note that in this case the BLOCK vector represents all 64 bytes which are divided into 4 equal parts representing the sub blocks **A**, **B**, **C** and **D**. The complete mixword() routine is repeated 16 times.

2.3.7 Block_change function()

It is the third and last function to be processed in the main body of the Viktoria hash function. Its purpose is to perform a byte exchange, that is, to reorder the 64 bytes of the block being processed. Thus each processed block can be rearranged in 64! different ways (1.268869322×10^{89}, which corresponds approximately to a 296-bit key). This permutation is dynamic so that it always changes for each block. Thus there is an extra difficulty for the cryptoanalyst to know which permutation is used since this information depends on data present in the whole file. See the pseudo code:

ALGORITHM 7

```
posic=0;

   inicio = (tipo%4)*64;
   fim = inicio + 64;
   if (tipo < 4){
      for(ct=0;ct<256;ct++){
           if (T2[ct] >= inicio & T2[ct] < fim){
                 BLOCK_TMP[posic] = BLOCK[T2[ct]%64];
                 posic++;
            if (posic > 63){
               break;
         }
        }
   }
   } else {
      for(ct=0;ct<256;ct++){
           if (T1[ct] >= inicio & T1[ct] < fim){
                 BLOCK_TMP[posic] = BLOCK[T1[ct]%64];
                 posic++;
           if (posic > 63){
           break;
           }
      }
   }
}

for (ct=0;ct<64;ct++){
           BLOCK[ct] = BLOCK_TMP[ct];
 }
```

The first processing loop permutes the data block of the file according to the swap tables **T2** and **T1** (pivot tables). The second loop of this routine only transfers the data from the temporary vector to the final BLOCK that will be used together with the next block to be processed. The block size is always 512 bits.

2.3.8 Mixword_final() function

This routine is very similar to the mixword() function except that it is executed at least 8192 times and at most 16382 times per file. It is the penultimate operation to be performed before ending with hash output. More details can be found in Appendix IV.

2.3.9 End Function()

It is executed only once for each file processed. This routine performs an XOR operation after the last block swap operation in the last block of the file. It consists of doing a multiplication operation between bytes of the swapping tables **T1** and **T2**. From the result of this operation we extract one byte necessary for the final XOR operation.

See the pseudo code:

ALGORITHM 8

```
position=0

For ct from 1 to 64 do:
           tmp1 = (T1[posicao] * 256) + T2[posicao];
           tmp2 = (T2[posicao+64] * 256) + T1[posicao+64];
```

```
If tmp1 == 0
        tmp1 = 65536

If tmp2 == 0
        tmp2 = 65536

result = (tmp1 * tmp2) MOD 65537

BLOCK[ct]= BLOCK[ct] XOR (result MOD 256)
 position = position + 2
```

This routine aims to make the hash analysis more difficult, protecting from a possible attack that aims to undo the last byte exchange operation. The number of permutation possibilities is 64!, and in this multiplication we have $(256! / 128!)^2 = 4{,}948458079 \times 10^{582}$ which is approximately equivalent to a 1935 bit key. The number of combinations is very large but in practice it is limited to a 512-bit XOR operation which injects an uncertainty as to the content of the **T1** and **T2** interchange tables and the order in which the bytes were exchanged in the previous operation.

3. DESIGN JUSTIFICATION

The Viktoria algorithm has an elegant and efficient design. The mechanism starts differentiating messages by their size through a header and a null byte control block. Only in this step that is part of the pre-processing and in the initialization of the exchange tables T1 and T2 the algorithm already promotes positive disagreements between similar messages. Regarding the central processing of the algorithm we have a dynamic block reading where the information read from each message block is processed by a different dynamic SBOX. The processing of the mixword() function is very efficient, requiring in general only 4 of the 16 runs performed to promote non-compressiveness and randomness in the data. And the byte-switching function is also very efficient being performed dynamically for each block, always doing a different permutation. The following tables illustrate this mechanism:

Word 0															
SUB-BLOCK A				**SUB-BLOCK B**				**SUB-BLOCK C**				**SUB-BLOCK D**			
Current				**T0 (without SBOX)**				**T1**				**T2**			
$P0_A$	$P1_A$	$P2_A$	$P3_A$	**16**	17	18	19	**32**	**33**	**34**	**35**	**48**	49	50	51
$P0_B$	$P1_B$	$P2_B$	$P3_B$	20	**21**	22	23	**36**	**37**	**38**	**39**	52	53	54	**55**
$P0_C$	$P1_C$	$P2_C$	$P3_C$	24	25	**26**	27	**40**	**41**	**42**	**43**	56	57	**58**	59
$P0_D$	$P1_D$	$P2_D$	$P3_D$	28	29	30	**31**	**44**	**45**	**46**	**47**	60	**61**	62	63

Table 6

Word 1															
SUB-BLOCK A				**SUB-BLOCK B**				**SUB-BLOCK C**				**SUB-BLOCK D**			
Current				**T0 (without SBOX)**				**T1**				**T2**			
$P0_A$	**$P1_A$**	$P2_A$	$P3_A$	16	**17**	18	19	32	**33**	34	35	48	**49**	50	51
$P0_B$	**$P1_B$**	$P2_B$	$P3_B$	20	21	**22**	23	36	37	38	**39**	**52**	53	54	55
$P0_C$	**$P1_C$**	$P2_C$	$P3_C$	24	25	26	**27**	40	**41**	42	43	56	57	58	**59**
$P0_D$	**$P1_D$**	$P2_D$	$P3_D$	**28**	29	30	31	44	45	46	**47**	60	61	**62**	63

Table 7

Word 2															
SUB-BLOCK A				**SUB-BLOCK B**				**SUB-BLOCK C**				**SUB-BLOCK D**			
Current				**T0 (without SBOX)**				**T1**				**T2**			
$P0_A$	$P1_A$	**$P2_A$**	$P3_A$	16	17	**18**	19	32	33	**34**	35	48	49	**50**	51
$P0_B$	$P1_B$	**$P2_B$**	$P3_B$	20	21	22	**23**	**36**	37	38	39	52	**53**	54	55
$P0_C$	$P1_C$	**$P2_C$**	$P3_C$	**24**	25	26	27	40	41	**42**	43	**56**	57	58	59
$P0_D$	$P1_D$	**$P2_D$**	$P3_D$	28	**29**	30	31	**44**	45	46	47	60	61	62	**63**

Table 8

Word 3															
SUB-BLOCK A				**SUB-BLOCK B**				**SUB-BLOCK C**				**SUB-BLOCK D**			
Current				**T0 (without SBOX)**				**T1**				**T2**			
$P0_A$	$P1_A$	$P2_A$	**$P3_A$**	16	17	18	**19**	32	33	34	**35**	48	49	50	**51**
$P0_B$	$P1_B$	$P2_B$	**$P3_B$**	**20**	21	22	23	36	**37**	38	39	52	53	**54**	55
$P0_C$	$P1_C$	$P2_C$	**$P3_C$**	24	**25**	26	27	40	41	42	**43**	56	**57**	58	59
$P0_D$	$P1_D$	$P2_D$	**$P3_D$**	28	29	**30**	31	44	**45**	46	47	**60**	61	62	63

Table 9

Block rotation															
SUB-BLOCK B				**SUB-BLOCK C**				**SUB-BLOCK D**				**SUB-BLOCK A**			
Current				**T0 (without SBOX)**				**T1**				**T2**			
16	17	18	19	32	33	34	35	48	49	50	51	$P1_A$	$P1_B$	$P1_C$	$P1_D$
20	21	22	23	36	37	38	39	52	53	54	55	$P2_A$	$P2_B$	$P2_C$	$P2_D$
24	25	26	27	40	41	42	43	56	57	58	59	$P3_A$	$P3_B$	$P3_C$	$P3_D$
28	29	30	31	44	45	46	47	60	61	62	63	**$P0_A$**	**$P0_B$**	**$P0_C$**	**$P0_D$**

Table 10

In the main algorithm we use three sub-blocks to change the first block. An important observation in the design of the rotation function of sub-blocks is that it transforms a word that is initially represented by a **column of sub-block A** into a **row of sub-block D**. This is very useful because in the next round of the function **sub-block B** will be changed by elements of the 4 columns of **sub-block A** that originate from elements of the four words of sub-blocks **A**, **B**, **C** and **D**.

Round 1															
SUB-BLOCK A				**SUB-BLOCK B**				**SUB-BLOCK C**				**SUB-BLOCK D**			
Current				**T0 (without SBOX)**				**T1**				**T2**			
$P0_A$	**$P1_A$**	**$P2_A$**	**$P3_A$**	**16**	17	18	19	**32**	33	34	35	**48**	49	50	51
$P0_B$	**$P1_B$**	**$P2_B$**	**$P3_B$**	20	**21**	22	23	36	37	**38**	39	52	53	54	**55**
$P0_C$	**$P1_C$**	**$P2_C$**	**$P3_C$**	24	25	**26**	27	**40**	41	42	43	56	57	**58**	59
$P0_D$	**$P1_D$**	**$P2_D$**	**$P3_D$**	28	29	30	**31**	44	45	**46**	47	60	**61**	62	63

Table 11

Note in table 12 the shift to the left of the sub-blocks with respect to table 11. This movement allows you to set the new word **P0** to:

P0 = ((byte nº20) * 2^{24}) + ((byte nº 24) * 2^{16}) + ((byte nº 28) * 2^{8})+ (byte nº 16)

Round 2															
SUB-BLOCK B				**SUB-BLOCK C**				**SUB-BLOCK D**				**SUB-BLOCK A**			
Current				**T0 (without SBOX)**				**T1**				**T2**			
16	17	18	19	**32**	33	34	35	**48**	49	50	51	**$P1_A$**	$P1_B$	$P1_C$	$P1_D$
20	21	22	23	36	**37**	38	39	52	53	**54**	55	$P2_A$	$P2_B$	$P2_C$	**$P2_D$**
24	25	26	27	40	41	**42**	43	**56**	57	58	59	$P3_A$	$P3_B$	**$P3_C$**	$P3_D$
28	29	30	31	44	45	46	**47**	60	61	**62**	63	$P0_A$	**$P0_B$**	$P0_C$	$P0_D$

Table 12

The word **P0** will be modified by the words highlighted in sub-blocks **C**, **D** and **A** (see table 12). Sub-blocks **B**, **C** and **D** were not changed in the first round of the mixword() function. Note carefully the bytes 32 and 48. In the first round of the mixword() function they interacted with block **A** through functions T1[32] and T2[48] (in fact it's just the Sbox's). In the second round the interaction is T0[32] and T1[48]. Remember that T0 represents the raw byte (without any change by the Sbox). This way the same data provides different changes to each round of the mixword() function. This feature provides a true pseudo-random number generator taking into account that every round the swap tables are changed and consequently the Sbox's change as well. After the 16th round of the mixword() function is executed the permutation_block() function which performs a permutation of the 64 bytes of the block being processed. Then a further 64 bytes of the message is read out and an XOR operation is performed with the previous block, repeating the processing cycle in the new block.

Regarding the two post-processing functions (mixword_final and final) we can say that they only provide an additional barrier to make statistical attacks as difficult as possible (such as differential and linear cryptoanalysis) besides guaranteeing a minimum number of execution rounds of the algorithm's main structure.

Viktoria hash can also generate other hash sizes with 1024 or 2048 bits thus ensuring versatility of the algorithm. The hashes with larger sizes are generated through the concatenation of 512-bit hashes generated through algorithm 9. While the 512-bit hash is secure the others are also secure because they are built through post-processing of the data used to generate the basic 512-bit hash.

ALGORITHM 9

```
permutation_binary_512()
mixword_final()
permutation_block()
mixword_final()
finalizes()
```

Each time Algorithm 9 is run it produces a 512 bit output that is joined to the previous hash to form the final hash value.

4. LOGICAL BASIS

Viktoria hash has in its main core 3 processes that perform different functions: read_block(), mixword() and permutation_block().

4.1 Read_block() function

The read_block() function reads 64-byte blocks of the message as a function of the T1 interchange box, which in this case functions as a SBOX interchange box. However this substitution box is dynamic and changes every 64 bytes read from the message. This makes it much more difficult to trace the content of the message (especially when it is very long) because it is being "encrypted" by a mechanism similar to a polyalphabetic substitution whose key changes every 64 bytes read. It is worth noting that the mixword() function is executed 16 times and changes the T1 table every round (depending on the block data). Another important fact is that the initial state of the swap tables T1 and T2 depend on the content of the entire message and are processed before the data blocks of the message are read.

For example, for a message with 256 concatenated letters "A" we have the readings of the 4 blocks from the following substitution boxes (the asterisks after the numbers indicate that the byte was substituted in relation to the previous table):

```
 33  226   57  123  220   27   53   70    5   62  253    4    1  234   69   86
119  189  186  255  130  251  201  144  245  221  192  247  116  225   47   89
230  198  154  229  199  151  193   56  248  113   99   90   28   59  132   55
101  191  222   38  108  135  178  160  140  242  244   72   96   95   98   19
163  215    8   18   20  104  169  194   46   48   87   15  219  158   79   97
173  121  161  124   16  159   81  153  227   65  127  218   41  109  200  202
 42  128   39  111    3  204  134  213  211  187  205   25   94   17   14   31
 76   50   45   21  110   10   40  118  170  181  195  236  184  217   29   63
228  155  162  252   64   26   93    0   92  103   34  210   58   83  138   75
 43    2  207   24   11   32  254  164  180   68  208  249  157  147  139  156
 49   78  175  235  141  250  243    7  209  166   73  148  185  136  149  146
188   44   51  203  183  100   77  172  223   13  129  206  176   80  196  114
241  106  167   85   82  117  212  112   61  102   88  190  115  246   30  233
122  240   91   37   71   60  126    9   35  152  125  131  143  133  174   84
224   67   52  177  239  238  197  232  182    6  105  214  137   23  231  237
 36  107  179   22   12  150  142  120   54  171   74  168  145  216   66  165
```

SBOX for block 1

```
126* 211*  57  123   96*  27   53  199*   5  243* 134*  35*   1  179* 215*  86
119  143* 101* 255  130  251  242*  13* 245  217* 192  237*  83* 225  177*  89
230  198   24* 154* 120* 138* 144*  26* 248  158*  99   52* 156*  66* 115*  32*
 29*  17* 222   38  108   56* 178  160    3* 252* 244   72  240*  95   98   19
 21*  34* 139* 116*  20  213* 104* 194  145* 212* 100*  15  167* 197*  79   97
173  111* 239*  16* 232* 159  141* 153   64*  93* 190* 186* 164* 189*  22* 202
 42   36*  40* 226*  73*  78* 140*  18* 231* 161* 205   25  254*  28* 181*  31
 88*  58*  45  235* 182* 253*  62* 118   48* 112* 195   85* 184  207* 127* 249*
121* 187*  39* 172*  76* 201*  30*   0   92  228* 214* 210  155*  63*   8*  75
105*   2  247* 128*  11  219* 147* 216* 148* 234* 208   37* 157  169*  59*   4*
 49   94* 175  149* 250* 224* 152*   7   91* 233* 103*  55* 209* 136   10* 146
188   44   51  203  183  221*  77  180* 223  200* 185* 206  176   46* 196  114
241   33* 129* 238*  82  117  113*  43*  70* 102   87* 171* 168* 236*  41* 163*
122   69* 151* 227*  71  170* 191*   9   60*  61* 125  131  193* 133  174   84
 81*  67  132* 229*  68*  12* 110* 246* 220*   6  124*  14* 137   23  166* 204*
 80* 107  109* 218* 162* 150  142  106*  54   90*  47* 135*  74*  50*  65* 165
```

SBOX for block 2

```
 92* 211  182* 123  206*  96*  53  119*   5   50* 134   64*   1  179  215   86
233* 143  101  255  149* 169* 242   38*  93* 150* 191* 237   83   87* 177   46*
126* 145*  24  254* 120  105* 144   26   48* 158   99  214*  69* 125* 115   32
  2* 113* 222  154* 202* 198*  33* 104*   3  234* 244   72  240   58* 224* 188*
160*   6* 176*  63* 221*  89* 114* 102* 132* 136* 253* 146*   7* 197  170* 243*
173   62* 239   16  232  159  246* 153  128*  35* 190  111*  57* 192* 187* 167*
 42  196*  40  226   73  217* 140  174* 161* 227* 129*  25  130* 156*  29*  31
 71* 164*  45  235   28*  36* 223*  20* 155*  75* 195   85  184  207  162* 137*
121  189*  59* 108* 142* 201   91*  18*  10* 228   66* 210   17*  70*   8  185*
251* 208* 247   98*  11  168* 147  252* 110*   0*  97*  37  157  181* 152*   4
 41* 248*  34* 249* 250   21*  65* 245*  61*  74* 103  139* 118*  78*  95* 141*
172*  94*  51  203  183   79*  77  180  127*  27* 219* 218*  76* 186* 225* 109*
231* 117* 112* 238   82  163* 213*  43  200*  55* 194* 171   12* 236   56* 106*
122   15*  22* 212*  88*  44* 199*   9   60  138* 151* 131  193  133   13*  84
 81   67  100* 229   68  148* 178*  30* 220  209* 124   14   80*  23  166  204
 52* 107   19* 230* 175* 205* 241* 116*  54   90   47  135   39* 216*  49* 165
```

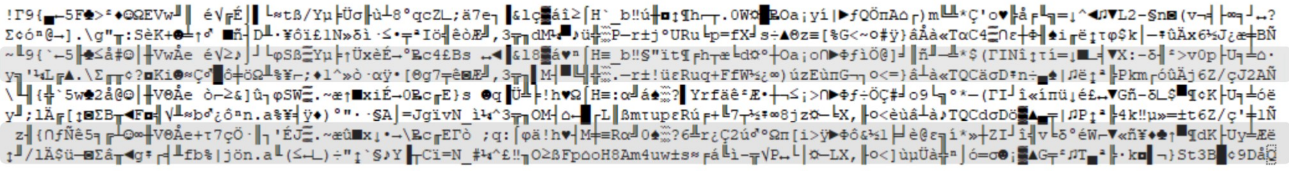
SBOX for block 3

```
255* 122* 182  123  239* 159* 165* 136*  53* 184* 201* 193*   1  236* 215   86
233  143  101   43* 231*  55* 135* 153* 250* 186* 191   39* 144*  74* 177   46
126  145  150* 254  120   25*   7*  26   92* 158   99  214   69  226* 149*  32
 59* 113   58* 244* 237* 132*  33  104    3  180*  77* 216* 240   82* 224  188
 48*   6  176   63   54* 202* 114  168* 128*  50* 163*  11* 167* 234* 227*  91*
105*  62  152*  16  232  147*  38* 171*  49* 198* 190  138*  64* 238* 187  161*
 42  175* 197*  90*  73  217  140  185* 118* 212* 235* 248* 130   87* 169*  31
174* 164  157*   4*   5*  24* 223   20  100*  75  195   85  121* 207  146* 137
 18* 189   47* 108  142   36* 129* 196*  10  228  131* 210   17  103*  12* 218*
181* 208  102*  98   37* 124* 106* 148* 110    0   97  211*  40* 243*  27*  28*
 41  246*  34   23*  96*  21   13*  89* 222* 194*  67* 139   61*  78   95   35*
172   94  156*  19* 183   79  242* 225*  70* 112* 127* 111*  72*  56*  65* 109
 52* 117  119* 241* 115* 247* 213  160* 200  141*  45* 203* 251*  80*  29* 192*
179*  15   22   76*  88   44  199    9   60   93* 151  230* 154* 133  206* 252*
245* 162* 205* 229    2* 173* 178   30   71* 209  253*  14   84* 220* 166  204
249* 107    8* 221* 170* 125*  83* 116   51*  66* 219* 155*  57*  68* 134*  81*
```

SBOX for block 4

This way the read_block() function does not work exactly like a pseudo-random number generator but it helps to modify each message block with a different table. The period of the function is undetermined because it depends on the content of each read block. Since the swap tables T1 and T2 are changed every round of the mixword() function and they interact with each other we can only note here their maximum period of 256!^{2}. About 5/8 of the T1 values are changed every time the mixword() function is passed.

The 4 sbox's previously seen can be represented visually as follows:



Picture 1

4.2 Mixword function()

Here we have some notes on word formation in the mixword() function. The vector **B** represents the 64 bytes read block of the message. Vectors T1 and T2 are the interchange tables that in this case work as Sbox's:

ALGORITHM 10

```
Word 0 =
         (   B[ 0]  * 256^3) + (   B[ 4]  * 256^2) + (   B[ 8]  * 256) + (   B[12] )
     xor (   B[16]  * 256^3) + (   B[21]  * 256^2) + (   B[26]  * 256) + (   B[31] )
     add (T1[B[32]] * 256^3) + (T1[B[38]] * 256^2) + (T1[B[40]] * 256) + (T1[B[46]])
     xor (T2[B[48]] * 256^3) + (T2[B[55]] * 256^2) + (T2[B[58]] * 256) + (T2[B[61]])

Word 1 =
         (   B[ 1]  * 256^3) + (   B[ 5]  * 256^2) + (   B[ 9]  * 256) + (   B[13] )
     xor (   B[17]  * 256^3) + (   B[22]  * 256^2) + (   B[27]  * 256) + (   B[28] )
     add (T1[B[33]] * 256^3) + (T1[B[39]] * 256^2) + (T1[B[41]] * 256) + (T1[B[47]])
     xor (T2[B[49]] * 256^3) + (T2[B[52]] * 256^2) + (T2[B[59]] * 256) + (T2[B[62]])

Word 2 =
         (   B[ 2]  * 256^3) + (   B[ 6]  * 256^2) + (   B[10]  * 256) + (   B[14] )
     xor (   B[18]  * 256^3) + (   B[23]  * 256^2) + (   B[24]  * 256) + (   B[29] )
     add (T1[B[34]] * 256^3) + (T1[B[36]] * 256^2) + (T1[B[42]] * 256) + (T1[B[44]])
     xor (T2[B[50]] * 256^3) + (T2[B[53]] * 256^2) + (T2[B[56]] * 256) + (T2[B[63]])

Word 3 =
         (   B[ 3]  * 256^3) + (   B[ 7]  * 256^2) + (   B[11]  * 256) + (   B[15] )
     xor (   B[19]  * 256^3) + (   B[20]  * 256^2) + (   B[25]  * 256) + (   B[30] )
     add (T1[B[35]] * 256^3) + (T1[B[37]] * 256^2) + (T1[B[43]] * 256) + (T1[B[45]])
     xor (T2[B[51]] * 256^3) + (T2[B[54]] * 256^2) + (T2[B[57]] * 256) + (T2[B[60]])
```

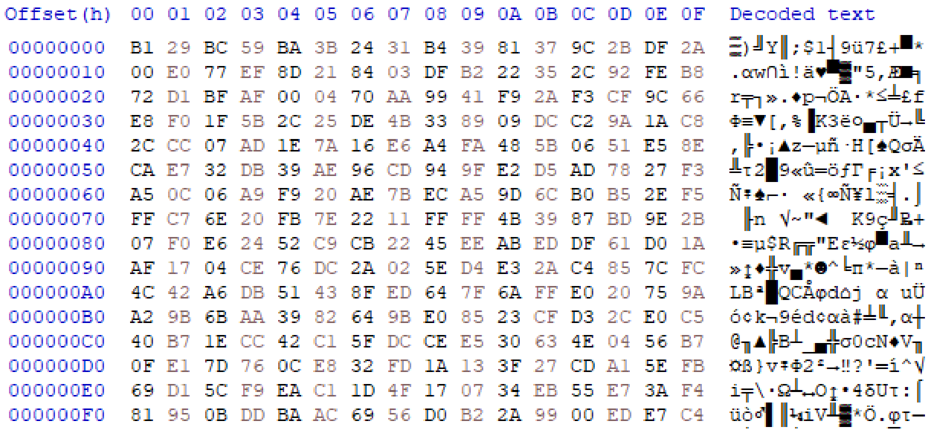
Here we have the initial formations of the 4 words that represent the beginning of the processing of the mixword() function. Each word contains 128-bit information from the message block and the 4 words together condense information from the entire block. We see for example that sub-block A when receiving information from sub-blocks B, C and D retain information from the entire 512-bit block. As this operation is reversible there are no collisions and the process guarantees that given the three sub-blocks **B**, **C** and **D**, there is only one corresponding sub-block **A** with the interchange tables **T1** and **T2** in the same states.

In the next phase of the mixword() function the words interact with each other using the XOR, ADD (2^{32} module sum), NOT and ROTL32 (left bit rotation) operations. See details in algorithm 5. In table 13 we see the result of the transformations made by this code. The input is composed by 4 words of 32 bits and the output also produces 4 words of 32 bits:

ENTRY				OUT			
Word 0	**Word 1**	**Word 2**	**Word 3**	**Word 0**	**Word 1**	**Word 2**	**Word 3**
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	B1 29 BC 59	BA 3B 24 31	B4 39 81 37	9C 2B DF 2A
00 00 00 01	00 00 00 00	00 00 00 00	00 00 00 00	00 E0 77 EF	8D 21 84 03	DF B2 22 35	2C 92 FE B8
00 00 00 02	00 00 00 00	00 00 00 00	00 00 00 00	72 D1 BF AF	00 04 70 AA	99 41 F9 2A	F3 CF 9C 66
00 00 00 03	00 00 00 00	00 00 00 00	00 00 00 00	E8 F0 1F 5B	2C 25 DE 4B	33 89 09 DC	C2 9A 1A C8
00 00 00 04	00 00 00 00	00 00 00 00	00 00 00 00	2C CC 07 AD	1E 7A 16 E6	A4 FA 48 5B	06 51 E5 8E
00 00 00 05	00 00 00 00	00 00 00 00	00 00 00 00	CA E7 32 DB	39 AE 96 CD	94 9F E2 D5	AD 78 27 F3
00 00 00 06	00 00 00 00	00 00 00 00	00 00 00 00	A5 0C 06 A9	F9 20 AE 7B	EC A5 9D 6C	B0 B5 2E F5
00 00 00 07	00 00 00 00	00 00 00 00	00 00 00 00	FF C7 6E 20	FB 7E 22 11	FF FF 4B 39	87 BD 9E 2B
00 00 00 08	00 00 00 00	00 00 00 00	00 00 00 00	07 F0 E6 24	52 C9 CB 22	45 EE AB ED	DF 61 D0 1A
00 00 00 09	00 00 00 00	00 00 00 00	00 00 00 00	AF 17 04 CE	76 DC 2A 02	5E D4 E3 2A	C4 85 7C FC
00 00 00 0A	00 00 00 00	00 00 00 00	00 00 00 00	4C 42 A6 DB	51 43 8F ED	64 7F 6A FF	E0 20 75 9A
00 00 00 0B	00 00 00 00	00 00 00 00	00 00 00 00	A2 9B 6B AA	39 82 64 9B	E0 85 23 CF	D3 2C E0 C5
00 00 00 0C	00 00 00 00	00 00 00 00	00 00 00 00	40 B7 1E CC	42 C1 5F DC	CE E5 30 63	4E 04 56 B7
00 00 00 0D	00 00 00 00	00 00 00 00	00 00 00 00	0F E1 7D 76	0C E8 32 FD	1A 13 3F 27	CD A1 5E FB
00 00 00 0E	00 00 00 00	00 00 00 00	00 00 00 00	69 D1 5C F9	EA C1 1D 4F	17 07 34 EB	55 E7 3A F4
00 00 00 0F	00 00 00 00	00 00 00 00	00 00 00 00	81 95 0B DD	BA AC 69 56	D0 B2 2A 99	00 ED E7 C4

Table 13

We can graphically represent the entries and exits in this way:



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  B1 29 BC 59 BA 3B 24 31 B4 39 81 37 9C 2B DF 2A  ▒)╝Y║;$1┤9ü7£+▀*
00000010  00 E0 77 EF 8D 21 84 03 DF B2 22 35 2C 92 FE B8  .αw∩ì!ä♥▀▓"5,Æ■╕
00000020  72 D1 BF AF 00 04 70 AA 99 41 F9 2A F3 CF 9C 66  r╤┐».♦p¬ÖA·*≤╧£f
00000030  E8 F0 1F 5B 2C 25 DE 4B 33 89 09 DC C2 9A 1A C8  Φ≡▼[,%▐K3ë○▄┬Ü→╚
00000040  2C CC 07 AD 1E 7A 16 E6 A4 FA 48 5B 06 51 E5 8E  ,╠•¡▲z▬µñ·H[♠QσÄ
00000050  CA E7 32 DB 39 AE 96 CD 94 9F E2 D5 AD 78 27 F3  ╩τ2█9«û═öƒΓ╒¡x'≤
00000060  A5 0C 06 A9 F9 20 AE 7B EC A5 9D 6C B0 B5 2E F5  Ñ♀♠⌐· «{∞Ñ¥l░╡.⌡
00000070  FF C7 6E 20 FB 7E 22 11 FF FF 4B 39 87 BD 9E 2B   ╟n √~"◄  K9ç╜₧+
00000080  07 F0 E6 24 52 C9 CB 22 45 EE AB ED DF 61 D0 1A  •≡µ$R╔╦"Eε½φ▀a╨→
00000090  AF 17 04 CE 76 DC 2A 02 5E D4 E3 2A C4 85 7C FC  »↨♦╬v▄*☻^╘π*─à|ⁿ
000000A0  4C 42 A6 DB 51 43 8F ED 64 7F 6A FF E0 20 75 9A  LBª█QCÅφd⌂j α uÜ
000000B0  A2 9B 6B AA 39 82 64 9B E0 85 23 CF D3 2C E0 C5  ó¢k¬9éd¢αà#╧╙,α┼
000000C0  40 B7 1E CC 42 C1 5F DC CE E5 30 63 4E 04 56 B7  @╖▲╠B┴_▄╬σ0cN♦V╖
000000D0  0F E1 7D 76 0C E8 32 FD 1A 13 3F 27 CD A1 5E FB  ☼ß}v♀Φ2²→‼?'═í^√
000000E0  69 D1 5C F9 EA C1 1D 4F 17 07 34 EB 55 E7 3A F4  i╤\·Ω┴↔O↨•4δUτ:⌠
000000F0  81 95 0B DD BA AC 69 56 D0 B2 2A 99 00 ED E7 C4  üò♂▌║¼iV╨▓*Ö.φτ─
```

Picture 2

Using the Diehard3 test battery (see test description in Annex V) we obtained a positive result in terms of randomness by testing 12 megabytes of data of the function represented in figure 2:

```
BIRTHDAY SPACINGS TEST, M= 512 N=2**24 LAMBDA=  2.0000
          saida.bin       using bits  1 to 24 p-value=  .944304
          saida.bin       using bits  2 to 25 p-value=  .288732
          saida.bin       using bits  3 to 26 p-value=  .138885
          saida.bin       using bits  4 to 27 p-value=  .376896
          saida.bin       using bits  5 to 28 p-value=  .629163
          saida.bin       using bits  6 to 29 p-value=  .920090
          saida.bin       using bits  7 to 30 p-value=  .372930
          saida.bin       using bits  8 to 31 p-value=  .099840
          saida.bin       using bits  9 to 32 p-value=  .223834
  The 9 p-values were
       .944304   .288732   .138885   .376896   .629163
       .920090   .372930   .099840   .223834
 A KSTEST for the 9 p-values yields  .339095
--------------------------------------------------------------------------------
          OPERM5 test for file saida.bin
chisquare for 99 degrees of freedom= 87.308; p-value= .206513
          OPERM5 test for file saida.bin
chisquare for 99 degrees of freedom= 53.514; p-value= .000055
--------------------------------------------------------------------------------
   Binary rank test for saida.bin
        Rank test for 31x31 binary matrices:
       rows from leftmost 31 bits of each 32-bit integer
     rank   observed  expected  (o-e)^2/e   sum
       28       238     211.4  3.342203    3.342
       29      5073    5134.0   .725018    4.067
       30     23218   23103.0   .571969    4.639
       31     11471   11551.5   .561327    5.201
 chisquare= 5.201 for 3 d. of f.; p-value= .853601
   Binary rank test for saida.bin
        Rank test for 32x32 binary matrices:
       rows from leftmost 32 bits of each 32-bit integer
     rank   observed  expected  (o-e)^2/e   sum
       29       188     211.4  2.593929    2.594
       30      5227    5134.0  1.684276    4.278
       31     23181   23103.0   .263025    4.541
       32     11404   11551.5  1.884033    6.425
 chisquare= 6.425 for 3 d. of f.; p-value= .912969
--------------------------------------------------------------------------------
b-rank test for bits  1 to  8 p=1-exp(-SUM/2)= .83751
b-rank test for bits  2 to  9 p=1-exp(-SUM/2)= .85710
b-rank test for bits  3 to 10 p=1-exp(-SUM/2)= .89791
b-rank test for bits  4 to 11 p=1-exp(-SUM/2)= .36146
b-rank test for bits  5 to 12 p=1-exp(-SUM/2)= .38158
b-rank test for bits  6 to 13 p=1-exp(-SUM/2)= .73410
b-rank test for bits  7 to 14 p=1-exp(-SUM/2)= .20633
b-rank test for bits  8 to 15 p=1-exp(-SUM/2)= .96461
b-rank test for bits  9 to 16 p=1-exp(-SUM/2)= .99192
b-rank test for bits 10 to 17 p=1-exp(-SUM/2)= .04526
b-rank test for bits 11 to 18 p=1-exp(-SUM/2)= .04721
b-rank test for bits 12 to 19 p=1-exp(-SUM/2)= .20809
b-rank test for bits 13 to 20 p=1-exp(-SUM/2)= .10942
b-rank test for bits 14 to 21 p=1-exp(-SUM/2)= .82995
b-rank test for bits 15 to 22 p=1-exp(-SUM/2)= .11413
b-rank test for bits 16 to 23 p=1-exp(-SUM/2)= .58813
b-rank test for bits 17 to 24 p=1-exp(-SUM/2)= .69165
b-rank test for bits 18 to 25 p=1-exp(-SUM/2)= .86054
b-rank test for bits 19 to 26 p=1-exp(-SUM/2)= .33184
```

```
 b-rank test for bits 20 to 27 p=1-exp(-SUM/2)= .09620
 b-rank test for bits 21 to 28 p=1-exp(-SUM/2)= .86821
 b-rank test for bits 22 to 29 p=1-exp(-SUM/2)= .13135
 b-rank test for bits 23 to 30 p=1-exp(-SUM/2)= .19439
 b-rank test for bits 24 to 31 p=1-exp(-SUM/2)= .93140
 b-rank test for bits 25 to 32 p=1-exp(-SUM/2)= .89434
   TEST SUMMARY, 25 tests on 100,000 random 6x8 matrices
 These should be 25 uniform [0,1] random variables:
     .837512     .857097     .897906     .361459     .381575
     .734100     .206330     .964606     .991925     .045263
     .047208     .208091     .109424     .829950     .114130
     .588132     .691647     .860544     .331844     .096200
     .868210     .131347     .194395     .931403     .894339
   brank test summary for saida.bin
       The KS test for those 25 supposed UNI's yields
                    KS p-value= .820499
--------------------------------------------------------------------------------
  No. missing words should average  141909. with sigma=428.
 tst no  1:  142413 missing words,    1.18 sigmas from mean, p-value= .88036
 tst no  2:  142632 missing words,    1.69 sigmas from mean, p-value= .95434
 tst no  3:  141693 missing words,    -.51 sigmas from mean, p-value= .30663
 tst no  4:  142108 missing words,     .46 sigmas from mean, p-value= .67874
 tst no  5:  142941 missing words,    2.41 sigmas from mean, p-value= .99203
 tst no  6:  141816 missing words,    -.22 sigmas from mean, p-value= .41369
 tst no  7:  142271 missing words,     .85 sigmas from mean, p-value= .80095
 tst no  8:  141305 missing words,   -1.41 sigmas from mean, p-value= .07898
 tst no  9:  141995 missing words,     .20 sigmas from mean, p-value= .57933
 tst no 10:  141158 missing words,   -1.76 sigmas from mean, p-value= .03959
 tst no 11:  142668 missing words,    1.77 sigmas from mean, p-value= .96185
 tst no 12:  141586 missing words,    -.76 sigmas from mean, p-value= .22499
 tst no 13:  141437 missing words,   -1.10 sigmas from mean, p-value= .13489
 tst no 14:  141689 missing words,    -.51 sigmas from mean, p-value= .30335
 tst no 15:  142280 missing words,     .87 sigmas from mean, p-value= .80677
 tst no 16:  141039 missing words,   -2.03 sigmas from mean, p-value= .02100
 tst no 17:  142366 missing words,    1.07 sigmas from mean, p-value= .85701
 tst no 18:  141718 missing words,    -.45 sigmas from mean, p-value= .32743
 tst no 19:  141867 missing words,    -.10 sigmas from mean, p-value= .46061
 tst no 20:  141832 missing words,    -.18 sigmas from mean, p-value= .42831
--------------------------------------------------------------------------------
    OPSO for saida.bin       using bits 23 to 32        141999   .309  .6214
    OPSO for saida.bin       using bits 22 to 31        142241  1.144  .8736
    OPSO for saida.bin       using bits 21 to 30        141755  -.532  .2973
    OPSO for saida.bin       using bits 20 to 29        141318 -2.039  .0207
    OPSO for saida.bin       using bits 19 to 28        141790  -.411  .3404
    OPSO for saida.bin       using bits 18 to 27        142213  1.047  .8525
    OPSO for saida.bin       using bits 17 to 26        142011   .351  .6371
    OPSO for saida.bin       using bits 16 to 25        142394  1.671  .9527
    OPSO for saida.bin       using bits 15 to 24        142054   .499  .6911
    OPSO for saida.bin       using bits 14 to 23        141475 -1.498  .0671
    OPSO for saida.bin       using bits 13 to 22        141948   .133  .5530
    OPSO for saida.bin       using bits 12 to 21        142219  1.068  .8572
    OPSO for saida.bin       using bits 11 to 20        141704  -.708  .2395
    OPSO for saida.bin       using bits 10 to 19        141819  -.311  .3777
    OPSO for saida.bin       using bits  9 to 18        141996   .299  .6175
    OPSO for saida.bin       using bits  8 to 17        141772  -.474  .3179
    OPSO for saida.bin       using bits  7 to 16        141820  -.308  .3790
    OPSO for saida.bin       using bits  6 to 15        141682  -.784  .2166
    OPSO for saida.bin       using bits  5 to 14        142302  1.354  .9121
    OPSO for saida.bin       using bits  4 to 13        141880  -.101  .4597
    OPSO for saida.bin       using bits  3 to 12        141797  -.387  .3493
    OPSO for saida.bin       using bits  2 to 11        141798  -.384  .3505
    OPSO for saida.bin       using bits  1 to 10        142223  1.082  .8603
    OQSO for saida.bin       using bits 28 to 32        142129   .745  .7718
    OQSO for saida.bin       using bits 27 to 31        142233  1.097  .8637
    OQSO for saida.bin       using bits 26 to 30        142185   .934  .8250
    OQSO for saida.bin       using bits 25 to 29        141998   .301  .6181
    OQSO for saida.bin       using bits 24 to 28        141896  -.045  .4820
    OQSO for saida.bin       using bits 23 to 27        142335  1.443  .9255
    OQSO for saida.bin       using bits 22 to 26        142174   .897  .8152
    OQSO for saida.bin       using bits 21 to 25        142324  1.406  .9201
    OQSO for saida.bin       using bits 20 to 24        141481 -1.452  .0733
    OQSO for saida.bin       using bits 19 to 23        142004   .321  .6259
    OQSO for saida.bin       using bits 18 to 22        141435 -1.608  .0539
    OQSO for saida.bin       using bits 17 to 21        141723  -.632  .2638
    OQSO for saida.bin       using bits 16 to 20        141654  -.866  .1934
    OQSO for saida.bin       using bits 15 to 19        141393 -1.750  .0400
    OQSO for saida.bin       using bits 14 to 18        141821  -.299  .3823
    OQSO for saida.bin       using bits 13 to 17        141663  -.835  .2019
    OQSO for saida.bin       using bits 12 to 16        141933   .080  .5320
    OQSO for saida.bin       using bits 11 to 15        142344  1.473  .9297
    OQSO for saida.bin       using bits 10 to 14        142085   .595  .7242
    OQSO for saida.bin       using bits  9 to 13        141916   .023  .5090
    OQSO for saida.bin       using bits  8 to 12        141971   .209  .5828
    OQSO for saida.bin       using bits  7 to 11        141991   .277  .6091
    OQSO for saida.bin       using bits  6 to 10        142638  2.470  .9932
    OQSO for saida.bin       using bits  5 to  9        141954   .151  .5602
    OQSO for saida.bin       using bits  4 to  8        141162 -2.533  .0056
    OQSO for saida.bin       using bits  3 to  7        141994   .287  .6130
    OQSO for saida.bin       using bits  2 to  6        142059   .507  .6940
    OQSO for saida.bin       using bits  1 to  5        142140   .782  .7829
     DNA for saida.bin       using bits 31 to 32        140969 -2.774  .0028
     DNA for saida.bin       using bits 30 to 31        142173   .778  .7817
     DNA for saida.bin       using bits 29 to 30        141849  -.178  .4294
     DNA for saida.bin       using bits 28 to 29        142156   .728  .7666
     DNA for saida.bin       using bits 27 to 28        141498 -1.213  .1125
     DNA for saida.bin       using bits 26 to 27        141884  -.075  .4702
     DNA for saida.bin       using bits 25 to 26        141750  -.470  .3192
     DNA for saida.bin       using bits 24 to 25        142412  1.483  .9309
     DNA for saida.bin       using bits 23 to 24        141977   .200  .5791
     DNA for saida.bin       using bits 22 to 23        142010   .297  .6168
     DNA for saida.bin       using bits 21 to 22        141645  -.780  .2178
     DNA for saida.bin       using bits 20 to 21        142302  1.158  .8766
     DNA for saida.bin       using bits 19 to 20        142042   .391  .6522
     DNA for saida.bin       using bits 18 to 19        141401 -1.499  .0669
     DNA for saida.bin       using bits 17 to 18        141574  -.989  .1613
     DNA for saida.bin       using bits 16 to 17        141817  -.272  .3927
     DNA for saida.bin       using bits 15 to 16        142148   .704  .7593
     DNA for saida.bin       using bits 14 to 15        142169   .766  .7782
     DNA for saida.bin       using bits 13 to 14        141335 -1.694  .0451
     DNA for saida.bin       using bits 12 to 13        141722  -.553  .2903
     DNA for saida.bin       using bits 11 to 12        142508  1.766  .9613
     DNA for saida.bin       using bits 10 to 11        141642  -.789  .2152
     DNA for saida.bin       using bits  9 to 10        141508 -1.184  .1182
     DNA for saida.bin       using bits  8 to  9        142247   .996  .8404
     DNA for saida.bin       using bits  7 to  8        141926   .049  .5196
     DNA for saida.bin       using bits  6 to  7        142218   .911  .8187
     DNA for saida.bin       using bits  5 to  6        142036   .374  .6457
     DNA for saida.bin       using bits  4 to  5        141949   .117  .5466
     DNA for saida.bin       using bits  3 to  4        141705  -.603  .2733
     DNA for saida.bin       using bits  2 to  3        142267  1.055  .8543
     DNA for saida.bin       using bits  1 to  2        141532 -1.113  .1328
-------------------------------------------------------------------------------
   Test results for saida.bin
 Chi-square with 5^5-5^4=2500 d.of f. for sample size:2560000
                               chisquare  equiv normal  p-value
  Results fo COUNT-THE-1's in successive bytes:
 byte stream for saida.bin        2507.10       .100      .540019
 byte stream for saida.bin        2596.27      1.361      .913321
-------------------------------------------------------------------------------
 Chi-square with 5^5-5^4=2500 d.of f. for sample size: 256000
                      chisquare  equiv normal  p value
  Results for COUNT-THE-1's in specified bytes:
           bits  1 to  8  2478.15      -.309      .378634
           bits  2 to  9  2456.48      -.616      .269101
           bits  3 to 10  2605.03      1.485      .931277
           bits  4 to 11  2475.59      -.345      .364995
           bits  5 to 12  2666.04      2.348      .990568
           bits  6 to 13  2437.06      -.890      .186717
           bits  7 to 14  2459.38      -.574      .282823
           bits  8 to 15  2504.35       .062      .524520
           bits  9 to 16  2456.36      -.617      .268580
           bits 10 to 17  2437.80      -.880      .189510
           bits 11 to 18  2623.20      1.742      .959273
           bits 12 to 19  2580.91      1.144      .873747
           bits 13 to 20  2487.44      -.178      .429481
           bits 14 to 21  2456.20      -.619      .267822
           bits 15 to 22  2458.71      -.584      .279652
           bits 16 to 23  2589.98      1.273      .898404
           bits 17 to 24  2463.19      -.521      .301318
           bits 18 to 25  2353.40     -2.073      .019075
           bits 19 to 26  2538.48       .544      .706821
           bits 20 to 27  2406.35     -1.324      .092679
           bits 21 to 28  2425.19     -1.058      .145031
           bits 22 to 29  2410.35     -1.268      .102426
           bits 23 to 30  2553.25       .753      .774291
           bits 24 to 31  2547.10       .666      .747304
           bits 25 to 32  2607.51      1.520      .935805
-------------------------------------------------------------------------------
           CDPARK: result of ten tests on file saida.bin
            Of 12,000 tries, the average no. of successes
                 should be 3523 with sigma=21.9
            Successes: 3535    z-score:   .548 p-value: .708135
            Successes: 3486    z-score: -1.689 p-value: .045562
            Successes: 3533    z-score:   .457 p-value: .676028
            Successes: 3562    z-score:  1.781 p-value: .962529
            Successes: 3538    z-score:   .685 p-value: .753306
            Successes: 3503    z-score:  -.913 p-value: .180558
            Successes: 3532    z-score:   .411 p-value: .659449
            Successes: 3532    z-score:   .411 p-value: .659449
            Successes: 3547    z-score:  1.096 p-value: .863437
            Successes: 3473    z-score: -2.283 p-value: .011212

           square size   avg. no.  parked   sample sigma
             100.            3524.100       26.427
            KSTEST for the above 10: p=  .648382
-------------------------------------------------------------------------------
               This is the MINIMUM DISTANCE test
              for random integers in the file saida.bin
     Sample no.    d^2     avg     equiv uni
           5    1.3787   1.3375     .749829
          10     .0117   1.1296     .011683
          15    1.4851    .9229     .775206
          20     .1637    .9030     .151702
          25    1.1759    .8889     .693289
          30     .4955    .9591     .392263
          35     .7621    .9263     .535093
          40     .2903    .8923     .253050
          45     .0596    .8558     .058102
          50     .0394    .9355     .038793
          55     .5495    .9011     .424325
          60     .1112    .8655     .105701
          65     .2349    .9345     .210254
          70     .2236    .9194     .201255
          75     .7600    .8915     .534135
          80     .1740    .8856     .160427
          85     .0721    .8363     .069860
          90     .8988    .8717     .594758
          95     .5872    .8910     .445750
         100     .0885    .8640     .085103
     MINIMUM DISTANCE TEST for saida.bin
          Result of KS test on 20 transformed mindist^2's:
                                  p-value= .852708
--------------------------------------------------------------------------------
               The 3DSPHERES test for file saida.bin
 sample no:  1     r^3=  28.766     p-value= .61668
 sample no:  2     r^3=   4.676     p-value= .14432
 sample no:  3     r^3=  10.218     p-value= .28866
 sample no:  4     r^3=  69.193     p-value= .90039
 sample no:  5     r^3=   5.989     p-value= .18097
 sample no:  6     r^3=  17.869     p-value= .44879
 sample no:  7     r^3=   4.553     p-value= .14082
 sample no:  8     r^3=  12.876     p-value= .34896
 sample no:  9     r^3=  32.553     p-value= .66213
 sample no: 10     r^3=  47.344     p-value= .79364
 sample no: 11     r^3=   4.166     p-value= .12966
 sample no: 12     r^3=  31.622     p-value= .65148
 sample no: 13     r^3=   1.461     p-value= .04754
 sample no: 14     r^3=  33.158     p-value= .66887
 sample no: 15     r^3=  52.608     p-value= .82685
 sample no: 16     r^3=  31.461     p-value= .64961
 sample no: 17     r^3=  21.009     p-value= .50357
 sample no: 18     r^3= 189.053     p-value= .99817
 sample no: 19     r^3=  48.964     p-value= .80449
 sample no: 20     r^3=  12.580     p-value= .34252
       3DSPHERES test for file saida.bin            p-value= .071285
--------------------------------------------------------------------------------
            RESULTS OF SQUEEZE TEST FOR saida.bin
         Table of standardized frequency counts
     ( (obs-exp)/sqrt(exp) )^2
        for j taking values <=6,7,8,...,47,>=48:
      .6     -.3     -.8     -.8    -1.3      .1
    -1.3      .8     -.2     -.7     -.6     1.0
      .5      .4      .1     -.7     -.5      .5
     1.4      .8     -.1     -.5      .1     -.3
     -.6      .4      .1    -2.1    -1.7     -.1
     -.3     1.1     1.6    -1.3    -1.0     1.9
    -1.2      .2     -.8    -1.3     -.6      .0
     1.8
           Chi-square with 42 degrees of freedom: 37.196
              z-score=  -.524  p-value= .318165

--------------------------------------------------------------------------------
                Test no.  1      p-value  .896175
                Test no.  2      p-value  .352902
                Test no.  3      p-value  .668301
                Test no.  4      p-value  .397798
                Test no.  5      p-value  .184775
                Test no.  6      p-value  .905759
                Test no.  7      p-value  .529621
                Test no.  8      p-value  .924611
                Test no.  9      p-value  .518276
                Test no. 10      p-value  .768969
   Results of the OSUM test for saida.bin
        KSTEST on the above 10 p-values:  .656776
--------------------------------------------------------------------------------
           The RUNS test for file saida.bin
     Up and down runs in a sample of 10000

                 Run test for saida.bin      :
       runs up; ks test for 10 p's: .952766
     runs down; ks test for 10 p's: .435459
                 Run test for saida.bin      :
       runs up; ks test for 10 p's: .965311
     runs down; ks test for 10 p's: .219269
--------------------------------------------------------------------------------
                Results of craps test for saida.bin
  No. of wins:  Observed Expected
```

```
                             99075    98585.86
 Chisq=  24.38 for 20 degrees of freedom, p=  .77366
              Throws Observed Expected  Chisq     Sum
           SUMMARY  FOR saida.bin
               p-value for no. of wins: .985655
               p-value for throws/game: .773663
  Test completed.  File saida.bin
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

This test strongly indicates that the diffusion function used in the Viktoria hash algorithm is effective, this routine being a part of the mixword() function.

4.3 Block_change function()

This function is responsible for the final exchange of the block where the elements of sub-blocks **A**, **B**, **C** and **D** are exchanged and form a new data block (see algorithm 7). For a file with 1.000.000.000 bytes only filled with "0" bytes we have the following permutations:

BLOCK	PERMUTATION
1	51 56 12 11 14 5 61 58 17 22 33 49 53 23 21 55 35 47 8 63 1 29 50 0 6 36 45 20 26 39 40 37 10 62 31 24 9 4 41 15 7 16 27 30 32 59 46 54 19 38 3 13 60 57 25 34 48 52 2 43 28 42 18 44
2	13 27 9 52 7 14 4 39 51 49 26 63 20 33 25 44 57 21 36 62 23 32 38 31 50 56 0 5 61 40 15 24 1 3 12 54 41 55 28 37 18 6 60 22 35 19 30 46 47 59 2 10 16 8 42 34 48 58 53 29 11 43 17 45
3	4 28 43 40 30 50 63 59 15 55 14 27 17 45 53 22 34 24 6 39 21 35 16 7 42 3 32 37 13 62 47 38 33 41 49 12 31 54 2 25 19 0 10 61 56 20 60 52 18 5 51 48 9 11 36 46 44 23 1 57 29 26 8 58
4	18 26 50 37 61 5 16 38 1 56 57 0 32 34 2 20 35 13 59 17 30 62 48 8 39 28 4 12 25 15 24 10 9 3 58 45 7 36 44 54 55 46 41 6 23 49 63 11 22 52 51 31 27 60 53 29 40 43 14 19 21 42 33 47
5	38 31 2 3 15 16 43 18 61 42 52 34 51 35 55 25 0 9 60 37 20 50 54 5 40 6 19 49 56 57 23 4 30 33 24 59 32 7 36 22 41 11 13 62 47 27 53 45 39 21 29 48 10 17 46 12 8 58 28 14 63 44 26 1
6	35 38 59 37 5 11 62 15 60 58 54 26 48 28 63 12 27 2 23 3 10 25 4 43 52 29 41 39 13 47 18 22 20 53 33 0 17 34 8 46 49 40 7 6 57 9 51 32 42 50 16 44 55 24 14 21 19 61 36 1 31 45 56 30
7	31 22 9 36 6 16 38 44 51 56 53 1 62 57 52 7 32 46 10 26 0 54 17 14 33 27 28 63 58 30 59 50 35 47 25 45 39 48 23 3 5 40 15 20 2 19 60 11 29 18 4 24 61 8 49 41 13 34 37 43 12 42 55 21
8	12 24 36 60 55 32 51 35 33 4 49 34 23 48 20 44 8 59 13 0 43 18 31 15 27 47 38 9 63 61 1 53 52 25 62 40 17 29 42 26 21 3 54 28 16 37 22 45 11 6 57 46 10 2 58 30 41 5 39 19 7 14 56 50

Table 14

These permutations are totally dependent on the exchange tables T1 and T2 which are dynamic. Each round 64 bytes of each table are used, which makes a cycle of 8 rounds. As each block processing corresponding to 16 rounds of the mixword() function we have a very big change in the swap tables which makes the cycle not repeat easily. See table 15 for the next 8 permutations of blocks:

BLOCK	PERMUTATION
9	53 36 2 4 58 14 29 22 63 49 57 5 59 12 27 62 52 32 42 7 56 1 37 20 41 17 0 16 21 45 61 35 6 19 8 34 28 9 10 23 54 30 48 26 39 3 11 33 43 44 24 38 55 47 25 60 46 31 40 51 13 50 15 18
10	63 54 58 29 22 21 20 31 61 18 32 50 41 49 25 11 9 15 55 39 26 60 5 10 6 57 2 16 40 28 48 17 3 12 37 1 45 24 0 19 56 42 35 7 23 4 13 59 51 33 46 52 47 27 30 44 8 53 14 43 34 38 36 62
11	10 4 2 25 35 18 62 20 54 44 36 6 17 63 5 11 22 60 33 27 48 3 57 8 14 28 45 29 40 50 41 37 39 59 42 51 13 43 52 53 38 30 7 31 21 49 15 61 23 55 0 19 46 58 9 16 24 56 47 26 34 12 32 1
12	18 35 42 59 57 15 29 48 23 1 54 14 62 61 27 43 31 32 50 8 20 37 63 12 10 25 21 56 52 38 58 11 40 36 45 16 2 7 44 3 55 51 13 28 33 39 34 26 17 30 41 5 9 60 0 4 53 46 22 49 47 19 24 6
13	37 61 11 2 17 14 48 27 4 15 36 47 20 58 54 35 49 63 33 12 16 0 1 50 30 9 5 6 21 46 51 40 44 45 7 43 22 42 55 41 62 57 32 25 28 8 23 52 39 13 38 56 60 59 34 3 18 10 31 29 53 24 19 26
14	44 36 21 43 38 9 7 57 27 8 53 52 45 5 41 59 63 2 12 33 17 47 39 20 32 4 28 42 24 14 50 23 22 48 19 34 58 31 10 26 1 61 18 54 25 16 30 40 49 60 0 51 15 11 13 46 35 62 55 3 56 29 37 6
15	24 39 44 22 33 26 48 51 47 13 38 10 6 11 5 19 8 4 7 25 18 63 20 59 45 21 60 58 54 37 52 61 31 2 57 12 23 27 15 16 62 35 46 36 17 55 50 9 40 56 32 41 53 29 1 14 43 28 3 49 34 42 30 0
16	29 61 8 31 40 59 58 52 43 56 30 14 57 42 10 24 5 28 53 39 54 34 50 15 0 2 9 33 36 1 23 46 20 27 25 18 38 26 49 55 21 13 6 60 35 48 16 17 19 41 4 51 63 44 32 37 12 47 11 45 7 22 3 62

Table 15

From a 1 gigabyte file with bytes "0" we obtained positive results for this permutation routine. Each of the 64 numbers (from 0 to 63) appear in the 64 positions (position 0 to 63) a minimum of 242.347 times and a maximum of 245.820 times. The average that should appear for each number in each position is 244.154 times. It remains to be observed that the analyzed file only contains "0" bytes so that the content of the message has no great influence in this sense. Extremely redundant messages still generate balanced exchange results.

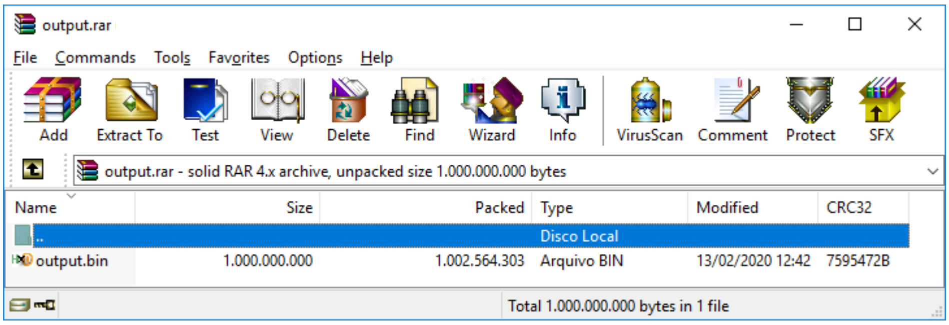
5. STATISTICAL TESTING

The Viktoria hash algorithm processes the message and produces supposedly pseudo-random outputs. To verify this hypothesis we submit the algorithm to some statistical tests that can verify the pseudo-randomness of the data. To validate the data output we use the Dieharder battery of statistical tests.

5.1 The compression test

The first and simplest randomness test is the compression test. We can safely say that if a binary sequence is compactable by some algorithm it cannot be random. However, the reverse cannot be said. There are noncompactable sequences that have no characteristics of a random sequence.

To test the mixword function (since it is the heart of the Viktoria hash function) we experimented with generating a 1 gigabyte file size containing in the first three bytes a growing code book, the other bytes were filled with "0". Then we processed each 512-bit block with the mixword function considering the interchange tables T1 and T2 in their initial states and considering each block separately (we omitted the XOR between the current and previous blocks). Then we concatenate all these blocks and generate an output file with 1 gigabyte in size (1.000.000.000 bytes). After this procedure we try to compress the output file with some of the best known compression algorithms (RAR). We used only 4 turns of the mixword() function and the block swap function. There was no compression of the information.



output.rar - solid RAR 4.x archive, unpacked size 1.000.000.000 bytes

Name	Size	Packed	Type	Modified	CRC32
..			Disco Local		
output.bin	1.000.000.000	1.002.564.303	Arquivo BIN	13/02/2020 12:42	7595472B

Total 1.000.000.000 bytes in 1 file

Figure 3

We also performed the Dieharder[3] tests (an evolution of the Diehard test battery) with this same file and the result of the randomization was satisfactory. The complete test is shown in Annex VI.

5.2 The hash function test with 4 rounds for mixword() - version 1

To perform the test we follow these steps:

1. We generated a file with 1.000.000.000 bytes "0".
2. We divided the file into 15.625.000 blocks.
3. We calculate the hash[4] for each block separately disregarding the pre-processing routine and the post-processing routines.
4. We concatenate the result of these hashes in a single output file with 1.000.000.000 bytes.
5. We performed the Dieharder tests on the output file.

[3] The main point of dieharder (like diehard before it) is to facilitate time and test (pseudo)random number generators, both software and hardware, for a variety of research and encryption purposes. The tool is built entirely on top of the GSL random number generator interface and uses a variety of other GSL tools (e.g. sort, erfc, incomplete range, distribution generators) in its operation (https://webhome.phy.duke.edu/~rgb/General/dieharder.php).

[4] In fact only the processing of the algorithm's core functions considering only 4 of the 16 rounds of the mixword() function.

The results of the Dieharder tests performed can be seen in Annex VII and are successful. The data provided from the Viktoria hash algorithm pass in almost all tests.

5.3 The hash function test with 4 rounds for mixword() - version 2

This test is similar to the previous test, however, the input file is filled with 1.000.000.000 bytes "255" instead of bytes "0" as in the previous test. The test result can be seen in Appendix VIII and corroborates the fact that the Viktoria hash function produces supposedly random data with only 4 of the 16 rounds of the mixword() function.

5.4 Bit Shift Test 1 with 4 rounds for mixword() - chained blocks

This test works to verify that changing a single bit in the input data produces large changes in the output of the hash function. In this test we consider the XOR with the previous data block when processing each block.

1. We generate a block of 512 bits whose first bit is "1", the others are zero.
2. We generate a second block of 512 bits where the 2nd bit is "1" and the others "0". Then we generate a third block of 512 bits where the 3rd bit is "1" and the others are "0", and so on.
3. We concatenate the 512 blocks generated in step 2 and form an input file of 32,768 bytes.
4. The structure of the input file is (in hexadecimal, first 8 blocks):

```
1º 01000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
2º 02000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
3º 04000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
4º 08000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
5º 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
6º 20000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
7º 40000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
8º 80000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

5. Using the Viktoria hash function (read_block + mixword + permutation_block functions) disregarding the pre-processing and post-processing routines we generate the output file.
6. The structure of the output file is (in hexadecimal, first 8 blocks):

```
1° block:
        42E982038E0327D54E3F4D36CD15890B579DA4400615A98A98F70836CBF9F192
        E6A90EA154323FEB2038B7DF0641CC55549A2FDDAA8D4D8B898124C6F1E5EEE7
2° block:
        59E4B9832DA633A8739875D763350CD153DDD61F5F5FECC778F5BB8046359C17
        517C6EE27D111B7B122424DA909FE5BECDEE1902C005CEC96344D08CF66BF563
3° block:
        EFE60B01AF55F634E7A5C3CE32A382F33BA5BA2EE39D524CB9404FA2C43CBFF7
        86A85E00A91B682EF75C4FB500B316CB90752192B6EBDE642D4F7D8E223076C8
4° block:
        7ECBEDB792586C6F83CF09BBC25201EAFA781EBF1AC4645EE958545469A71C14
        B1068CDA15A74738246DE83059526C032E1A976C0EA26D3703C0011A336171E7
5° block:
        E2B29E669B9F4EA668FA57EA71F9EBE926E417A027DD185A069FE38398FF29B1
        6216A409C231CFF4157B73E13BBD4FE5482A041D129C04F646B5E9C9B4079707
6° block:
        1560D33B43C563C8F500FC083B283FD5E0109D5C84225FA38B878B72EB48B2D4
        255DC7997CCD379D76CD2655C9E53935E72C26BF988B2EF67C8FC8DBD84072CA
7° block:
        16BF4C5F4A6005DBDC91EEC1453598BDB6966032AB4FEC2EE74274E185DC9F2B
        7A06A922BBDE6CA24AE657D1331D8BDBD6CF15BF4EF511206804CCFC039F1C1F
8° block:
        7E0EAF297FDD052B7822F59B5C12FA643BF8DE81DCDBEE7A1A68A4597C3B6BB0
        144F7E63C53F99540185C22CC20B5AA2A3637C9015033DEA8E3DCDB73F5FDBA9
```

7. Checking the minimum and maximum number of "1" bits in each block.

Minimum: 227
Maximum: 291

In this test we verify if the position change of a bit "1" produces the avalanche effect on the output of the hash function. In fact, according to the test performed, the bit numbers "1" and "0" are balanced.

5.5 The 0 bit shift test with 4 rounds for mixword() - chained blocks

This test is the same as the previous test, only this time we move one bit "0" in place of bit "1". The block fill bits are also "1" instead of bit "0". In this test we consider the XOR with the previous data block when processing each block.

1. We generate a block of 512 bits whose first bit is "0", the others are "1".
2. We generate a second block of 512 bits where the 2nd bit is "0" and the others "1". Then we generate a third block of 512 bits where the 3rd bit is "0" and the others are "1", and so on.
3. We concatenate the 512 blocks generated in step 2 and form an input file of 32.768 bytes.
4. The structure of the input file is (in hexadecimal, first 8 blocks):

```
1º FEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
2º FDFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
3º FBFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
4º F7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
5º EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
6º DFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
7º BFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
8º 7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

5. Using the Viktoria hash function (read_block + mixword + permutation_block functions) disregarding the pre-processing and post-processing routines we generate the output file.
6. The structure of the output file is (in hexadecimal, first 8 blocks):

```
1° block:
39CAA4CD337774FD304D58BFCECB92BF523EC35830EB7F8543A0D68A359ED982
772D0CCFFD9B2B1BBE4EFCC61353FEBBA3367E9B9BC98FA4D8FFB12CCDA48539
2° block:
E383C3D09D26F74910F73621AB4BDF226AAEB3581DDA23D45AED2B2694BFA668
4DD3541D92D779C3564E3D03951F1CF7A61AB64C641857D8DF26EF399C1F4616
3° block:
2697C5DC13203380E623CA6DB405E0B9D77EF93B49588CC2D02A80720CA80A7B
D9F01DCCB8F60C65C06F4425E4B64DECBA67BB56F572D9E5180B42FA9AF650A2
4° block:
A06CFC926FE3996C2ECBF56B5E9D9E2E5D88602EEA39D09C951E6A90EECBFCA1
FC95AC9EDF5BFBE2E35EDF772F24A0E1F7826538BF0D2467DD65DE8B213C2661
5° block:
9253314F07C13FDA5B427C0EA18A34F7D8A3519C824A8E96DB0BF76BD08AA414
BFA0D2E2EE8E951F60FEDEA9CA2DE0F96BDB9D41435D2852FDD9895080DE979C
6° block:
05001430EEDB23D438EBCD48EBE6B2C05D8866CAC68C9A1A0E3906B1BAC1C105
EEE04204AF382401E33B5119BB4AFEE1CBC71D49690525B3C710AE26DF08A787
7° block:
80B641DCF29174BCBC51F7B73CD2B1A6060F3D3B6AED141C7B101AF3CF4358B0
8F940068F82E28D91A3CB94C283ABFB68CF06591FAF56941BCF7CA6EAD490DBE
8° block:
8866CCADF6AA7FD7072489C3E78DFADD2045148FDCCDB13636A7429B9320129F
545BFA603BAEE097496E38E4E06D4FE9828277CA3674A22D01D1709F2EE78F34
```

7. Checking the minimum and maximum number of "0" bits in each block.

Minimum: 216
Maximum: 291

5.6 Bit Shift Test 1 with 4 rounds for mixword() - single blocks

This test is similar to the 5.4 test, except that the blocks processed by the Viktoria hash function are processed in isolation.

1. The structure of the output file is (in hexadecimal, first 8 blocks):

```
1° block:
C22EA1081C789FBC71CA272A57CE3809097F298E9ED3AB5599675C774C087BC2
41048D64F9A071E91DF457CEF951BE60EA56DD54E86511721B7348F6F0352C14
2° block:
0D1629C10069859EF94FAF4E8389B0A8525D88FA3BAFE4100525277E51E21F7A
005D45FB3DE2D84FB6D836C138710523A15FC0C8395EB9EDA05C79C93823C500
3° block:
9E7BC0978F3220C6C3C6C690F508B7187FA94AEBEA57AFF18C04093BEBA34242
5D20868491A72F22C17415D70F9009FDA95183B85E2442316D8DFAD67E037415
4° block:
E86FC9B959D0561C406F11BD8D2E889FCA61547C54C5BF948D376BDC7C96244F
5744C19036AB187EEC0DAF6045CE790C186C7F65910972E5FD0E40D11DC4BB70
5° block:
4C98C73263445F7916EBF36C25392DE9A0E82A61C49472E1AC42859050E3DF10
360BB41FF21B7FB577615EBA15EEFF4FDD90065B89367B2C350BBA3D98F249AD
6° block:
D9C302D88E89D82E05F81DBA5999D0736971B8412CE19033B3D99775A73D6CEC
DDE09F504F71D99524E052DEBEFA0047F2087CBC28A7B41212F9E12F9A90DD15
7° block:
50E79806253E394C8455F6EEA95AFD607A7AD60799AB7F9FCEDE3522B4B7D280
8BC2489B5B21CDF100B120865D0A4E118AE055BD661F99A9CA1E64A2EEFA870E
8° block:
7E5307D52EB046DEA30386DAAE280D3175B348F580193E11C8696504B0E661A5
D6083BB960233BA18FF03BF89D08AC5AD2A67DAC4FB86E159452EB00977D1CC8
```

2. Checking the minimum and maximum number of "1" bits in each block.

Minimum: 225
Maximum: 289

5.7 The 0 bit shift test with 4 rounds for mixword() - single blocks

This test is similar to the 5.5 test, except that the blocks processed by the Viktoria hash function are processed in isolation.

1. The structure of the output file is (in hexadecimal, first 8 blocks):

```
1° block:
0606542816F93C4C134C675E49A087CA190C2DA78344FD61D258BE643FA7D4B4
21DB452B5A2DA7A618F482AB89E00CA50506B030509186AF1148B624EC92D6D0
2° block:
CAFE09C932C2A103677D1C378D8B471717087AE45B8715F9E182784D597B522A
D6C661CE29C88FB38A319CA81837C4DF0096C218350D76C83459858A6C8D65CD
3° block:
3F1FCFB79E653E2E9E48577F1DA62E3771882F87C706134D31D28B9FAB0B43E8
D001C082272E0E096EBEF2B6A7510DFDA81AFBD486637622FF1FA649A3274494
4° block:
0400C22CB9E93060E9D4FB2AFC15F7B1FDA035006AA1E4678C35268D0BB17E02
1A871F55AD752C31DE37E5C773A08D20442A02E0CE9E0A9E7BAF9420DEA38FBB
5° block:
A362D3DC58884712EA74B2FA0C01CC4927FF4A4EF3DC240690F6AD9C3A2312F2
343A4B586B23072D193D9F796E7076032FF4FDDCAA5434706E7D01B67FEDF338
6° block:
```

```
5ACF38386B141995965F223FE8EA327E654E659576C346E9F7DAD3B2A1514216
E978479908F9CA9597E6356F8F21D9E0B0EFAEE337DA68DCB5F214008E0E4237
7° block:
F72575D36DC73A82C245346B6B1141B7B2290A7E70508DD5447D9DFEC908F112
3948CB73F9B4C5E72FA32DD73C0449A4D6A92A6620E9057E78AA8086483BFF58
8° block:
1E947967C141DF53DC31AB5F930AEB5A5AA8FD3767B752BA5B5BB1B8F8D03328
6C19FBEA7C574BFD2B4AAFBC6B039AF15B960BA38D8A9DDFD482B68EB0EC20E5
```

2. Checking the minimum and maximum number of "0" bits in each block.

Minimum: 225
Maximum: 295

5.8 The long file test with 4 rounds for mixword() - isolated blocks

This test is executed from a file with 10.000.000.000 bytes (there are 156.250.000 blocks of 512 bits). Each 512 bit block is 64 bytes. In each block the first 4 bytes vary in this order 00-00-00, 01-00-00, 02-00-00, ..., 8D-2F-50-09, 8E-2F-50-09 and 8F-2F-50-09 (hexadecimal notations) and the others are filled with "0". We then use the mixword function (4 turns) to process each block separately, disregarding the pre-processing and post-processing routines. The result of Dieharder tests to check the supposed randomness of the data is excellent. The result is presented in Annex IX.

5.9 The long file test with 4 rounds for mixword() - chained blocks

This test is executed from a file with 10.000.000.000 bytes (there are 156.250.000 blocks of 512 bits). Each 512 bit block is 64 bytes "0". We then use the mixword function (4 loops) to process each block with sequencing, disregarding the pre-processing and post-processing routines. The result of the Dieharder test is presented in Annex X and indicates supposed randomness in the data.

5.10 The superlong file test with 4 rounds for mixword() - isolated blocks

This test is executed from a file with 50.000.000.000 bytes (there are 781.250.000 blocks of 512 bits). Each 512 bit block is 64 bytes. In each block the first 4 bytes vary in this order 00-00-00, 01-00-00, 02-00-00, ..., 8D-2F-50-09, 8E-2F-50-09 and 8F-2F-50-09 (hexadecimal notations) and the others are filled with values from "01" to "3C". We then use the mixword function (4 turns) to process each block separately, disregarding the pre-processing and post-processing routines. The result of Dieharder tests to check the supposed randomness of the data is very good. It is shown in Annex XI.

5.11 The superlong file test with 4 rounds for mixword() - chained blocks

This test is executed from a file with 50.000.000.000 bytes (there are 781.250.000 blocks of 512 bits). Unlike the previous test in this test the file is filled entirely with "FF" bytes. Once again we use the mixword function (4 loops) to process each block in a chained way, disregarding the pre-processing and post-processing routines. The result of Dieharder tests to check the supposed randomness of the data is satisfactory. It is shown in Annex XII.

5.12 Testing the super-long file with full Viktoria hash - central processing

In this test we use an input file of 100.000.000.000 bytes "0" (that's 1.562.500.000 blocks). In this case we use the Viktoria hash function almost as a whole, excluding only the pre-processing and post-processing. The results of the Dieharder tests are excellent and can be seen in Annex XIII.

5.13 Completion of tests

The statistical tests to which we submitted the Viktoria hash algorithm prove that it produces pseudo-random data already from 4 rounds of the mixword function. In the complete algorithm there are 16 runs of the mixword function and the permutation after each block, besides the pre-processing and post-processing routines.

According to the tests performed, the Viktoria hash algorithm makes correct use of the avalanche effect in its internal structure and produces accurate pseudo-random data sets that are very important for hash algorithms. This feature helps to avoid attacks based on bit relationships such as linear and differential cryptoanalysis.

6. COMPARISON WITH OTHER HASH FUNCTIONS

In this part of the work we present some comparisons of VIKTORIA HASH with SHA2-512 and with SHA3-512.

6.1 Sequence search test on 16-bit messages

This test looks for hexadecimal number sequences in hashes produced by SHA2-512, SHA3-512 and VIKTORIA algorithms. We search the hexadecimal sequences "000000", "00000", "0000", "000", "00" and "0" for all hashes produced by the 3 functions and compare the results. Then we look for the sequences "111111", "222222", ..., up to "ffffff".

6.1.1 Searching hex "0":

Here we present the hash values for the hexadecimal "0" and its concatenations. We see in table 16 (an example for each sequence of each algorithm) that for each function the bit sequences were found within the statistical forecast. The complete results can be seen in Annex XIV. There they are detailed how many times in each position the sequences appeared for each algorithm. Also at the end of each table you can check the sum of the terms found, the term that appears most in the ratio and the average of the appearances of binary sequences for each algorithm. We analyzed 65536 hashes which correspond to all possible 16-bit messages.

Function	**Hexadecimal**	**Hexadecimal Message**	**Hash**
SHA2-512	**000000**	-	-
SHA3-512	**000000**	d5d4	7ff146ae9933db67846e46e1b161841197f203ca14c28a5de4afdb4df17b5450ff50685afceed5fdc275**000000**4a1bb232a89383dc6d6864adf5f35abd6889be
VIKTORIA	**000000**	dfd8	5f041dcec155a77e8deffb5b4b7fb02b82fa2e5df5047b18ccac66cfbc826582417a51203a6920f0e0b4bf**000000**cf4b049bf6f4db3bba822de870548c37ebe7
SHA2-512	**00000**	180d	897c8bc4bcfa1446cee003dfd5cc9c4f5e03438d**00000**a3a4b6554a16ca42d64ec943c7f0dfb8c1f8562f7cb11f58079fefd7eb4cf187ea139222e0d6d7fa854
SHA3-512	**00000**	1272	d1dd2a17eaf0dbedf2260b8327f**00000**b2aa62e11b4eecb1f40e73d0595b9f69979cb14a3038ae0ad24d6eec3af8a3dfd9fa4e41d48f8dd9ba7855c4c39225f2
VIKTORIA	**00000**	feeb	1327c**00000**b186766373d4b9b2e2e58dff717662360a9652b607f0df226e2ce747ee71f0fa50e416c1c4da0029172fce9f8d2f354f685eb2dda476b2f49e2b51
SHA2-512	**0000**	ec72	**0000**d70af47141ffcddb05761f6ea99369abc49fc973ab5b70a0e6174a3208fbbcdc246da51bcfbca2ee4c862e86ae3ad8321f2b7254268dd0ff8a4ee587c0d7
SHA3-512	**0000**	5c1c	**0000**38d34f0b27d1604b090e6b51ff9a37cabc95ee35a6528e8e4a5a281f1f408396b7843681aba907e06554f827435b46a1ca3259b3c076fd01acff5b17e6cc
VIKTORIA	**0000**	2b5b	87c**0000**e4361b8f8476b6f23787c0e6a8366901ac459886259a2ef827b996a1ab125b439e45827384f1da2

			2ddb049694dcf648da2d890b1b5c68881732b09816
SHA2-512	**000**	0123	**000**4b80f21c57a47b074aaa34abc16f0a9c0a9a45808cdad2f267ea0bd6b8843231d55ace3b1a38b187dc07ea4f545b09d0575eceb635979351cc1bfdb0209a6
SHA3-512	**000**	17a3	**000**eb7599f9bf5b16cd9c220e46287ab2d43eb2024a3b521d63a12ae1dbf8a68ea5229c43c7c3387219b2ba509a2e6d38c2485b0c4ed27b917ec0267d5b30bb5
VIKTORIA	**000**	1105	**000**94da1de5aa36b4d81163d2e74e6301e06e8505c276d7f380e5782f9611232bdc7f91288223a55459bcc1d6fd665e05d3ff88964cff4a6f65dde4cdd0e87c6
SHA2-512	**00**	00f9	**00**52acb042f490c6ae205cc29b9ea875161f5866328537de85557d15df2600b783b0b48d1c59284c14dc445feb2b102f8bcd467d12d0cc776d31c324dce7099c
SHA3-512	**00**	00dc	**00**fd311aed14b59fe0f6473795042a4d4cf5357574f63a76e07f247ac1174b579033fb42789bfb065d09ccc8d5f51735d815c0d200950958090d103cc24e466c
VIKTORIA	**00**	7800	**00**a9cd79cedfe62f915ca89592aa90cb51aab990b4a5999d14cb39b24c9102c92f89cb602599c6f3783e7d3592a06a1a0b847295baabcd267437b911f53c2718
SHA2-512	**0**	0007	**0**83c0151f931208dcb4b0134762c30d1858c6cafa40eaeb4113b69717dc286ac69a890b548b7dfb489cd3b2527903ac45236bb13af8d2c5f2f27807c6d62b6e7
SHA3-512	**0**	0021	**0**24ad19e301c6bf99dbcbd630a1a439c3c36b8840eb627f513d175690ba386f2fea9550d1fa9c304284f3413e554a1b3e4858be9456edb93ce2b0ec6cc97883e
VIKTORIA	**0**	0f00	**0**85a529f5878f6038455bac3d6866476dfd87c151a8e5caab89e43f67b434bf22d49da05d4e31c9d0fffc712431e711fb3eb278cb6bbcc202164fb75d69f4a08

Table 16

6.1.1 Searching for hexadecimal (other values):

The results of searches for hexadecimal values for '123456789abcdef' may be followed in Annex XV.

6.1.2 Conclusion

For all the values searched, the three algorithms behaved in a very similar way, presenting on average all the occurrences of hexadecimal values searched in the 65536 hashes analyzed.

Hexadecimal value sought and concatenations	**Expected average**	**Observed average SHA2-256**	**Observed average SHA3-256**	**Observed average Viktoria**
000000	0,00390625	0	0,00813	0,00813
00000	0,0625	0,06452	0,06452	0,10484
0000	1	0,896	0,952	1,032
000	16	16,2778	15,6508	15,6429
00	256	257,331	255,039	255,173
0	4096	4100,54	4092,31	4090,92
111111	0,00390625	0	0,01626	0
11111	0,0625	0,03226	0,09677	0,01613

1111	1	0,944	1,144	0,928
111	16	16,031746031746	16,031746031746	15,8650793650794
11	256	257,1496062992	256,76377952755	255,661417322835
1	4096	4093,2890625	4097,234375	4094,03125
222222	0,00390625	0,008130081	0	0
22222	0,0625	0,096774194	0,040322581	0,024193548
2222	1	0,88	0,928	0,776
222	16	15,61904762	16,05555556	15,93650794
22	256	252,480315	259,8188976	253,6141732
2	4096	4086,695313	4104,515625	4089,5
333333	0,00390625	0,008130081	0	0
33333	0,0625	0,10483871	0,032258065	0,056451613
3333	1	1,016	0,816	1,08
333	16	15,51587302	15,35714286	16,68253968
33	256	256,488189	254,1811024	253,9370079
3	4096	4091,914063	4096,671875	4092,171875
444444	0,00390625	0,008130081	0,016260163	0,008130081
44444	0,0625	0,056451613	0,064516129	0,048387097
4444	1	0,992	0,848	0,832
444	16	16,18253968	15,17460317	15,62698413
44	256	256,1968504	254,1259843	252,480315
4	4096	4097,945313	4094,140625	4093,648438
555555	0,00390625	0	0	0
55555	0,0625	0,072580645	0,056451613	0,056451613
5555	1	0,896	1,08	0,96
555	16	16,08730159	15,82539683	16,06349206
55	256	255,3543307	257,1259843	256,9133858
5	4096	4085,976563	4112,242188	4095,617188
666666	0,00390625	0	0	0
66666	0,0625	0,064516129	0,096774194	0,048387097
6666	1	0,992	1,144	0,92
666	16	16,5	16,5	15,8015873
66	256	256,3149606	256,3149606	256,0944882
6	4096	4097,570313	4092,570313	4088,71875
777777	0,00390625	0	0	0
77777	0,0625	0,048387097	0,048387097	0,064516129
7777	1	0,936	0,848	0,928
777	16	15,41269841	15,88888889	15,48412698
77	256	254,1811024	256,3622047	256,2125984
7	4096	4102,71875	4091,992188	4110,007813

888888	0,00390625	0	0	0
88888	0,0625	0,112903226	0,040322581	0,10483871
8888	1	1,096	1,12	1,056
888	16	15,85714286	15,57936508	16,04761905
88	256	257,7322835	255,7244094	255,4724409
8	4096	4103,476563	4099,054688	4091,710938
999999	0,00390625	0	0,016260163	0,008130081
99999	0,0625	0,056451613	0,072580645	0,072580645
9999	1	1,016	0,944	0,888
999	16	15,67460317	15,9047619	16,16666667
99	256	254,7086614	257,2677165	256,3464567
9	4096	4094,242188	4096,195313	4100,445313
aaaaaa	0,00390625	0,008130081	0	0
aaaaa	0,0625	0,064516129	0,056451613	0,056451613
aaaa	1	1,016	1,056	0,872
aaa	16	16,03174603	15,88888889	15,68253968
aa	256	254,8503937	253,9370079	256,8582677
a	4096	4097,328125	4087,242188	4089,5625
bbbbbb	0,00390625	0,008130081	0	0,008130081
bbbbb	0,0625	0,064516129	0,056451613	0,088709677
bbbb	1	1,216	0,856	1,144
bbb	16	15,46825397	15,87301587	16,20634921
bb	256	255,2519685	255,023622	258,0551181
b	4096	4092,398438	4101,15625	4103,789063
cccccc	0,00390625	0,016260163	0,008130081	0
ccccc	0,0625	0,072580645	0,072580645	0,040322581
cccc	1	0,864	1,024	0,96
ccc	16	15,88888889	16,29365079	15,21428571
cc	256	257,5590551	252,8188976	255,8740157
c	4096	4103,507813	4089,945313	4092,273438
dddddd	0,00390625	0	0	0,016260163
ddddd	0,0625	0,056451613	0,016129032	0,10483871
dddd	1	0,936	1	1,048
ddd	16	15,6031746	16,41269841	15,70634921
dd	256	253,9448819	256,4251969	256,6535433
d	4096	4097,40625	4102,296875	4101,164063
eeeeee	0,00390625	0	0,024390244	0,008130081
eeeee	0,0625	0,056451613	0,10483871	0,088709677
eeee	1	0,96	0,936	1,224
eee	16	15,48412698	16,20634921	16,34920635

ee	256	253,0708661	258,8110236	257,7165354
e	4096	4095,085938	4089,484375	4101,976563
ffffff	0,00390625	0	0	0,008130081
fffff	0,0625	0,072580645	0,072580645	0,10483871
ffff	1	0,888	0,912	1,032
fff	16	16,06349206	16,23809524	15,64285714
ff	256	255,0708661	255,9370079	255,1732283
f	4096	4095,90625	4088,945313	4090,921875

Table 17

Table 17 presents the test summary and shows that the Viktoria hash algorithm has outputs comparable to SHA2-512 and SHA3-512 algorithms. According to the test performed the Viktoria algorithm produces balanced outputs and makes good use of the avalanche effect in its internal structure. These are the minimum acceptable characteristics of a good hash function.

6.2 Differential test with functions SHA2-512, SHA3-512 and VIKTORIA

One of the main cryptographic analysis tools is differential cryptoanalysis. It is generally based on the differences of two inputs or outputs of an algorithm where these inputs have peculiar differences. This test is designed to test a possible vulnerability of hash functions to differential cryptoanalysis.

In this test we generated 16384 distinct but very similar files and from them their respective hashes. To generate the test file we XORed the distinct hashes (all possible combinations of pairs). The file created has 8.589.410.304 bytes. The following pseudocode exemplifies this process:

ALGORITHM 11

```
B1 = [128,64,32,16,8,4,2,1]
b2 = [127, 191, 223, 239, 247, 251, 253, 254]
vector = array[16384]

    counter = 1
    for ct3 := 0 to 15
          for ct:= 1 to 64
                for ct2:= 1 to 8
                      prefix = replicate(chr(ct3),ct-1)
                      word = chr(b1[ct2])
                      suffix =  replicate(chr(ct3),64-1-(ct-1))
                      vector[counter] = prefix + word + suffix
                      ++counter
                next
          next
    next

    for ct3 := 0 to 15
          for ct:= 1 to 64
                for ct2:= 1 to 8
                      prefix = replicate(chr(ct3),ct-1)
                      word = chr(b2[ct2])
                      suffix =  replicate(chr(ct3),64-1-(ct-1))
                      vector[counter] = prefix + word + suffix
                      ++counter
                next
          next
    next
```

The test result is very similar for the three hash algorithms. The file of 8.589.410.304 bytes is tested by the battery of Dieharder pseudo-random number tests. The complete result can be seen in Annexes XVI, XVII and XVIII.

TESTS	SHA2-512	SHA3-512	VIKTORIA
SUCCESS	95	91	91
FAIL	17	16	17
POOR PERFORMANCE	2	7	6

Table 18

The test summary is shown in table 18. The results are very similar among the three algorithms indicating that they are at a similar level when dealing with the difficulty of implementing differential attacks.

6.3 Performance review for SHA2-512, SHA3-512 and Viktoria

In this test we analyze the behavior of the three hash functions regarding their processing speed. We use different file sizes to check for possible oscillations. The tests were performed on a computer with intel Core i5 processor - 3210M, 2.5 GHZ and 6 GB of RAM.

FILE	SHA2-512	SHA3-512	VIKTORIA
1 KB	0m0,002s	0m0,027s	0m0,062s
100 KB	0m0,003s	0m0,030s	0m0,067s
500 KB	0m0,007s	0m0,034s	0m0,091s
1 MB	0m0,012s	0m0,040s	0m0,101s
100 MB	0m0,369s	0m1,114s	0m6,438s
500 MB	0m1,818s	0m5,464s	0m32,118s
1 GB	0m3,704s	0m11,163s	1m5,790s

Table 19

In table 19 we see the test result. The SHA-512 algorithm is the fastest of the three that can process a 1.073.741.824 byte file in just under 4 seconds. The SHA-512 algorithm has a good performance too and processes the same file in just over 11 seconds. Viktoria hash is by far the most expensive algorithm due to its complex structure and spends almost 66 seconds processing the same file. Table 20 shows these comparisons.

ALGORITHM	SHA2-512	SHA3-512	VIKTORIA
SHA2-512	-	3,013768898	17,76187905
SHA3-512	0,331810445	-	5,893576995
VIKTORIA	0,05630035	0,169676243	-

Table 20

In practice Viktoria hash is the slowest algorithm, being 17,8 times slower than SHA2-512 and 5.9 times slower than SHA3-512. Despite this Viktoria hash can process in a single thread 16.320.745,16 bytes per second in our reference implementation (it is not fully optimized). In a computer capable of working with 6 threads (something common nowadays) Viktoria hash can match the SHA3-512 algorithm by processing 6 files at the same time.

An important note to note is that the Viktoria hash structure is more complex than the SHA3-512 structure. Particularly due to its dynamic permutation we conjecture that the cost of cryptoanalysis may justify a waste of time in calculating hash values. As we have few hash functions available Viktoria hash presents itself with an interesting alternative mainly for its innovative design.

7. How to use the Viktoria hash function

We have made a reference implementation in the C language of the Viktoria hash function. The implementation was written in the simplest way possible (it was optimized but not entirely) to facilitate the understanding of the algorithm.

The Viktoria hash algorithm can be operated from the linux or windows command line (just compile for each platform). The compilation only requires the GCC compiler:

```
gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

To compile use the following command (the source file is vik.c):

```
gcc vik.c -o vik
```

Suppose you want to see the 512-bit hash of a file called 1kb.bin. To do this type in the linux command line (in windows just omit the "./"):

```
./vik 1kb.bin 1 0 0
```

The Viktoria hash algorithm has 4 mandatory parameters:

Parameter		Significance
1	**File name (string)**	It's the name of the file you want to extract the hash.
2	**Hash size (numeric)**	It represents a number that indicates the hash size to calculate: 1 = 512 bits 2 = 1024 bits 3 = 1536 bits 4 = 2048 bits and so on.
3	**Bits to insert (numeric 0 to 7)**	Used to extract the hash of byte-oriented binary messages (whose last byte is incomplete). This number represents the number of bits to be added at the end of the file.
4	**Byte representing the bits (numeric from 0 to 127)**	Byte that represents the surplus bits that will be inserted at the end of the file for processing.

Table 21

Although the Viktoria algorithm accepts file entries of up to 2^{480}-1 bytes the reference version is limited to 2^{64}-1 bytes. This was done to simplify the implementation and not to need to use external libraries.

CONCLUSION

We present in this work a new hash function: Viktoria Hash. It is a function with an innovative internal design and which, according to the tests performed, seems to provide security and usability for modern times.

Some works after this one leap to our eyes: create attacks for weakened versions of Viktoria (this test is important as in the work [6] that attacks a weakened version of the BLAKE algorithm, finalist of the SHA-3 contest), try to find collisions or pseudo-collisions through various techniques as in [7][9][10][11], implement an optimized version of the algorithm among other works.

Bouillaguet [12] says there seems to have been a problem with the Merkle-Damgard construction. Is this model of hash function construction really outdated? Or is this construction model alone being "blamed" for the design flaws in the algorithms that were broken? These are important questions for further research. A note should be made that although the Viktoria algorithm is based on the Merkle-Damgard construction it does not contain its vulnerabilities because it has a huge internal state and also pre-processing and post-processing functions.

We believe that Viktoria is crash resistant, resistant to the first pre-image and resistant to the second pre-image. These requirements are absolutely indispensable for a good hash function [3][13]. In later works various tests will be applied to this new hash function but for now, according to the tests performed, we can say that Viktoria seems to be a reliable hash algorithm.

REFERENCES

[1] A. menezes; P. van Oorschot; S. Vanstone. **Handbook of Applied Cryptography.** CRC Press, 1996. Available at: <http://cacr.uwaterloo.ca/hac/about/chap9.pdf>. Last access: February 05, 2017.

[2] WANG, Xiaoyun; YIN, Yiqun Lisa; YU, Hongbo. **Finding Collisions in the Full SHA-1**. 2005. Available at: <https://www.iacr.org/archive/crypto2005/36210017/36210017.pdf>. Last access: February 05, 2017.

[3] Black J., Cochran m., Highland T. **A Study of the MD5 Attacks: Insights and Improvements.** 2006. Available at: <https://www.cs.colorado.edu/~jrblack/papers/md5e-full.pdf>. Last access: February 05, 2017.

[4] Marsaglia, George. **The marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness.** Copyright 1995. Available at: <https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/>. Last access: 12 February 2020.

[5] mironov, Ilya. **Hash functions: Theory, attacks, and applications**. 2005. Available at: <https://www.microsoft.com/en-us/research/wp-content/uploads/2005/11/hash_survey.pdf>. Last access: February 05, 2017.

[6] Vidali, Janos, Nose, Peter, Pasalic Enes. **Collisions for variants of the BLAKE hash function**. 2010. Disponível em: <http://lkrv.fri.uni-lj.si/~janos/blake/collisions.pdf>. Last access: 14 February 2020.

[7] Stevens, marc martinus Jacobus. **Attacks on Hash Functions and Applications**. 2012. Available at: <https://www.cwi.nl/system/files/PhD-Thesis-Marc-Stevens-Attacks-on-Hash-Functions-and-Applications.pdf>. Last access: 14 February 2020.

[8] Wang, Xiaoyun. **Survey of Cryptanalysis on Hash Functions.** Fse 2010. Available at: <http://www.iacr.org/workshops/fse2010/content/slide/The%20Survey%20of%20Cryptanalysis%20on%20Hash%20Functions.pdf>. Last access: February 05, 2017.

[9] Kelsey, John; Schneier, Bruce. **Second Preimages on n-bit Hash Functions for much Less than 2^n Work.** Available at: <https://www.schneier.com/academic/paperfiles/paper-preimages.pdf>. Last access: February 05, 2017.

[10] Lathrop, Joel. **Cube attacks on cryptographic hash functions**. Rochester Institute of Tecnology, 2009. Available at: <http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1653&context=theses>. Last access: February 05, 2017.

[11] Lai, Xuejia; Knudsen, Lars R. **Attacks on Double Block Length Hash Functions**. Available at: <https://link.springer.com/content/pdf/10.1007/3-540-58108-1_19.pdf>. Last access: 14 February 2020.

[12] Bouillaguet, Charles; Fouque, Pierre-Alain. **Practical Hash Functions Constructions Resistant to Generic Second Preimage Attacks Beyond the Birthday Bound.** Available at: <https://www.researchgate.net/publication/228575523_Practical_hash_functions_constructions_resistant_to_generic_second_preimage_attacks_beyond_the_birthday_bound>. Last access: February 05, 2017.

[13] Olivier, Gracielle Forechi. **Estudo e implementação do algoritmo de resumo criptográfico SHA-3.** Universidade de Brasília, Instituto de Ciências Exatas, Departamento de Ciência da Computação, 2013. Available at: <https://bdm.unb.br/bitstream/10483/6597/1/2013_GracielleForechiOlivier.pdf> Last access: 14 February 2020.

[14] Preneel, Bart. **Analysis and Design of Cryptographic Hash Functions**. 2003. Available at: <http://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf>. Last access: 14 February 2020.

ANNEX I - GENERATION OF THE EXCHANGE TABLES

The Viktoria hash algorithm works with an internal state of $256!^2$ which corresponds to a total of approximately 3367 bits. We know that the SHA-3 algorithm works with an internal state of 1600 bits. Our algorithm has expanded this number to a little more than double that observed in the SHA-3 algorithm.

To generate the two interchangeable tables, called here T1 and T2, we performed the following procedure:

1) We created a vector containing the first 690 prime numbers:

```
prime numbers = {2, 3, 5, 7, …, 5171, 5179}
```

2) We put the prime numbers together and form a decimal sequence as follows:

```
23571113171923 ...
```

2) The first 2560 digits of this decimal string can be divided into two sets of 1280 digits, which in turn can be divided into two two two-dimensional vectors with 256 5-digit elements. This procedure generates two numerical sequences defined as follows:

```
S1 = { (0, 23571), (1, 11317), (2, 19232), …,
       (254, 72503), (255, 25212) }

S2 = { (0, 53125), (1, 39254), (2, 32549),  …,
       (254, 35167), (255, 51715) }
```

By sorting the sequences S1 and S2 having as key the second number of each element we will obtain the vectors T1 and T2 presented below:

T1

204	193	96	10	100	208	104	212	109	52	70	95	108	99	103	11
107	98	102	106	118	22	122	111	130	1	154	162	166	115	186	198
119	238	250	123	30	127	216	131	61	135	14	139	143	147	151	112
220	54	155	57	159	60	163	167	63	17	171	69	205	175	7	179
75	183	187	116	191	97	165	2	195	20	90	199	88	203	207	211
133	215	225	224	43	219	79	223	120	227	23	231	235	153	185	239
0	243	247	251	228	26	255	124	29	232	128	121	141	32	13	114
217	12	34	142	18	190	132	33	236	48	35	240	249	136	38	72
84	244	237	25	41	177	4	248	140	44	64	73	144	47	252	148
101	50	197	46	152	53	113	145	82	91	156	56	16	55	160	157
37	59	221	164	6	62	169	209	65	168	229	68	28	172	9	110
189	241	134	158	170	178	182	194	21	202	206	218	226	234	242	254
27	71	253	176	67	76	5	74	125	180	87	49	77	93	184	137
19	85	80	181	8	83	188	105	149	58	86	192	213	40	126	138
146	150	15	174	94	210	214	222	230	24	246	117	3	201	233	245
31	196	36	89	39	42	45	51	66	129	161	92	78	81	200	173

T2

240	49	145	148	52	244	152	193	56	248	41	229	241	156	137	157
165	252	60	6	14	50	66	21	74	77	114	118	160	222	226	234
242	250	97	109	64	164	9	69	149	185	213	68	168	129	3	72
7	172	11	15	19	23	177	27	31	35	39	43	47	176	51	55
76	59	141	63	67	71	2	10	18	26	46	75	80	62	33	79
89	121	106	83	110	126	130	142	146	87	170	174	182	186	190	194
91	206	210	214	218	95	254	99	103	180	107	111	115	84	119	61
123	127	131	135	139	1	205	143	147	151	155	184	53	88	159	101
169	163	167	171	175	81	179	233	183	187	191	188	195	199	203	92
207	211	215	197	253	219	223	227	25	231	192	235	239	243	247	96
251	245	255	196	0	161	100	4	104	45	189	200	73	113	217	37
108	225	8	13	133	181	209	204	112	208	12	93	16	116	212	20
173	120	30	34	42	85	82	153	237	98	102	134	138	150	24	158
162	166	178	202	238	216	124	28	5	220	125	105	32	128	224	57
132	36	65	17	40	228	136	29	201	22	38	140	54	58	70	78
86	90	94	232	122	154	198	230	246	44	236	117	144	221	249	48

ANNEX II - INITIALIZATION OF EXCHANGE TABLES

The Viktoria hash algorithm works with two 256 element interchange tables. These tables are generated through the process of generating the interchange tables (see Annex I). However, to promote a quick differentiation between similar messages these tables are started with an order that depends on the complete content of the message.

This process is done as follows:

1) We start two variables:

```
accumulator = 0
control = 0
```

2) In this step we will start reading the message. If the byte of the file is in an odd position (1st, 3rd, 5th, ...) we perform the following operation:

```
readByte = Read a byte of the message
exchange = T2[readByte]
tmp = T1[readByte]
position = (change + control) MOD 256
T1[readByte] = T1[position]
T1[position] = tmp
accumulator = (accumulator + T1[T2[readByte]])
```

3) If the byte of the file is in an even position (2nd, 4th, 6th, ...) then we execute this operation:

```
readByte = Read a byte of the message
exchange = T1[readByte]
tmp = T2[readByte]
position = (change + control) MOD 256
T2[readByte] = T2[position]
T2[position] = tmp
accumulator = (accumulator + T1[T2[readByte]])
```

4) This code is executed until the end of the message. Then we change each element of the vectors **T1** and **T2** according to the code:

```
tmp1 = DIV accumulator 256
tmp2 = MOD 256 accumulator
FOR counter 0 TO 255 DO:
    T1[counter] = (T1[counter] + tmp1) MOD 256
    T2[counter] = (T2[counter] + tmp2) MOD 256
```

Note that the abbreviation **DIV** represents the entire result of a division and the abbreviation **MOD** represents the rest of a division. This algorithm allows changing the order of the **T1** and **T2** vectors according to the content of the incoming message. This way similar files will already start the processing of the 2nd phase of the algorithm with very different parameters which will contribute to make similar messages start the generation of the hash with different content.

ANNEX III - HEADING AND ZERO BYTE CONTROL

A) Generating a header for the inbound message

It is common to exist in the functions of a block to control the size of the input file. In the case of the Viktoria hash function the initial header will be a 64 byte block with the following structure:

```
1st byte: 255
2nd byte: File size in 64 module
3rd byte: Amount of surplus bits in the message
4th byte: Byte representing the bits in excess
5th to 64th byte: bits representing the file size in bytes
```

The file size is represented by a variable of 480 bits. In this example implementation we represent the file size in the last 64 bits[5] through the following polynomial: $\mathbf{ax^7 + bx^6 + cx^5 + dx^4 + ex^3 + fx^2 + gx^1 + hx^0}$. The coefficients **a - h** are in the [0,255] range. Each term of this polynomial occupies a position starting at the 57th byte and ending at the 64th byte of the file header block. All files must have a 64 byte header in this format.

B) Control for files with a size other than a multiple of 64

This control block will consist of 64 bytes with value zero for files with multiple sizes of 64. In the case of a file with a size other than a multiple of 64 we will have to read a number of bytes from the file that corresponds to its size in 64 module. These will be the first bytes of the block and the others will have a value of zero. For example, a file with 100 bytes will have this block filled with the first 36 bytes and the other 28 equal to zero, except the last byte of this block which will be filled with the number of bytes read.

```
if (file size% 64! = 0) {
        quant_bytes = (file size% 64);
        size = size - quant_bytes; // Recalculating file size

        for (ct = 0; ct <quant_bytes; ct ++) {
              fread (& read_block, sizeof (read_block), 1, p1);
              BLOCK_TMP [ct] = T1 [read_block];
        }
}

BLOCK_TMP [63] = (64 - (file size% 64))% 64; // Number of null bytes considered

for (ct = 0; ct <64; ct ++) {
        BLOCK [ct] = BLOCK [ct] ^ BLOCK_TMP [ct]; // XOR with previous BLOCK
}
```

The BLOCK vector represents the result of file header processing. This way, before processing the content of the file itself we have to process these 2 blocks of 512 bits.

We form an initial block of 512 bits called file header and we process this block through several operations. Then we form the 2nd block of the file according to the previous pseudocode. We will also process this block but first we will make a XOR operation with the previous block obtained through the header processing.

The main routine that does the processing of each block will be divided in 3 parts: read_block() function, mixword() function and permutation_block() function. After the last block the mixword_final() functions will be executed and finalized().

[5] Just to avoid having to incorporate into the source code a library to work with large numbers. The main difference from other hash functions like SHA-256 is that the file size is given in bytes and not bits. The Viktoria hash function supports inputs up to 2^{480} -1 bytes.

ANNEX IV - DETAILS OF THE MIXWORD_FINAL() FUNCTION

This routine is very similar to the mix_word() function except that it is executed at least 8192 times and at most 16382 times per file. It is the penultimate operation to be performed before ending with hash output. To calculate how many times this routine will be executed we need all the data from the block processed up to this point:

```
limit = 0
For ct from 1 to 64 do:
    limit = limit XOR T1[BLOCK[ct]]
    limit = limit + T2[BLOCK[ct]]
    limit = (limite + ( (T1[BLOCK[ct]]+1) * (T2[BLOCK[ct]]+1) )) MOD 8191

limit = 8192 + limit
```

The number 8191 was chosen because it is the closest cousin to 8192. This routine is almost identical to the mixword() function. The differences are these:

a) before the base word mixing routine we execute the code:

```
for ct from 1 to 256 do:
    T1[ct] = (T1[ct] + T2[round MOD 256]) MOD 256

for ct from 1 to 256 do:
    T2[ct] = (T2[ct] + T1[(round + 128) MOD 256]) MOD 256
```

This routine allows you to change all the elements of the T1 and T2 exchange tables more quickly by moving the elements of one table based on an element of the other table. The variable "round" refers to the current round of the function.

b) The word mixing routine is performed 4 times per round and uses a dynamic permutation:

```
for (ct=0;ct<4;ct++){
      p0 ^= (ROTL32(~p1,13) ^ ROTL32(p2,3)) + ROTL32(~p3,27);
      p1 += (ROTL32(p0,14) ^ ROTL32(~p2,11)) + ROTL32(p3,26);
      p2 ^= (ROTL32(~p0,9) ^ ROTL32(p1,20)) + ROTL32(~p3,28);
      p3 += (ROTL32(p0,17) ^ ROTL32(~p1,2)) + ROTL32(p2,1);

      p0 ^= (ROTL32(~p1,25) ^ ROTL32(p2,7)) + ROTL32(~p3,18);
      p1 += (ROTL32(p0,10) ^ ROTL32(~p2,8)) + ROTL32(p3,23);
      p2 ^= (ROTL32(~p0,15) ^ ROTL32(p1,31)) + ROTL32(~p3,29);
      p3 += (ROTL32(p0,30) ^ ROTL32(~p1,16)) + ROTL32(p2,21);

      p0 ^= (ROTL32(~p1,19) ^ ROTL32(p2,24)) + ROTL32(~p3,12);
      p1 += (ROTL32(p0,22) ^ ROTL32(~p2,4)) + ROTL32(p3,6);
      p2 ^= (ROTL32(~p0,5) ^ ROTL32(p1,8)) + ROTL32(~p3,13);
      p3 += (ROTL32(p0,14) ^ ROTL32(~p1,24)) + ROTL32(p2,20);

      In this part apply all the permutations resulting from the combinations of
      the prime numbers up to 31 (are 7920 combinations)

      tmp = (p0 % 7920);
      p0 = ~(ROTL32(p0,PERMUTATION[tmp][0]));
      p1 = ROTL32(p1,PERMUTATION[tmp][1]);
      p2 = ~(ROTL32(p2,PERMUTATION[tmp][2]));
      p3 = ROTL32(p3,PERMUTATION[tmp][3]);

      tmp = p0;
      p0 = p1;
```

```
        p1 = p2;
        p2 = p3;
        p3 = tmp;
}
```

The PERMUTATION matrix has 7920 x 4 elements and represents all possible permutations with 4 elements that can be obtained using the prime numbers from 2 to 31.

c) Before the function rotates_blocks() a binary permutation is performed on the first 128-bit subblock (A).

```
position=1
for ct from 1 to 256 do:
    If T1[ct] <= 128
        vector2[position] = vector[T1[ct]]
        position = position + 1
```

Considering vector[] with 128 binary elements we have that vector2[] is reordered from the dynamic table T1.

d) After the function rotates_block() every 16 laps the operation exchanges_block().

e) Finally, every 64 turns, a binary permutation of 512 bits is made with the exchange tables T1 and T2 as reordering parameter. The algorithm is similar to the one executed in item (c).

ANNEX V - DIEHARD TESTS

The description of the Diehard tests has been copied from the official documentation of this battery of tests to pseudo-random numbers.

This is the BIRTHDAY SPACINGS TEST

Choose m birthdays in a year of n days. List the spacings between the birthdays. If j is the number of values that occur more than once in that list, then j is asymptotically Poisson distributed with mean m^3/(4n). Experience shows n must be quite large, say n>=2^18, for comparing the results to the Poisson distribution with that mean. This test uses n=2^24 and m=2^9, so that the underlying distribution for j is taken to be Poisson with lambda=2^27/(2^26)=2. A sample of 500 j's is taken, and a chi-square goodness of fit test provides a p value. The first test uses bits 1-24 (counting from the left) from integers in the specified file.

Then the file is closed and reopened. Next, bits 2-25 are used to provide birthdays, then 3-26 and so on to bits 9-32. Each set of bits provides a p-value, and the nine p-values provide a sample for a KSTEST.

THE OVERLAPPING 5-PERMUTATION TEST

This is the OPERM5 test. It looks at a sequence of one million 32-bit random integers. Each set of five consecutive integers can be in one of 120 states, for the 5! possible orderings of five numbers. Thus the 5th, 6th, 7th,...numbers each provide a state. As many thousands of state transitions are observed, cumulative counts are made of the number of occurences of each state. Then the quadratic form in the weak inverse of the 120x120 covariance matrix yields a test equivalent to the likelihood ratio test that the 120 cell counts came from the specified (asymptotically) normal distribution with the specified 120x120 covariance matrix (with rank 99). This version uses 1,000,000 integers, twice.

This is the BINARY RANK TEST for 31x31 matrices. The leftmost 31 bits of 31 random integers from the test sequence are used to form a 31x31 binary matrix over the field {0,1}. The rank is determined. That rank can be from 0 to 31, but ranks< 28 are rare, and their counts are pooled with those for rank 28. Ranks are found for 40,000 such random matrices and a chisquare test is performed on counts for ranks 31,30,29 and <=28.

This is the BINARY RANK TEST for 32x32 matrices. A random 32x 32 binary matrix is formed, each row a 32-bit random integer. The rank is determined. That rank can be from 0 to 32, ranks less than 29 are rare, and their counts are pooled with those for rank 29. Ranks are found for 40,000 such random matrices and a chisquare test is performed on counts for ranks 32,31, 30 and <=29.

This is the BINARY RANK TEST for 6x8 matrices. From each of six random 32-bit integers from the generator under test, a specified byte is chosen, and the resulting six bytes form a 6x8 binary matrix whose rank is determined. That rank can be from 0 to 6, but ranks 0,1,2,3 are rare; their counts are pooled with those for rank 4. Ranks are found for 100,000 random matrices, and a chi-square test is performed on counts for ranks 6,5 and <=4.

THE BITSTREAM TEST

The file under test is viewed as a stream of bits. Call them b1,b2,... . Consider an alphabet with two "letters", 0 and 1 and think of the stream of bits as a succession of 20-letter "words", overlapping. Thus the first word is b1b2...b20, the second is b2b3...b21, and so on. The bitstream test counts the number of missing 20-letter (20-bit) words in a string of 2^21 overlapping 20-letter words. There are 2^20 possible 20 letter words. For a truly random string of 2^21+19 bits, the number of missing words j should be (very close to) normally distributed with mean 141,909 and sigma 428. Thus (j-141909)/428 should be a standard normal variate (z score) that leads to a uniform [0,1) p value. The test is repeated twenty times.

The tests OPSO, OQSO and DNA

OPSO means Overlapping-Pairs-Sparse-Occupancy

The OPSO test considers 2-letter words from an alphabet of 1024 letters. Each letter is determined by a specified ten bits from a 32-bit integer in the sequence to be tested. OPSO generates 2^21 (overlapping) 2-letter words (from 2^21+1 "keystrokes") and counts the number of missing words---that is 2-letter words which do not appear in the entire sequence. That count should be very close to normally distributed with mean 141,909, sigma 290. Thus (missingwrds-141909)/290 should be a standard normal variable. The OPSO test takes 32 bits at a time from the test file and uses a designated set of ten consecutive bits. It then restarts the file for the next designated 10 bits, and so on.

OQSO means Overlapping-Quadruples-Sparse-Occupancy

The test OQSO is similar, except that it considers 4-letter words from an alphabet of 32 letters, each letter determined by a designated string of 5 consecutive bits from the test file, elements of which are assumed 32-bit random integers. The mean number of missing words in a sequence of 2^21 four-letter words, (2^21+3 "keystrokes"), is again 141909, with sigma = 295. The mean is based on theory; sigma comes from extensive simulation.

This is the COUNT-THE-1's TEST for specific bytes.

Consider the file under test as a stream of 32-bit integers. From each integer, a specific byte is chosen , say the leftmost bits 1 to 8. Each byte can contain from 0 to 8 1's, with probabilitie 1,8,28,56,70,56,28,8,1 over 256. Now let the specified bytes from successive integers provide a string of (overlapping) 5-letter words, each "letter" taking values A,B,C,D,E. The letters are determined by the number of 1's, in that byte 0,1,or 2 ---> A, 3 ---> B, 4 ---> C, 5 ---> D, and 6,7 or 8 ---> E. Thus we have a monkey at a typewriter hitting five keys with with various probabilities 37,56,70, 56,37 over 256. There are 5^5 possible 5-letter words, and from a string of 256,000 (overlapping) 5-letter words, counts are made on the frequencies for each word. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a chisquare test Q5-Q4, the difference of the naive Pearson sums of (OBS-EXP)^2/EXP on counts for 5- and 4-letter cell counts.

THIS IS A PARKING LOT TEST

In a square of side 100, randomly "park" a car---a circle of radius 1. Then try to park a 2nd, a 3rd, and so on, each time parking "by ear". That is, if an attempt to park a car causes a crash with one already parked, try again at a new random location. (To avoid path problems, consider parking helicopters rather than cars.) Each attempt leads to either a crash or a success, the latter followed by an increment to the list of cars already parked. If we plot n the number of attempts, versus k the number successfully parked, we get a curve that should be similar to those provided by a perfect random number generator. Theory for the behavior of such a random curve seems beyond reach, and as graphics displays are not available for this battery of tests, a simple characterization of the random experiment is used k, the number of cars successfully parked after n=12,000 attempts. Simulation shows that k should average 3523 with sigma 21.9 and is very close to normally distributed. Thus (k-3523)/21.9 should be a standard normal variable, which, converted to a uniform variable, provides input to a KSTEST based on a sample of 10.

THE MINIMUM DISTANCE TEST

It does this 100 times choose n=8000 random points in a square of side 10000. Find d, the minimum distance between the (n^2-n)/2 pairs of points. If the points are truly independent uniform, then d^2, the square of the minimum distance should be (very close to) exponentially distributed with mean .995 . Thus 1-exp(-d^2/.995) should be uniform on [0,1) and a KSTEST on the resulting 100 values serves as a test of uniformity for random points in the square. Test numbers=0 mod 5 are printed but the KSTEST is based on the full set of 100 random choices of 8000 points in the 10000x10000 square.

THE 3DSPHERES TEST

Choose 4000 random points in a cube of edge 1000. At each point, center a sphere large enough to reach the next closest point. Then the volume of the smallest such sphere is (very close to) exponentially distributed with mean 120pi/3. Thus the radius cubed is exponential with mean 30. (The mean is obtained by extensive simulation). The 3DSPHERES test generates 4000 such spheres 20 times. Each min radius cubed leads to a uniform variable by means of 1-exp(-r^3/30.), then a KSTEST is done on the 20 p-values.

This is the SQEEZE test

Random integers are floated to get uniforms on [0,1). Starting with k=2^31=2147483647, the test finds j, the number of iterations necessary to reduce k to 1, using the reduction k=ceiling(k*U), with U provided by floating integers from the file being tested. Such j's are found 100,000 times, then counts for the number of times j was <=6,7,...,47,>=48 are used to provide a chi-square test for cell frequencies.

The OVERLAPPING SUMS test

Integers are floated to get a sequence U(1),U(2),... of uniform [0,1) variables. Then overlapping sums, S(1)=U(1)+...+U(100), S2=U(2)+...+U(101),... are formed. The S's are virtually normal with a certain covariance matrix. A linear transformation of the S's converts them to a sequence of independent standard normals, which are converted to uniform variables for a KSTEST. The p-values from ten KSTESTs are given still another KSTEST.

This is the RUNS test. It counts runs up, and runs down, in a sequence of uniform [0,1) variables, obtained by floating the 32-bit integers in the specified file. This example shows how runs are counted .123,.357,.789,.425,.224,.416,.95 contains an up-run of length 3, a down-run of length 2 and an up-run of (at least) 2, depending on the next values. The covariance matrices for the runs-up and runs-down are well known, leading to chisquare tests for quadratic forms in the weak inverses of the covariance matrices. Runs are counted for sequences of length 10,000. This is done ten times. Then repeated.

The DNA test considers an alphabet of 4 letters C,G,A,T, determined by two designated bits in the sequence of random integers being tested. It considers 10-letter words, so that as in OPSO and OQSO, there are 2^20 possible words, and the mean number of missing words from a string of 2^21 (overlapping) 10-letter words (2^21+9 "keystrokes") is 141909. The standard deviation sigma=339 was determined as for OQSO by simulation. (Sigma for OPSO, 290, is the true value (to three places), not determined by simulation.

This is the COUNT-THE-1's TEST on a stream of bytes. Consider the file under test as a stream of bytes (four per 32 bit integer). Each byte can contain from 0 to 8 1's, with probabilities 1,8,28,56,70,56,28,8,1 over 256. Now let the stream of bytes provide a string of overlapping 5-letter words, each "letter" taking values A,B,C,D,E. The letters are determined by the number of 1's in a byte 0,1,or 2 yield A, 3 yields B, 4 yields C, 5 yields D and 6,7 or 8 yield E. Thus we have a monkey at a typewriter hitting five keys with various probabilities (37,56,70,56,37 over 256). There are 5^5 possible 5-letter words, and from a string of 256,000 (overlapping) 5-letter words, counts are made on the frequencies for each word. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a chisquare test Q5-Q4, the difference of the naive Pearson sums of (OBS-EXP)^2/EXP on counts for 5- and 4-letter cell counts.

This is the CRAPS TEST. It plays 200,000 games of craps, finds the number of wins and the number of throws necessary to end each game. The number of wins should be (very close to) a normal with mean 200000p and variance 200000p(1-p), with p=244/495. Throws necessary to complete the game can vary from 1 to infinity, but counts for all>21 are lumped with 21. A chi-square test is made on the no.-of-throws cell counts. Each 32-bit integer from the test file provides the value for the throw of a die, by floating to [0,1), multiplying by 6 and taking 1 plus the integer part of the result.

NOTE Most of the tests in DIEHARD return a p-value, which should be uniform on [0,1) if the input file contains truly independent random bits. Those p-values are obtained by p=F(X), where F is the assumed distribution of the sample random variable X---often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus you should not be surprised with occasional p-values near 0 or 1, such as .0012 or .9983. When a bit stream really FAILS BIG, you will get p's of 0 or 1 to six or more places. By all means, do not, as a Statistician might, think that a p < .025 or p> .975 means that the RNG has "failed the test at the .05 level". Such p's happen among the hundreds that DIEHARD produces, even with good RNG's. So keep in mind that " p happens".

ANNEX VI - DIEHARDER TESTS FOR MIXWORD FUNCTION

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                       saida.bin|  4.17e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.97248073|  PASSED
      diehard_operm5|   0|   1000000|     100|0.94376408|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.81385689|  PASSED
# The file file_input_raw was rewound 1 times
    diehard_rank_6x8|   0|    100000|     100|0.72690594|  PASSED
# The file file_input_raw was rewound 1 times
   diehard_bitstream|   0|   2097152|     100|0.20706381|  PASSED
# The file file_input_raw was rewound 2 times
        diehard_opso|   0|   2097152|     100|0.32838701|  PASSED
# The file file_input_raw was rewound 2 times
        diehard_oqso|   0|   2097152|     100|0.67803575|  PASSED
# The file file_input_raw was rewound 2 times
         diehard_dna|   0|   2097152|     100|0.45293128|  PASSED
# The file file_input_raw was rewound 2 times
diehard_count_1s_str|   0|    256000|     100|0.37148978|  PASSED
# The file file_input_raw was rewound 3 times
diehard_count_1s_byt|   0|    256000|     100|0.87959823|  PASSED
# The file file_input_raw was rewound 3 times
 diehard_parking_lot|   0|     12000|     100|0.39087401|  PASSED
# The file file_input_raw was rewound 3 times
    diehard_2dsphere|   2|      8000|     100|0.53347123|  PASSED
# The file file_input_raw was rewound 3 times
    diehard_3dsphere|   3|      4000|     100|0.28537603|  PASSED
# The file file_input_raw was rewound 4 times
     diehard_squeeze|   0|    100000|     100|0.09544435|  PASSED
# The file file_input_raw was rewound 4 times
        diehard_sums|   0|       100|     100|0.56057958|  PASSED
# The file file_input_raw was rewound 4 times
        diehard_runs|   0|    100000|     100|0.69124466|  PASSED
        diehard_runs|   0|    100000|     100|0.61856359|  PASSED
# The file file_input_raw was rewound 5 times
       diehard_craps|   0|    200000|     100|0.04413745|  PASSED
       diehard_craps|   0|    200000|     100|0.25940083|  PASSED
# The file file_input_raw was rewound 13 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.04248610|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.02639916|  PASSED
# The file file_input_raw was rewound 13 times
         sts_monobit|   1|    100000|     100|0.79946186|  PASSED
# The file file_input_raw was rewound 13 times
            sts_runs|   2|    100000|     100|0.15112950|  PASSED
# The file file_input_raw was rewound 13 times
          sts_serial|   1|    100000|     100|0.61720878|  PASSED
          sts_serial|   2|    100000|     100|0.48522312|  PASSED
          sts_serial|   3|    100000|     100|0.46402745|  PASSED
          sts_serial|   3|    100000|     100|0.03336863|  PASSED
          sts_serial|   4|    100000|     100|0.78946545|  PASSED
          sts_serial|   4|    100000|     100|0.35807014|  PASSED
          sts_serial|   5|    100000|     100|0.62599452|  PASSED
          sts_serial|   5|    100000|     100|0.59366956|  PASSED
          sts_serial|   6|    100000|     100|0.03023550|  PASSED
          sts_serial|   6|    100000|     100|0.44024507|  PASSED
          sts_serial|   7|    100000|     100|0.26579847|  PASSED
          sts_serial|   7|    100000|     100|0.72760722|  PASSED
          sts_serial|   8|    100000|     100|0.87788724|  PASSED
          sts_serial|   8|    100000|     100|0.90439718|  PASSED
          sts_serial|   9|    100000|     100|0.98253761|  PASSED
          sts_serial|   9|    100000|     100|0.50792706|  PASSED
          sts_serial|  10|    100000|     100|0.88545285|  PASSED
          sts_serial|  10|    100000|     100|0.81038050|  PASSED
          sts_serial|  11|    100000|     100|0.96696265|  PASSED
          sts_serial|  11|    100000|     100|0.93897442|  PASSED
          sts_serial|  12|    100000|     100|0.78222224|  PASSED
          sts_serial|  12|    100000|     100|0.89014842|  PASSED
          sts_serial|  13|    100000|     100|0.97445602|  PASSED
          sts_serial|  13|    100000|     100|0.65033521|  PASSED
          sts_serial|  14|    100000|     100|0.94547908|  PASSED
          sts_serial|  14|    100000|     100|0.86955491|  PASSED
          sts_serial|  15|    100000|     100|0.26908769|  PASSED
          sts_serial|  15|    100000|     100|0.47573532|  PASSED
          sts_serial|  16|    100000|     100|0.97486031|  PASSED
          sts_serial|  16|    100000|     100|0.41465364|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   1|    100000|     100|0.67023755|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   2|    100000|     100|0.77100320|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   3|    100000|     100|0.08140147|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   4|    100000|     100|0.94459912|  PASSED
# The file file_input_raw was rewound 14 times
         rgb_bitdist|   5|    100000|     100|0.62094965|  PASSED
# The file file_input_raw was rewound 14 times
         rgb_bitdist|   6|    100000|     100|0.84123364|  PASSED
# The file file_input_raw was rewound 15 times
         rgb_bitdist|   7|    100000|     100|0.65724792|  PASSED
# The file file_input_raw was rewound 16 times
         rgb_bitdist|   8|    100000|     100|0.10337573|  PASSED
# The file file_input_raw was rewound 16 times
         rgb_bitdist|   9|    100000|     100|0.88810706|  PASSED
# The file file_input_raw was rewound 17 times
         rgb_bitdist|  10|    100000|     100|0.88326147|  PASSED
# The file file_input_raw was rewound 18 times
         rgb_bitdist|  11|    100000|     100|0.44445558|  PASSED
# The file file_input_raw was rewound 19 times
         rgb_bitdist|  12|    100000|     100|0.98517330|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   2|     10000|    1000|0.58501561|  PASSED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   3|     10000|    1000|0.87351969|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   4|     10000|    1000|0.20305995|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   5|     10000|    1000|0.06367083|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   2|    100000|     100|0.84149576|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   3|    100000|     100|0.92172787|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   4|    100000|     100|0.79279849|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   5|    100000|     100|0.95356490|  PASSED
# The file file_input_raw was rewound 20 times
      rgb_lagged_sum|   0|   1000000|     100|0.98791671|  PASSED
# The file file_input_raw was rewound 21 times
      rgb_lagged_sum|   1|   1000000|     100|0.08961281|  PASSED
# The file file_input_raw was rewound 22 times
      rgb_lagged_sum|   2|   1000000|     100|0.45905339|  PASSED
# The file file_input_raw was rewound 24 times
      rgb_lagged_sum|   3|   1000000|     100|0.23579093|  PASSED
# The file file_input_raw was rewound 26 times
      rgb_lagged_sum|   4|   1000000|     100|0.10975314|  PASSED
# The file file_input_raw was rewound 28 times
      rgb_lagged_sum|   5|   1000000|     100|0.62083414|  PASSED
# The file file_input_raw was rewound 31 times
      rgb_lagged_sum|   6|   1000000|     100|0.49633980|  PASSED
# The file file_input_raw was rewound 34 times
      rgb_lagged_sum|   7|   1000000|     100|0.03854615|  PASSED
# The file file_input_raw was rewound 38 times
      rgb_lagged_sum|   8|   1000000|     100|0.25264868|  PASSED
# The file file_input_raw was rewound 42 times
      rgb_lagged_sum|   9|   1000000|     100|0.16638465|  PASSED
# The file file_input_raw was rewound 46 times
      rgb_lagged_sum|  10|   1000000|     100|0.04970890|  PASSED
# The file file_input_raw was rewound 51 times
      rgb_lagged_sum|  11|   1000000|     100|0.50742996|  PASSED
# The file file_input_raw was rewound 56 times
      rgb_lagged_sum|  12|   1000000|     100|0.16004200|  PASSED
# The file file_input_raw was rewound 62 times
      rgb_lagged_sum|  13|   1000000|     100|0.72472382|  PASSED
# The file file_input_raw was rewound 68 times
      rgb_lagged_sum|  14|   1000000|     100|0.18347070|  PASSED
# The file file_input_raw was rewound 74 times
      rgb_lagged_sum|  15|   1000000|     100|0.00000006|  FAILED
# The file file_input_raw was rewound 81 times
      rgb_lagged_sum|  16|   1000000|     100|0.25167443|  PASSED
# The file file_input_raw was rewound 88 times
      rgb_lagged_sum|  17|   1000000|     100|0.39577767|  PASSED
# The file file_input_raw was rewound 96 times
      rgb_lagged_sum|  18|   1000000|     100|0.56812816|  PASSED
# The file file_input_raw was rewound 104 times
      rgb_lagged_sum|  19|   1000000|     100|0.09177789|  PASSED
# The file file_input_raw was rewound 112 times
      rgb_lagged_sum|  20|   1000000|     100|0.02153868|  PASSED
# The file file_input_raw was rewound 121 times
      rgb_lagged_sum|  21|   1000000|     100|0.01432522|  PASSED
# The file file_input_raw was rewound 130 times
      rgb_lagged_sum|  22|   1000000|     100|0.03587220|  PASSED
# The file file_input_raw was rewound 140 times
      rgb_lagged_sum|  23|   1000000|     100|0.15328293|  PASSED
# The file file_input_raw was rewound 150 times
      rgb_lagged_sum|  24|   1000000|     100|0.00003341|   WEAK
# The file file_input_raw was rewound 160 times
      rgb_lagged_sum|  25|   1000000|     100|0.13887878|  PASSED
# The file file_input_raw was rewound 171 times
      rgb_lagged_sum|  26|   1000000|     100|0.74023695|  PASSED
# The file file_input_raw was rewound 182 times
      rgb_lagged_sum|  27|   1000000|     100|0.23902428|  PASSED
# The file file_input_raw was rewound 194 times
      rgb_lagged_sum|  28|   1000000|     100|0.03451610|  PASSED
# The file file_input_raw was rewound 206 times
      rgb_lagged_sum|  29|   1000000|     100|0.06331289|  PASSED
# The file file_input_raw was rewound 218 times
      rgb_lagged_sum|  30|   1000000|     100|0.87167906|  PASSED
# The file file_input_raw was rewound 231 times
      rgb_lagged_sum|  31|   1000000|     100|0.07029562|  PASSED
# The file file_input_raw was rewound 244 times
      rgb_lagged_sum|  32|   1000000|     100|0.27068660|  PASSED
# The file file_input_raw was rewound 244 times
     rgb_kstest_test|   0|     10000|    1000|0.68500736|  PASSED
# The file file_input_raw was rewound 245 times
     dab_bytedistrib|   0|  51200000|       1|0.25990456|  PASSED
# The file file_input_raw was rewound 245 times
             dab_dct| 256|     50000|       1|0.15064612|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 246 times
        dab_filltree|  32|  15000000|       1|0.22816946|  PASSED
        dab_filltree|  32|  15000000|       1|0.51825753|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 246 times
       dab_filltree2|   0|   5000000|       1|0.72193593|  PASSED
       dab_filltree2|   1|   5000000|       1|0.33772856|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 246 times
        dab_monobit2|  12|  65000000|       1|0.22925706|  PASSED
```

ANNEX VII - THE 4 ROUND HASH FUNCTION TEST FOR MIXWORD - VERSION 1

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                       saida.bin|  1.82e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.40318857|  PASSED
      diehard_operm5|   0|   1000000|     100|0.74954088|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.27324647|  PASSED
# The file file_input_raw was rewound 1 times
    diehard_rank_6x8|   0|    100000|     100|0.98371456|  PASSED
# The file file_input_raw was rewound 1 times
   diehard_bitstream|   0|   2097152|     100|0.62993976|  PASSED
# The file file_input_raw was rewound 2 times
        diehard_opso|   0|   2097152|     100|0.28342410|  PASSED
# The file file_input_raw was rewound 2 times
        diehard_oqso|   0|   2097152|     100|0.38952479|  PASSED
# The file file_input_raw was rewound 2 times
         diehard_dna|   0|   2097152|     100|0.32747531|  PASSED
# The file file_input_raw was rewound 2 times
diehard_count_1s_str|   0|    256000|     100|0.86979112|  PASSED
# The file file_input_raw was rewound 3 times
diehard_count_1s_byt|   0|    256000|     100|0.49505766|  PASSED
# The file file_input_raw was rewound 3 times
 diehard_parking_lot|   0|     12000|     100|0.51526020|  PASSED
# The file file_input_raw was rewound 3 times
    diehard_2dsphere|   2|      8000|     100|0.36520462|  PASSED
# The file file_input_raw was rewound 3 times
    diehard_3dsphere|   3|      4000|     100|0.09576474|  PASSED
# The file file_input_raw was rewound 4 times
     diehard_squeeze|   0|    100000|     100|0.85116022|  PASSED
# The file file_input_raw was rewound 4 times
        diehard_sums|   0|       100|     100|0.18072296|  PASSED
# The file file_input_raw was rewound 4 times
        diehard_runs|   0|    100000|     100|0.15264676|  PASSED
        diehard_runs|   0|    100000|     100|0.07713804|  PASSED
# The file file_input_raw was rewound 5 times
       diehard_craps|   0|    200000|     100|0.13613602|  PASSED
       diehard_craps|   0|    200000|     100|0.48930166|  PASSED
# The file file_input_raw was rewound 13 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.03883149|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000416|   WEAK
# The file file_input_raw was rewound 13 times
         sts_monobit|   1|    100000|     100|0.78354365|  PASSED
# The file file_input_raw was rewound 13 times
            sts_runs|   2|    100000|     100|0.69576192|  PASSED
# The file file_input_raw was rewound 13 times
          sts_serial|   1|    100000|     100|0.42707376|  PASSED
          sts_serial|   2|    100000|     100|0.14224616|  PASSED
          sts_serial|   3|    100000|     100|0.08089240|  PASSED
          sts_serial|   3|    100000|     100|0.22730704|  PASSED
          sts_serial|   4|    100000|     100|0.40831094|  PASSED
          sts_serial|   4|    100000|     100|0.92648994|  PASSED
          sts_serial|   5|    100000|     100|0.42294912|  PASSED
          sts_serial|   5|    100000|     100|0.97666950|  PASSED
          sts_serial|   6|    100000|     100|0.48916546|  PASSED
          sts_serial|   6|    100000|     100|0.92810987|  PASSED
          sts_serial|   7|    100000|     100|0.56478251|  PASSED
          sts_serial|   7|    100000|     100|0.86957802|  PASSED
          sts_serial|   8|    100000|     100|0.77762109|  PASSED
          sts_serial|   8|    100000|     100|0.80309199|  PASSED
          sts_serial|   9|    100000|     100|0.08429482|  PASSED
          sts_serial|   9|    100000|     100|0.21794320|  PASSED
          sts_serial|  10|    100000|     100|0.61055271|  PASSED
          sts_serial|  10|    100000|     100|0.80953954|  PASSED
          sts_serial|  11|    100000|     100|0.41394867|  PASSED
          sts_serial|  11|    100000|     100|0.78206518|  PASSED
          sts_serial|  12|    100000|     100|0.84862147|  PASSED
          sts_serial|  12|    100000|     100|0.48084107|  PASSED
          sts_serial|  13|    100000|     100|0.07188715|  PASSED
          sts_serial|  13|    100000|     100|0.01973931|  PASSED
          sts_serial|  14|    100000|     100|0.08839256|  PASSED
          sts_serial|  14|    100000|     100|0.90504780|  PASSED
          sts_serial|  15|    100000|     100|0.17122162|  PASSED
          sts_serial|  15|    100000|     100|0.80430063|  PASSED
          sts_serial|  16|    100000|     100|0.91207908|  PASSED
          sts_serial|  16|    100000|     100|0.36033291|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   1|    100000|     100|0.75556615|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   2|    100000|     100|0.09428098|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   3|    100000|     100|0.55912672|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   4|    100000|     100|0.39479104|  PASSED
# The file file_input_raw was rewound 14 times
         rgb_bitdist|   5|    100000|     100|0.52061423|  PASSED
# The file file_input_raw was rewound 14 times
         rgb_bitdist|   6|    100000|     100|0.82671626|  PASSED
# The file file_input_raw was rewound 15 times
         rgb_bitdist|   7|    100000|     100|0.48351521|  PASSED
# The file file_input_raw was rewound 16 times
         rgb_bitdist|   8|    100000|     100|0.84756630|  PASSED
# The file file_input_raw was rewound 16 times
         rgb_bitdist|   9|    100000|     100|0.16580229|  PASSED
# The file file_input_raw was rewound 17 times
         rgb_bitdist|  10|    100000|     100|0.82247514|  PASSED
# The file file_input_raw was rewound 18 times
         rgb_bitdist|  11|    100000|     100|0.46344143|  PASSED
# The file file_input_raw was rewound 19 times
         rgb_bitdist|  12|    100000|     100|0.15356229|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   2|     10000|    1000|0.47383542|  PASSED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   3|     10000|    1000|0.92228379|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   4|     10000|    1000|0.65953551|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   5|     10000|    1000|0.42607677|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   2|    100000|     100|0.17140095|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   3|    100000|     100|0.98952140|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   4|    100000|     100|0.97468608|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   5|    100000|     100|0.92376620|  PASSED
# The file file_input_raw was rewound 20 times
      rgb_lagged_sum|   0|   1000000|     100|0.39013875|  PASSED
# The file file_input_raw was rewound 21 times
      rgb_lagged_sum|   1|   1000000|     100|0.38928700|  PASSED
# The file file_input_raw was rewound 22 times
      rgb_lagged_sum|   2|   1000000|     100|0.60296655|  PASSED
# The file file_input_raw was rewound 24 times
      rgb_lagged_sum|   3|   1000000|     100|0.66129016|  PASSED
# The file file_input_raw was rewound 26 times
      rgb_lagged_sum|   4|   1000000|     100|0.02910539|  PASSED
# The file file_input_raw was rewound 28 times
      rgb_lagged_sum|   5|   1000000|     100|0.02728181|  PASSED
# The file file_input_raw was rewound 31 times
      rgb_lagged_sum|   6|   1000000|     100|0.96182412|  PASSED
# The file file_input_raw was rewound 34 times
      rgb_lagged_sum|   7|   1000000|     100|0.76541880|  PASSED
# The file file_input_raw was rewound 38 times
      rgb_lagged_sum|   8|   1000000|     100|0.52733518|  PASSED
# The file file_input_raw was rewound 42 times
      rgb_lagged_sum|   9|   1000000|     100|0.00031215|   WEAK
# The file file_input_raw was rewound 46 times
      rgb_lagged_sum|  10|   1000000|     100|0.63542382|  PASSED
# The file file_input_raw was rewound 51 times
      rgb_lagged_sum|  11|   1000000|     100|0.00313570|   WEAK
# The file file_input_raw was rewound 56 times
      rgb_lagged_sum|  12|   1000000|     100|0.28479648|  PASSED
# The file file_input_raw was rewound 62 times
      rgb_lagged_sum|  13|   1000000|     100|0.26712807|  PASSED
# The file file_input_raw was rewound 68 times
      rgb_lagged_sum|  14|   1000000|     100|0.06739717|  PASSED
# The file file_input_raw was rewound 74 times
      rgb_lagged_sum|  15|   1000000|     100|0.61349737|  PASSED
# The file file_input_raw was rewound 81 times
      rgb_lagged_sum|  16|   1000000|     100|0.75906964|  PASSED
# The file file_input_raw was rewound 88 times
      rgb_lagged_sum|  17|   1000000|     100|0.11235164|  PASSED
# The file file_input_raw was rewound 96 times
      rgb_lagged_sum|  18|   1000000|     100|0.05896800|  PASSED
# The file file_input_raw was rewound 104 times
      rgb_lagged_sum|  19|   1000000|     100|0.00311817|   WEAK
# The file file_input_raw was rewound 112 times
      rgb_lagged_sum|  20|   1000000|     100|0.53707102|  PASSED
# The file file_input_raw was rewound 121 times
      rgb_lagged_sum|  21|   1000000|     100|0.39907193|  PASSED
# The file file_input_raw was rewound 130 times
      rgb_lagged_sum|  22|   1000000|     100|0.82699571|  PASSED
# The file file_input_raw was rewound 140 times
      rgb_lagged_sum|  23|   1000000|     100|0.68580867|  PASSED
# The file file_input_raw was rewound 150 times
      rgb_lagged_sum|  24|   1000000|     100|0.01247876|  PASSED
# The file file_input_raw was rewound 160 times
      rgb_lagged_sum|  25|   1000000|     100|0.10126232|  PASSED
# The file file_input_raw was rewound 171 times
      rgb_lagged_sum|  26|   1000000|     100|0.92013302|  PASSED
# The file file_input_raw was rewound 182 times
      rgb_lagged_sum|  27|   1000000|     100|0.02970593|  PASSED
# The file file_input_raw was rewound 194 times
      rgb_lagged_sum|  28|   1000000|     100|0.69833167|  PASSED
# The file file_input_raw was rewound 206 times
      rgb_lagged_sum|  29|   1000000|     100|0.00011186|   WEAK
# The file file_input_raw was rewound 218 times
      rgb_lagged_sum|  30|   1000000|     100|0.64511468|  PASSED
# The file file_input_raw was rewound 231 times
      rgb_lagged_sum|  31|   1000000|     100|0.00317694|   WEAK
# The file file_input_raw was rewound 244 times
      rgb_lagged_sum|  32|   1000000|     100|0.42082228|  PASSED
# The file file_input_raw was rewound 244 times
     rgb_kstest_test|   0|     10000|    1000|0.27418082|  PASSED
# The file file_input_raw was rewound 245 times
     dab_bytedistrib|   0|  51200000|       1|0.12726248|  PASSED
# The file file_input_raw was rewound 245 times
             dab_dct| 256|     50000|       1|0.82453388|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 246 times
        dab_filltree|  32|  15000000|       1|0.89886473|  PASSED
        dab_filltree|  32|  15000000|       1|0.65607642|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 246 times
       dab_filltree2|   0|   5000000|       1|0.10798413|  PASSED
       dab_filltree2|   1|   5000000|       1|0.72497860|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 246 times
        dab_monobit2|  12|  65000000|       1|0.56005539|  PASSED
```

ANNEX VIII - THE 4 ROUND HASH FUNCTION TEST FOR MIXWORD - VERSION 2

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                       saida.bin|  4.33e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.73346478|  PASSED
      diehard_operm5|   0|   1000000|     100|0.86829659|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.98183373|  PASSED
# The file file_input_raw was rewound 1 times
    diehard_rank_6x8|   0|    100000|     100|0.29051708|  PASSED
# The file file_input_raw was rewound 1 times
   diehard_bitstream|   0|   2097152|     100|0.37604905|  PASSED
# The file file_input_raw was rewound 2 times
        diehard_opso|   0|   2097152|     100|0.77228097|  PASSED
# The file file_input_raw was rewound 2 times
        diehard_oqso|   0|   2097152|     100|0.89743726|  PASSED
# The file file_input_raw was rewound 2 times
         diehard_dna|   0|   2097152|     100|0.97448002|  PASSED
# The file file_input_raw was rewound 2 times
diehard_count_1s_str|   0|    256000|     100|0.42466440|  PASSED
# The file file_input_raw was rewound 3 times
diehard_count_1s_byt|   0|    256000|     100|0.03289935|  PASSED
# The file file_input_raw was rewound 3 times
 diehard_parking_lot|   0|     12000|     100|0.67934334|  PASSED
# The file file_input_raw was rewound 3 times
    diehard_2dsphere|   2|      8000|     100|0.17337963|  PASSED
# The file file_input_raw was rewound 3 times
    diehard_3dsphere|   3|      4000|     100|0.35817494|  PASSED
# The file file_input_raw was rewound 4 times
     diehard_squeeze|   0|    100000|     100|0.44303776|  PASSED
# The file file_input_raw was rewound 4 times
        diehard_sums|   0|       100|     100|0.04588441|  PASSED
# The file file_input_raw was rewound 4 times
        diehard_runs|   0|    100000|     100|0.30428349|  PASSED
        diehard_runs|   0|    100000|     100|0.34979627|  PASSED
# The file file_input_raw was rewound 5 times
       diehard_craps|   0|    200000|     100|0.88073477|  PASSED
       diehard_craps|   0|    200000|     100|0.48259402|  PASSED
# The file file_input_raw was rewound 13 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.05337985|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000092|  FAILED
# The file file_input_raw was rewound 13 times
         sts_monobit|   1|    100000|     100|0.04597422|  PASSED
# The file file_input_raw was rewound 13 times
            sts_runs|   2|    100000|     100|0.87341704|  PASSED
# The file file_input_raw was rewound 13 times
          sts_serial|   1|    100000|     100|0.51474348|  PASSED
          sts_serial|   2|    100000|     100|0.55960177|  PASSED
          sts_serial|   3|    100000|     100|0.46656924|  PASSED
          sts_serial|   3|    100000|     100|0.47673289|  PASSED
          sts_serial|   4|    100000|     100|0.62902694|  PASSED
          sts_serial|   4|    100000|     100|0.49827090|  PASSED
          sts_serial|   5|    100000|     100|0.81056717|  PASSED
          sts_serial|   5|    100000|     100|0.94410878|  PASSED
          sts_serial|   6|    100000|     100|0.83400816|  PASSED
          sts_serial|   6|    100000|     100|0.94163400|  PASSED
          sts_serial|   7|    100000|     100|0.18959675|  PASSED
          sts_serial|   7|    100000|     100|0.01130727|  PASSED
          sts_serial|   8|    100000|     100|0.01948640|  PASSED
          sts_serial|   8|    100000|     100|0.03847916|  PASSED
          sts_serial|   9|    100000|     100|0.26320858|  PASSED
          sts_serial|   9|    100000|     100|0.84573524|  PASSED
          sts_serial|  10|    100000|     100|0.16689686|  PASSED
          sts_serial|  10|    100000|     100|0.99884626|   WEAK
          sts_serial|  11|    100000|     100|0.33625668|  PASSED
          sts_serial|  11|    100000|     100|0.90116610|  PASSED
          sts_serial|  12|    100000|     100|0.39664647|  PASSED
          sts_serial|  12|    100000|     100|0.86234781|  PASSED
          sts_serial|  13|    100000|     100|0.71823004|  PASSED
          sts_serial|  13|    100000|     100|0.56993416|  PASSED
          sts_serial|  14|    100000|     100|0.47011130|  PASSED
          sts_serial|  14|    100000|     100|0.65938272|  PASSED
          sts_serial|  15|    100000|     100|0.94724880|  PASSED
          sts_serial|  15|    100000|     100|0.51935692|  PASSED
          sts_serial|  16|    100000|     100|0.90125742|  PASSED
          sts_serial|  16|    100000|     100|0.93234111|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   1|    100000|     100|0.42320294|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   2|    100000|     100|0.03311286|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   3|    100000|     100|0.29936929|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   4|    100000|     100|0.77555651|  PASSED
# The file file_input_raw was rewound 14 times
         rgb_bitdist|   5|    100000|     100|0.53810624|  PASSED
# The file file_input_raw was rewound 14 times
         rgb_bitdist|   6|    100000|     100|0.60835561|  PASSED
# The file file_input_raw was rewound 15 times
         rgb_bitdist|   7|    100000|     100|0.64613935|  PASSED
# The file file_input_raw was rewound 16 times
         rgb_bitdist|   8|    100000|     100|0.44266502|  PASSED
# The file file_input_raw was rewound 16 times
         rgb_bitdist|   9|    100000|     100|0.19192416|  PASSED
# The file file_input_raw was rewound 17 times
         rgb_bitdist|  10|    100000|     100|0.60410573|  PASSED
# The file file_input_raw was rewound 18 times
         rgb_bitdist|  11|    100000|     100|0.97011502|  PASSED
# The file file_input_raw was rewound 19 times
         rgb_bitdist|  12|    100000|     100|0.30304835|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   2|     10000|    1000|0.32579545|  PASSED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   3|     10000|    1000|0.69701852|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   4|     10000|    1000|0.67959750|  PASSED
# The file file_input_raw was rewound 19 times
rgb_minimum_distance|   5|     10000|    1000|0.23608891|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   2|    100000|     100|0.03403235|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   3|    100000|     100|0.15783280|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   4|    100000|     100|0.40849598|  PASSED
# The file file_input_raw was rewound 20 times
    rgb_permutations|   5|    100000|     100|0.33820200|  PASSED
# The file file_input_raw was rewound 20 times
      rgb_lagged_sum|   0|   1000000|     100|0.97871570|  PASSED
# The file file_input_raw was rewound 21 times
      rgb_lagged_sum|   1|   1000000|     100|0.82687096|  PASSED
# The file file_input_raw was rewound 22 times
      rgb_lagged_sum|   2|   1000000|     100|0.54435002|  PASSED
# The file file_input_raw was rewound 24 times
      rgb_lagged_sum|   3|   1000000|     100|0.67959914|  PASSED
# The file file_input_raw was rewound 26 times
      rgb_lagged_sum|   4|   1000000|     100|0.39205808|  PASSED
# The file file_input_raw was rewound 28 times
      rgb_lagged_sum|   5|   1000000|     100|0.86971609|  PASSED
# The file file_input_raw was rewound 31 times
      rgb_lagged_sum|   6|   1000000|     100|0.66525076|  PASSED
# The file file_input_raw was rewound 34 times
      rgb_lagged_sum|   7|   1000000|     100|0.35576107|  PASSED
# The file file_input_raw was rewound 38 times
      rgb_lagged_sum|   8|   1000000|     100|0.79039371|  PASSED
# The file file_input_raw was rewound 42 times
      rgb_lagged_sum|   9|   1000000|     100|0.05032103|  PASSED
# The file file_input_raw was rewound 46 times
      rgb_lagged_sum|  10|   1000000|     100|0.85010735|  PASSED
# The file file_input_raw was rewound 51 times
      rgb_lagged_sum|  11|   1000000|     100|0.53388292|  PASSED
# The file file_input_raw was rewound 56 times
      rgb_lagged_sum|  12|   1000000|     100|0.33046241|  PASSED
# The file file_input_raw was rewound 62 times
      rgb_lagged_sum|  13|   1000000|     100|0.50997242|  PASSED
# The file file_input_raw was rewound 68 times
      rgb_lagged_sum|  14|   1000000|     100|0.18410617|  PASSED
# The file file_input_raw was rewound 74 times
      rgb_lagged_sum|  15|   1000000|     100|0.06634640|  PASSED
# The file file_input_raw was rewound 81 times
      rgb_lagged_sum|  16|   1000000|     100|0.25945277|  PASSED
# The file file_input_raw was rewound 88 times
      rgb_lagged_sum|  17|   1000000|     100|0.93604024|  PASSED
# The file file_input_raw was rewound 96 times
      rgb_lagged_sum|  18|   1000000|     100|0.22576527|  PASSED
# The file file_input_raw was rewound 104 times
      rgb_lagged_sum|  19|   1000000|     100|0.04983936|  PASSED
# The file file_input_raw was rewound 112 times
      rgb_lagged_sum|  20|   1000000|     100|0.63685199|  PASSED
# The file file_input_raw was rewound 121 times
      rgb_lagged_sum|  21|   1000000|     100|0.21580123|  PASSED
# The file file_input_raw was rewound 130 times
      rgb_lagged_sum|  22|   1000000|     100|0.83353688|  PASSED
# The file file_input_raw was rewound 140 times
      rgb_lagged_sum|  23|   1000000|     100|0.04717501|  PASSED
# The file file_input_raw was rewound 150 times
      rgb_lagged_sum|  24|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 160 times
      rgb_lagged_sum|  25|   1000000|     100|0.48249520|  PASSED
# The file file_input_raw was rewound 171 times
      rgb_lagged_sum|  26|   1000000|     100|0.57761783|  PASSED
# The file file_input_raw was rewound 182 times
      rgb_lagged_sum|  27|   1000000|     100|0.32261156|  PASSED
# The file file_input_raw was rewound 194 times
      rgb_lagged_sum|  28|   1000000|     100|0.03924419|  PASSED
# The file file_input_raw was rewound 206 times
      rgb_lagged_sum|  29|   1000000|     100|0.07901687|  PASSED
# The file file_input_raw was rewound 218 times
      rgb_lagged_sum|  30|   1000000|     100|0.43276784|  PASSED
# The file file_input_raw was rewound 231 times
      rgb_lagged_sum|  31|   1000000|     100|0.63384537|  PASSED
# The file file_input_raw was rewound 244 times
      rgb_lagged_sum|  32|   1000000|     100|0.46554901|  PASSED
# The file file_input_raw was rewound 244 times
     rgb_kstest_test|   0|     10000|    1000|0.31255804|  PASSED
# The file file_input_raw was rewound 245 times
     dab_bytedistrib|   0|  51200000|       1|0.89872923|  PASSED
# The file file_input_raw was rewound 245 times
             dab_dct| 256|     50000|       1|0.24541445|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 246 times
        dab_filltree|  32|  15000000|       1|0.39234983|  PASSED
        dab_filltree|  32|  15000000|       1|0.90006079|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 246 times
       dab_filltree2|   0|   5000000|       1|0.07547405|  PASSED
       dab_filltree2|   1|   5000000|       1|0.06745464|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 246 times
        dab_monobit2|  12|  65000000|       1|0.80324039|  PASSED
```

ANNEX IX - THE LONG FILE TEST WITH 4 TURNS FOR MIXWORD() SINGLE BLOCKS

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                      saida.bin|  1.09e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.97248073|  PASSED
      diehard_operm5|   0|   1000000|     100|0.94376408|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.81385689|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.00277584|   WEAK
   diehard_bitstream|   0|   2097152|     100|0.64580367|  PASSED
        diehard_opso|   0|   2097152|     100|0.90864018|  PASSED
        diehard_oqso|   0|   2097152|     100|0.97249539|  PASSED
         diehard_dna|   0|   2097152|     100|0.37405698|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.68593649|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.39441806|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.55728091|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.37677002|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.18499521|  PASSED
     diehard_squeeze|   0|    100000|     100|0.07932723|  PASSED
        diehard_sums|   0|       100|     100|0.23578846|  PASSED
        diehard_runs|   0|    100000|     100|0.17336659|  PASSED
        diehard_runs|   0|    100000|     100|0.40320506|  PASSED
       diehard_craps|   0|    200000|     100|0.98835337|  PASSED
       diehard_craps|   0|    200000|     100|0.32989449|  PASSED
# The file file_input_raw was rewound 1 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.99648576|   WEAK
 marsaglia_tsang_gcd|   0|  10000000|     100|0.09933217|  PASSED
# The file file_input_raw was rewound 1 times
         sts_monobit|   1|    100000|     100|0.99819493|   WEAK
# The file file_input_raw was rewound 1 times
            sts_runs|   2|    100000|     100|0.46145112|  PASSED
# The file file_input_raw was rewound 1 times
          sts_serial|   1|    100000|     100|0.61984524|  PASSED
          sts_serial|   2|    100000|     100|0.47299493|  PASSED
          sts_serial|   3|    100000|     100|0.78391153|  PASSED
          sts_serial|   3|    100000|     100|0.89867873|  PASSED
          sts_serial|   4|    100000|     100|0.24845939|  PASSED
          sts_serial|   4|    100000|     100|0.99452630|  PASSED
          sts_serial|   5|    100000|     100|0.50662284|  PASSED
          sts_serial|   5|    100000|     100|0.25591575|  PASSED
          sts_serial|   6|    100000|     100|0.09696611|  PASSED
          sts_serial|   6|    100000|     100|0.80255088|  PASSED
          sts_serial|   7|    100000|     100|0.56273221|  PASSED
          sts_serial|   7|    100000|     100|0.99476624|  PASSED
          sts_serial|   8|    100000|     100|0.72629905|  PASSED
          sts_serial|   8|    100000|     100|0.78756674|  PASSED
          sts_serial|   9|    100000|     100|0.23365137|  PASSED
          sts_serial|   9|    100000|     100|0.36826228|  PASSED
          sts_serial|  10|    100000|     100|0.09189629|  PASSED
          sts_serial|  10|    100000|     100|0.30037156|  PASSED
          sts_serial|  11|    100000|     100|0.60927228|  PASSED
          sts_serial|  11|    100000|     100|0.33332620|  PASSED
          sts_serial|  12|    100000|     100|0.17047296|  PASSED
          sts_serial|  12|    100000|     100|0.45842414|  PASSED
          sts_serial|  13|    100000|     100|0.79249327|  PASSED
          sts_serial|  13|    100000|     100|0.79860635|  PASSED
          sts_serial|  14|    100000|     100|0.95473919|  PASSED
          sts_serial|  14|    100000|     100|0.69948054|  PASSED
          sts_serial|  15|    100000|     100|0.85977779|  PASSED
          sts_serial|  15|    100000|     100|0.88256481|  PASSED
          sts_serial|  16|    100000|     100|0.43303554|  PASSED
          sts_serial|  16|    100000|     100|0.28530998|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   1|    100000|     100|0.89975950|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   2|    100000|     100|0.50881308|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   3|    100000|     100|0.06739926|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   4|    100000|     100|0.64377426|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   5|    100000|     100|0.08745996|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   6|    100000|     100|0.59081655|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   7|    100000|     100|0.42979821|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   8|    100000|     100|0.79315279|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   9|    100000|     100|0.43600283|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|  10|    100000|     100|0.43783641|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|  11|    100000|     100|0.19025658|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|  12|    100000|     100|0.58984736|  PASSED
# The file file_input_raw was rewound 1 times
rgb_minimum_distance|   2|     10000|    1000|0.65125227|  PASSED
# The file file_input_raw was rewound 1 times
rgb_minimum_distance|   3|     10000|    1000|0.29079657|  PASSED
# The file file_input_raw was rewound 1 times
rgb_minimum_distance|   4|     10000|    1000|0.31751725|  PASSED
# The file file_input_raw was rewound 1 times
rgb_minimum_distance|   5|     10000|    1000|0.23107879|  PASSED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
# The file file_input_raw was rewound 2 times
    rgb_permutations|   2|    100000|     100|0.18820488|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   3|    100000|     100|0.48340723|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   4|    100000|     100|0.34892344|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   5|    100000|     100|0.97379306|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   0|   1000000|     100|0.77266935|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   1|   1000000|     100|0.04886623|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   2|   1000000|     100|0.90884490|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   3|   1000000|     100|0.85722620|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   4|   1000000|     100|0.90868686|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   5|   1000000|     100|0.84473037|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   6|   1000000|     100|0.63848252|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   7|   1000000|     100|0.80473259|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   8|   1000000|     100|0.68946906|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   9|   1000000|     100|0.38983317|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  10|   1000000|     100|0.90707756|  PASSED
# The file file_input_raw was rewound 5 times
      rgb_lagged_sum|  11|   1000000|     100|0.19931159|  PASSED
# The file file_input_raw was rewound 5 times
      rgb_lagged_sum|  12|   1000000|     100|0.93636869|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  13|   1000000|     100|0.78243116|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  14|   1000000|     100|0.72748306|  PASSED
# The file file_input_raw was rewound 7 times
      rgb_lagged_sum|  15|   1000000|     100|0.94859096|  PASSED
# The file file_input_raw was rewound 8 times
      rgb_lagged_sum|  16|   1000000|     100|0.48265559|  PASSED
# The file file_input_raw was rewound 8 times
      rgb_lagged_sum|  17|   1000000|     100|0.91623356|  PASSED
# The file file_input_raw was rewound 9 times
      rgb_lagged_sum|  18|   1000000|     100|0.48112595|  PASSED
# The file file_input_raw was rewound 10 times
      rgb_lagged_sum|  19|   1000000|     100|0.34194850|  PASSED
# The file file_input_raw was rewound 11 times
      rgb_lagged_sum|  20|   1000000|     100|0.55054910|  PASSED
# The file file_input_raw was rewound 12 times
      rgb_lagged_sum|  21|   1000000|     100|0.51802810|  PASSED
# The file file_input_raw was rewound 13 times
      rgb_lagged_sum|  22|   1000000|     100|0.93725702|  PASSED
# The file file_input_raw was rewound 14 times
      rgb_lagged_sum|  23|   1000000|     100|0.44262679|  PASSED
# The file file_input_raw was rewound 15 times
      rgb_lagged_sum|  24|   1000000|     100|0.51206678|  PASSED
# The file file_input_raw was rewound 16 times
      rgb_lagged_sum|  25|   1000000|     100|0.96378705|  PASSED
# The file file_input_raw was rewound 17 times
      rgb_lagged_sum|  26|   1000000|     100|0.39532124|  PASSED
# The file file_input_raw was rewound 18 times
      rgb_lagged_sum|  27|   1000000|     100|0.23213363|  PASSED
# The file file_input_raw was rewound 19 times
      rgb_lagged_sum|  28|   1000000|     100|0.29788776|  PASSED
# The file file_input_raw was rewound 20 times
      rgb_lagged_sum|  29|   1000000|     100|0.14875039|  PASSED
# The file file_input_raw was rewound 21 times
      rgb_lagged_sum|  30|   1000000|     100|0.47180920|  PASSED
# The file file_input_raw was rewound 23 times
      rgb_lagged_sum|  31|   1000000|     100|0.26868268|  PASSED
# The file file_input_raw was rewound 24 times
      rgb_lagged_sum|  32|   1000000|     100|0.56076544|  PASSED
# The file file_input_raw was rewound 24 times
     rgb_kstest_test|   0|     10000|    1000|0.76522803|  PASSED
# The file file_input_raw was rewound 24 times
     dab_bytedistrib|   0|  51200000|       1|0.90069188|  PASSED
# The file file_input_raw was rewound 24 times
             dab_dct| 256|     50000|       1|0.79897091|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 24 times
        dab_filltree|  32|  15000000|       1|0.57076352|  PASSED
        dab_filltree|  32|  15000000|       1|0.94863125|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 24 times
       dab_filltree2|   0|   5000000|       1|0.04084003|  PASSED
       dab_filltree2|   1|   5000000|       1|0.38843384|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 24 times
        dab_monobit2|  12|  65000000|       1|0.19762403|  PASSED
```

ANNEX X - THE LONG FILE TEST WITH 4 TURNS FOR MIXWORD() CHAINED BLOCKS

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                       saida.bin|  3.88e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.97216705|  PASSED
      diehard_operm5|   0|   1000000|     100|0.40844114|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.98054019|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.88294556|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.99833870|   WEAK
        diehard_opso|   0|   2097152|     100|0.61331777|  PASSED
        diehard_oqso|   0|   2097152|     100|0.87595263|  PASSED
         diehard_dna|   0|   2097152|     100|0.66725716|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.18735007|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.59154066|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.46927986|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.76155639|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.47244568|  PASSED
     diehard_squeeze|   0|    100000|     100|0.16613889|  PASSED
        diehard_sums|   0|       100|     100|0.59766446|  PASSED
        diehard_runs|   0|    100000|     100|0.89069898|  PASSED
        diehard_runs|   0|    100000|     100|0.15142517|  PASSED
       diehard_craps|   0|    200000|     100|0.96467061|  PASSED
       diehard_craps|   0|    200000|     100|0.21821478|  PASSED
# The file file_input_raw was rewound 1 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.43786010|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.92930533|  PASSED
# The file file_input_raw was rewound 1 times
         sts_monobit|   1|    100000|     100|0.20540769|  PASSED
# The file file_input_raw was rewound 1 times
            sts_runs|   2|    100000|     100|0.42899176|  PASSED
# The file file_input_raw was rewound 1 times
          sts_serial|   1|    100000|     100|0.13293347|  PASSED
          sts_serial|   2|    100000|     100|0.63356384|  PASSED
          sts_serial|   3|    100000|     100|0.21492300|  PASSED
          sts_serial|   3|    100000|     100|0.56579042|  PASSED
          sts_serial|   4|    100000|     100|0.01110879|  PASSED
          sts_serial|   4|    100000|     100|0.21561006|  PASSED
          sts_serial|   5|    100000|     100|0.13757229|  PASSED
          sts_serial|   5|    100000|     100|0.81693926|  PASSED
          sts_serial|   6|    100000|     100|0.96190407|  PASSED
          sts_serial|   6|    100000|     100|0.14308013|  PASSED
          sts_serial|   7|    100000|     100|0.92577477|  PASSED
          sts_serial|   7|    100000|     100|0.60273775|  PASSED
          sts_serial|   8|    100000|     100|0.91178251|  PASSED
          sts_serial|   8|    100000|     100|0.81434519|  PASSED
          sts_serial|   9|    100000|     100|0.83367908|  PASSED
          sts_serial|   9|    100000|     100|0.77057779|  PASSED
          sts_serial|  10|    100000|     100|0.42289704|  PASSED
          sts_serial|  10|    100000|     100|0.64727664|  PASSED
          sts_serial|  11|    100000|     100|0.28252637|  PASSED
          sts_serial|  11|    100000|     100|0.88097098|  PASSED
          sts_serial|  12|    100000|     100|0.90459645|  PASSED
          sts_serial|  12|    100000|     100|0.80019503|  PASSED
          sts_serial|  13|    100000|     100|0.56395702|  PASSED
          sts_serial|  13|    100000|     100|0.46041979|  PASSED
          sts_serial|  14|    100000|     100|0.60889769|  PASSED
          sts_serial|  14|    100000|     100|0.71587550|  PASSED
          sts_serial|  15|    100000|     100|0.89625425|  PASSED
          sts_serial|  15|    100000|     100|0.56302916|  PASSED
          sts_serial|  16|    100000|     100|0.82397317|  PASSED
          sts_serial|  16|    100000|     100|0.13031628|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   1|    100000|     100|0.75113430|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   2|    100000|     100|0.65812494|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   3|    100000|     100|0.42458437|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   4|    100000|     100|0.74953653|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   5|    100000|     100|0.53684025|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   6|    100000|     100|0.70405393|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   7|    100000|     100|0.90716809|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   8|    100000|     100|0.56116053|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   9|    100000|     100|0.75318270|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|  10|    100000|     100|0.93134105|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|  11|    100000|     100|0.82345422|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|  12|    100000|     100|0.94082096|  PASSED
# The file file_input_raw was rewound 1 times
rgb_minimum_distance|   2|     10000|    1000|0.74977779|  PASSED
# The file file_input_raw was rewound 1 times
rgb_minimum_distance|   3|     10000|    1000|0.77143054|  PASSED
# The file file_input_raw was rewound 1 times
rgb_minimum_distance|   4|     10000|    1000|0.25552604|  PASSED
# The file file_input_raw was rewound 1 times
rgb_minimum_distance|   5|     10000|    1000|0.63367849|  PASSED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
# The file file_input_raw was rewound 2 times
    rgb_permutations|   2|    100000|     100|0.24735324|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   3|    100000|     100|0.59312320|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   4|    100000|     100|0.95325992|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   5|    100000|     100|0.76743641|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   0|   1000000|     100|0.13940397|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   1|   1000000|     100|0.46469015|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   2|   1000000|     100|0.50159420|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   3|   1000000|     100|0.09459787|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   4|   1000000|     100|0.38212100|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   5|   1000000|     100|0.25916133|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   6|   1000000|     100|0.91431144|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   7|   1000000|     100|0.70613566|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   8|   1000000|     100|0.10242062|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   9|   1000000|     100|0.84165977|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  10|   1000000|     100|0.74234847|  PASSED
# The file file_input_raw was rewound 5 times
      rgb_lagged_sum|  11|   1000000|     100|0.86095625|  PASSED
# The file file_input_raw was rewound 5 times
      rgb_lagged_sum|  12|   1000000|     100|0.70643695|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  13|   1000000|     100|0.98726943|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  14|   1000000|     100|0.85451298|  PASSED
# The file file_input_raw was rewound 7 times
      rgb_lagged_sum|  15|   1000000|     100|0.96411364|  PASSED
# The file file_input_raw was rewound 8 times
      rgb_lagged_sum|  16|   1000000|     100|0.52446571|  PASSED
# The file file_input_raw was rewound 8 times
      rgb_lagged_sum|  17|   1000000|     100|0.74319876|  PASSED
# The file file_input_raw was rewound 9 times
      rgb_lagged_sum|  18|   1000000|     100|0.67009256|  PASSED
# The file file_input_raw was rewound 10 times
      rgb_lagged_sum|  19|   1000000|     100|0.67874004|  PASSED
# The file file_input_raw was rewound 11 times
      rgb_lagged_sum|  20|   1000000|     100|0.71823762|  PASSED
# The file file_input_raw was rewound 12 times
      rgb_lagged_sum|  21|   1000000|     100|0.16671815|  PASSED
# The file file_input_raw was rewound 13 times
      rgb_lagged_sum|  22|   1000000|     100|0.32558470|  PASSED
# The file file_input_raw was rewound 14 times
      rgb_lagged_sum|  23|   1000000|     100|0.01320589|  PASSED
# The file file_input_raw was rewound 15 times
      rgb_lagged_sum|  24|   1000000|     100|0.33077365|  PASSED
# The file file_input_raw was rewound 16 times
      rgb_lagged_sum|  25|   1000000|     100|0.12394756|  PASSED
# The file file_input_raw was rewound 17 times
      rgb_lagged_sum|  26|   1000000|     100|0.18422830|  PASSED
# The file file_input_raw was rewound 18 times
      rgb_lagged_sum|  27|   1000000|     100|0.84064250|  PASSED
# The file file_input_raw was rewound 19 times
      rgb_lagged_sum|  28|   1000000|     100|0.76122756|  PASSED
# The file file_input_raw was rewound 20 times
      rgb_lagged_sum|  29|   1000000|     100|0.88943617|  PASSED
# The file file_input_raw was rewound 21 times
      rgb_lagged_sum|  30|   1000000|     100|0.76485827|  PASSED
# The file file_input_raw was rewound 23 times
      rgb_lagged_sum|  31|   1000000|     100|0.40750963|  PASSED
# The file file_input_raw was rewound 24 times
      rgb_lagged_sum|  32|   1000000|     100|0.90111994|  PASSED
# The file file_input_raw was rewound 24 times
     rgb_kstest_test|   0|     10000|    1000|0.36627513|  PASSED
# The file file_input_raw was rewound 24 times
     dab_bytedistrib|   0|  51200000|       1|0.44122388|  PASSED
# The file file_input_raw was rewound 24 times
             dab_dct| 256|     50000|       1|0.94312560|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 24 times
        dab_filltree|  32|  15000000|       1|0.38661011|  PASSED
        dab_filltree|  32|  15000000|       1|0.70659322|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 24 times
       dab_filltree2|   0|   5000000|       1|0.71695842|  PASSED
       dab_filltree2|   1|   5000000|       1|0.01485260|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 24 times
        dab_monobit2|  12|  65000000|       1|0.23142330|  PASSED
```

ANNEX XI - THE SUPERLONG TEST FOR MIXWORD()
SINGLE BLOCKS

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                      saida.bin|  1.52e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.95141600|  PASSED
      diehard_operm5|   0|   1000000|     100|0.41243873|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.99823322|   WEAK
    diehard_rank_6x8|   0|    100000|     100|0.75951837|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.81570668|  PASSED
        diehard_opso|   0|   2097152|     100|0.78982768|  PASSED
        diehard_oqso|   0|   2097152|     100|0.55077429|  PASSED
         diehard_dna|   0|   2097152|     100|0.60548275|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.99168366|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.59422292|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.06065386|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.97429927|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.72705305|  PASSED
     diehard_squeeze|   0|    100000|     100|0.73043957|  PASSED
        diehard_sums|   0|       100|     100|0.78742838|  PASSED
        diehard_runs|   0|    100000|     100|0.55123016|  PASSED
        diehard_runs|   0|    100000|     100|0.35328146|  PASSED
       diehard_craps|   0|    200000|     100|0.14618153|  PASSED
       diehard_craps|   0|    200000|     100|0.32194898|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.28937161|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.10468132|  PASSED
         sts_monobit|   1|    100000|     100|0.79308108|  PASSED
            sts_runs|   2|    100000|     100|0.98947598|  PASSED
          sts_serial|   1|    100000|     100|0.70332724|  PASSED
          sts_serial|   2|    100000|     100|0.79738304|  PASSED
          sts_serial|   3|    100000|     100|0.96298288|  PASSED
          sts_serial|   3|    100000|     100|0.95730720|  PASSED
          sts_serial|   4|    100000|     100|0.79196604|  PASSED
          sts_serial|   4|    100000|     100|0.38676235|  PASSED
          sts_serial|   5|    100000|     100|0.41932334|  PASSED
          sts_serial|   5|    100000|     100|0.37853604|  PASSED
          sts_serial|   6|    100000|     100|0.18011685|  PASSED
          sts_serial|   6|    100000|     100|0.95030228|  PASSED
          sts_serial|   7|    100000|     100|0.09211233|  PASSED
          sts_serial|   7|    100000|     100|0.17487153|  PASSED
          sts_serial|   8|    100000|     100|0.08301392|  PASSED
          sts_serial|   8|    100000|     100|0.94786447|  PASSED
          sts_serial|   9|    100000|     100|0.47654142|  PASSED
          sts_serial|   9|    100000|     100|0.83454470|  PASSED
          sts_serial|  10|    100000|     100|0.01856147|  PASSED
          sts_serial|  10|    100000|     100|0.53950231|  PASSED
          sts_serial|  11|    100000|     100|0.55488699|  PASSED
          sts_serial|  11|    100000|     100|0.80878432|  PASSED
          sts_serial|  12|    100000|     100|0.52470009|  PASSED
          sts_serial|  12|    100000|     100|0.73240630|  PASSED
          sts_serial|  13|    100000|     100|0.93585710|  PASSED
          sts_serial|  13|    100000|     100|0.99469473|  PASSED
          sts_serial|  14|    100000|     100|0.97340922|  PASSED
          sts_serial|  14|    100000|     100|0.61704871|  PASSED
          sts_serial|  15|    100000|     100|0.67604062|  PASSED
          sts_serial|  15|    100000|     100|0.43061010|  PASSED
          sts_serial|  16|    100000|     100|0.81100013|  PASSED
          sts_serial|  16|    100000|     100|0.90188464|  PASSED
         rgb_bitdist|   1|    100000|     100|0.83540782|  PASSED
         rgb_bitdist|   2|    100000|     100|0.54467922|  PASSED
         rgb_bitdist|   3|    100000|     100|0.08603828|  PASSED
         rgb_bitdist|   4|    100000|     100|0.35740458|  PASSED
         rgb_bitdist|   5|    100000|     100|0.73839099|  PASSED
         rgb_bitdist|   6|    100000|     100|0.94098118|  PASSED
         rgb_bitdist|   7|    100000|     100|0.30406384|  PASSED
         rgb_bitdist|   8|    100000|     100|0.57296031|  PASSED
         rgb_bitdist|   9|    100000|     100|0.31594410|  PASSED
         rgb_bitdist|  10|    100000|     100|0.99135737|  PASSED
         rgb_bitdist|  11|    100000|     100|0.66873736|  PASSED
         rgb_bitdist|  12|    100000|     100|0.91498165|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.60502200|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.57167784|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.40338001|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.66404486|  PASSED
    rgb_permutations|   2|    100000|     100|0.62059683|  PASSED
    rgb_permutations|   3|    100000|     100|0.41106117|  PASSED
    rgb_permutations|   4|    100000|     100|0.17650922|  PASSED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
    rgb_permutations|   5|    100000|     100|0.38673757|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.96346000|  PASSED
      rgb_lagged_sum|   1|   1000000|     100|0.95423354|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.66850821|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.39831743|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.86789885|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.73096920|  PASSED
      rgb_lagged_sum|   6|   1000000|     100|0.12755066|  PASSED
      rgb_lagged_sum|   7|   1000000|     100|0.31610895|  PASSED
      rgb_lagged_sum|   8|   1000000|     100|0.59465785|  PASSED
      rgb_lagged_sum|   9|   1000000|     100|0.97175046|  PASSED
      rgb_lagged_sum|  10|   1000000|     100|0.61078022|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  11|   1000000|     100|0.30705711|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  12|   1000000|     100|0.98392129|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  13|   1000000|     100|0.67775011|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  14|   1000000|     100|0.47260395|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  15|   1000000|     100|0.00513248|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  16|   1000000|     100|0.99750226|   WEAK
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  17|   1000000|     100|0.76659495|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  18|   1000000|     100|0.10193456|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  19|   1000000|     100|0.68425609|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  20|   1000000|     100|0.39802065|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  21|   1000000|     100|0.46401461|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  22|   1000000|     100|0.33975559|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  23|   1000000|     100|0.96417643|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  24|   1000000|     100|0.64532119|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  25|   1000000|     100|0.72118097|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  26|   1000000|     100|0.96998640|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  27|   1000000|     100|0.56975907|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  28|   1000000|     100|0.37667062|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  29|   1000000|     100|0.76577907|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  30|   1000000|     100|0.50299511|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  31|   1000000|     100|0.05107993|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  32|   1000000|     100|0.64309149|  PASSED
# The file file_input_raw was rewound 4 times
     rgb_kstest_test|   0|     10000|    1000|0.33400051|  PASSED
# The file file_input_raw was rewound 4 times
     dab_bytedistrib|   0|  51200000|       1|0.17684057|  PASSED
# The file file_input_raw was rewound 4 times
             dab_dct| 256|     50000|       1|0.95428754|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 4 times
        dab_filltree|  32|  15000000|       1|0.92957892|  PASSED
        dab_filltree|  32|  15000000|       1|0.21724793|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 4 times
       dab_filltree2|   0|   5000000|       1|0.53477448|  PASSED
       dab_filltree2|   1|   5000000|       1|0.24571782|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 4 times
        dab_monobit2|  12|  65000000|       1|0.12609546|  PASSED
```

ANNEX XII - THE SUPERLONG FILE TEST WITH 4 LAPS FOR MIXWORD() CHAINED BLOCKS

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                       saida.bin|  1.46e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.91888681|  PASSED
      diehard_operm5|   0|   1000000|     100|0.98512620|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.67407479|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.85505880|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.19356584|  PASSED
        diehard_opso|   0|   2097152|     100|0.76171650|  PASSED
        diehard_oqso|   0|   2097152|     100|0.95499590|  PASSED
         diehard_dna|   0|   2097152|     100|0.13074889|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.64054473|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.99986212|   WEAK
 diehard_parking_lot|   0|     12000|     100|0.73555833|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.35527770|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.90415849|  PASSED
     diehard_squeeze|   0|    100000|     100|0.31866389|  PASSED
        diehard_sums|   0|       100|     100|0.39434447|  PASSED
        diehard_runs|   0|    100000|     100|0.15558885|  PASSED
        diehard_runs|   0|    100000|     100|0.01076092|  PASSED
       diehard_craps|   0|    200000|     100|0.52152413|  PASSED
       diehard_craps|   0|    200000|     100|0.62657113|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.34006402|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.62480364|  PASSED
         sts_monobit|   1|    100000|     100|0.09440159|  PASSED
            sts_runs|   2|    100000|     100|0.19478862|  PASSED
          sts_serial|   1|    100000|     100|0.57208935|  PASSED
          sts_serial|   2|    100000|     100|0.27996894|  PASSED
          sts_serial|   3|    100000|     100|0.11686174|  PASSED
          sts_serial|   3|    100000|     100|0.01457715|  PASSED
          sts_serial|   4|    100000|     100|0.94881598|  PASSED
          sts_serial|   4|    100000|     100|0.18284614|  PASSED
          sts_serial|   5|    100000|     100|0.99675978|   WEAK
          sts_serial|   5|    100000|     100|0.86450768|  PASSED
          sts_serial|   6|    100000|     100|0.93531559|  PASSED
          sts_serial|   6|    100000|     100|0.99997572|   WEAK
          sts_serial|   7|    100000|     100|0.82121466|  PASSED
          sts_serial|   7|    100000|     100|0.42359909|  PASSED
          sts_serial|   8|    100000|     100|0.87206206|  PASSED
          sts_serial|   8|    100000|     100|0.38996340|  PASSED
          sts_serial|   9|    100000|     100|0.54520070|  PASSED
          sts_serial|   9|    100000|     100|0.99460369|  PASSED
          sts_serial|  10|    100000|     100|0.97411533|  PASSED
          sts_serial|  10|    100000|     100|0.94604921|  PASSED
          sts_serial|  11|    100000|     100|0.87605173|  PASSED
          sts_serial|  11|    100000|     100|0.68399251|  PASSED
          sts_serial|  12|    100000|     100|0.50432065|  PASSED
          sts_serial|  12|    100000|     100|0.30954592|  PASSED
          sts_serial|  13|    100000|     100|0.93676795|  PASSED
          sts_serial|  13|    100000|     100|0.07859232|  PASSED
          sts_serial|  14|    100000|     100|0.89488714|  PASSED
          sts_serial|  14|    100000|     100|0.54032638|  PASSED
          sts_serial|  15|    100000|     100|0.49213798|  PASSED
          sts_serial|  15|    100000|     100|0.99967595|   WEAK
          sts_serial|  16|    100000|     100|0.49555002|  PASSED
          sts_serial|  16|    100000|     100|0.35303309|  PASSED
         rgb_bitdist|   1|    100000|     100|0.92754623|  PASSED
         rgb_bitdist|   2|    100000|     100|0.53569901|  PASSED
         rgb_bitdist|   3|    100000|     100|0.62931248|  PASSED
         rgb_bitdist|   4|    100000|     100|0.70442982|  PASSED
         rgb_bitdist|   5|    100000|     100|0.89645160|  PASSED
         rgb_bitdist|   6|    100000|     100|0.79209152|  PASSED
         rgb_bitdist|   7|    100000|     100|0.88825655|  PASSED
         rgb_bitdist|   8|    100000|     100|0.37366831|  PASSED
         rgb_bitdist|   9|    100000|     100|0.72982682|  PASSED
         rgb_bitdist|  10|    100000|     100|0.62885761|  PASSED
         rgb_bitdist|  11|    100000|     100|0.82285098|  PASSED
         rgb_bitdist|  12|    100000|     100|0.53712870|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.35612157|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.35971716|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.93083228|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.86067734|  PASSED
    rgb_permutations|   2|    100000|     100|0.70608320|  PASSED
    rgb_permutations|   3|    100000|     100|0.48684288|  PASSED
    rgb_permutations|   4|    100000|     100|0.64453117|  PASSED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
    rgb_permutations|   5|    100000|     100|0.23159814|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.96896725|  PASSED
      rgb_lagged_sum|   1|   1000000|     100|0.24791620|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.94630373|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.37389967|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.89401124|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.56067467|  PASSED
      rgb_lagged_sum|   6|   1000000|     100|0.68944195|  PASSED
      rgb_lagged_sum|   7|   1000000|     100|0.74615265|  PASSED
      rgb_lagged_sum|   8|   1000000|     100|0.80192221|  PASSED
      rgb_lagged_sum|   9|   1000000|     100|0.87417345|  PASSED
      rgb_lagged_sum|  10|   1000000|     100|0.48282964|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  11|   1000000|     100|0.02865775|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  12|   1000000|     100|0.54662082|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  13|   1000000|     100|0.62714964|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  14|   1000000|     100|0.19891682|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  15|   1000000|     100|0.06521358|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  16|   1000000|     100|0.35509826|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  17|   1000000|     100|0.17865214|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  18|   1000000|     100|0.52143204|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  19|   1000000|     100|0.42142524|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  20|   1000000|     100|0.10644319|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  21|   1000000|     100|0.80533482|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  22|   1000000|     100|0.89337227|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  23|   1000000|     100|0.67380471|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  24|   1000000|     100|0.97374037|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  25|   1000000|     100|0.59041324|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  26|   1000000|     100|0.43815015|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  27|   1000000|     100|0.12124206|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|  28|   1000000|     100|0.83013363|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  29|   1000000|     100|0.66347740|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  30|   1000000|     100|0.90618975|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  31|   1000000|     100|0.68005080|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|  32|   1000000|     100|0.16966602|  PASSED
# The file file_input_raw was rewound 4 times
     rgb_kstest_test|   0|     10000|    1000|0.53838908|  PASSED
# The file file_input_raw was rewound 4 times
     dab_bytedistrib|   0|  51200000|       1|0.04493086|  PASSED
# The file file_input_raw was rewound 4 times
             dab_dct| 256|     50000|       1|0.39824367|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 4 times
        dab_filltree|  32|  15000000|       1|0.53436708|  PASSED
        dab_filltree|  32|  15000000|       1|0.99450771|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 4 times
       dab_filltree2|   0|   5000000|       1|0.22655838|  PASSED
       dab_filltree2|   1|   5000000|       1|0.80058029|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 4 times
        dab_monobit2|  12|  65000000|       1|0.44864666|  PASSED
```

ANNEX XIII - THE SUPERLONG FILE TEST WITH FULL HASH VIKTORIA CENTRAL PROCESSING

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                       saida.bin|  1.69e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.13854295|  PASSED
      diehard_operm5|   0|   1000000|     100|0.80242220|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.60724035|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.58191905|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.90072207|  PASSED
        diehard_opso|   0|   2097152|     100|0.94817661|  PASSED
        diehard_oqso|   0|   2097152|     100|0.02740765|  PASSED
         diehard_dna|   0|   2097152|     100|0.13804776|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.40879229|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.13008880|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.07295537|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.88828150|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.32123081|  PASSED
     diehard_squeeze|   0|    100000|     100|0.11278876|  PASSED
        diehard_sums|   0|       100|     100|0.07410232|  PASSED
        diehard_runs|   0|    100000|     100|0.51019171|  PASSED
        diehard_runs|   0|    100000|     100|0.02843032|  PASSED
       diehard_craps|   0|    200000|     100|0.34789190|  PASSED
       diehard_craps|   0|    200000|     100|0.59198507|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.96222415|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.83020367|  PASSED
         sts_monobit|   1|    100000|     100|0.56502802|  PASSED
            sts_runs|   2|    100000|     100|0.27396091|  PASSED
          sts_serial|   1|    100000|     100|0.63222949|  PASSED
          sts_serial|   2|    100000|     100|0.90737558|  PASSED
          sts_serial|   3|    100000|     100|0.78936254|  PASSED
          sts_serial|   3|    100000|     100|0.94901384|  PASSED
          sts_serial|   4|    100000|     100|0.99213138|  PASSED
          sts_serial|   4|    100000|     100|0.34807831|  PASSED
          sts_serial|   5|    100000|     100|0.82538199|  PASSED
          sts_serial|   5|    100000|     100|0.60633774|  PASSED
          sts_serial|   6|    100000|     100|0.27605584|  PASSED
          sts_serial|   6|    100000|     100|0.29728821|  PASSED
          sts_serial|   7|    100000|     100|0.83422426|  PASSED
          sts_serial|   7|    100000|     100|0.91732204|  PASSED
          sts_serial|   8|    100000|     100|0.58855480|  PASSED
          sts_serial|   8|    100000|     100|0.89133102|  PASSED
          sts_serial|   9|    100000|     100|0.14517019|  PASSED
          sts_serial|   9|    100000|     100|0.38282467|  PASSED
          sts_serial|  10|    100000|     100|0.86174095|  PASSED
          sts_serial|  10|    100000|     100|0.85222015|  PASSED
          sts_serial|  11|    100000|     100|0.73135621|  PASSED
          sts_serial|  11|    100000|     100|0.70742019|  PASSED
          sts_serial|  12|    100000|     100|0.98468563|  PASSED
          sts_serial|  12|    100000|     100|0.05863890|  PASSED
          sts_serial|  13|    100000|     100|0.95769684|  PASSED
          sts_serial|  13|    100000|     100|0.93268011|  PASSED
          sts_serial|  14|    100000|     100|0.63304751|  PASSED
          sts_serial|  14|    100000|     100|0.75690207|  PASSED
          sts_serial|  15|    100000|     100|0.87711563|  PASSED
          sts_serial|  15|    100000|     100|0.90190386|  PASSED
          sts_serial|  16|    100000|     100|0.57332508|  PASSED
          sts_serial|  16|    100000|     100|0.49297798|  PASSED
         rgb_bitdist|   1|    100000|     100|0.54118831|  PASSED
         rgb_bitdist|   2|    100000|     100|0.86087763|  PASSED
         rgb_bitdist|   3|    100000|     100|0.06992712|  PASSED
         rgb_bitdist|   4|    100000|     100|0.64756000|  PASSED
         rgb_bitdist|   5|    100000|     100|0.99750326|   WEAK
         rgb_bitdist|   6|    100000|     100|0.81882250|  PASSED
         rgb_bitdist|   7|    100000|     100|0.96343528|  PASSED
         rgb_bitdist|   8|    100000|     100|0.99607921|   WEAK
         rgb_bitdist|   9|    100000|     100|0.99660898|   WEAK
         rgb_bitdist|  10|    100000|     100|0.96620441|  PASSED
         rgb_bitdist|  11|    100000|     100|0.88052542|  PASSED
         rgb_bitdist|  12|    100000|     100|0.96074709|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.77000515|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.88808875|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.71520121|  PASSED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
rgb_minimum_distance|   5|     10000|    1000|0.34422199|  PASSED
    rgb_permutations|   2|    100000|     100|0.79622578|  PASSED
    rgb_permutations|   3|    100000|     100|0.65237112|  PASSED
    rgb_permutations|   4|    100000|     100|0.93870234|  PASSED
    rgb_permutations|   5|    100000|     100|0.67175075|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.59647309|  PASSED
      rgb_lagged_sum|   1|   1000000|     100|0.06582648|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.97231804|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.26556467|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.92293616|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.61897839|  PASSED
      rgb_lagged_sum|   6|   1000000|     100|0.73668709|  PASSED
      rgb_lagged_sum|   7|   1000000|     100|0.90032334|  PASSED
      rgb_lagged_sum|   8|   1000000|     100|0.35287944|  PASSED
      rgb_lagged_sum|   9|   1000000|     100|0.39386037|  PASSED
      rgb_lagged_sum|  10|   1000000|     100|0.96825243|  PASSED
      rgb_lagged_sum|  11|   1000000|     100|0.58479856|  PASSED
      rgb_lagged_sum|  12|   1000000|     100|0.94540845|  PASSED
      rgb_lagged_sum|  13|   1000000|     100|0.57519365|  PASSED
      rgb_lagged_sum|  14|   1000000|     100|0.99749184|   WEAK
      rgb_lagged_sum|  15|   1000000|     100|0.37580944|  PASSED
      rgb_lagged_sum|  16|   1000000|     100|0.93752211|  PASSED
      rgb_lagged_sum|  17|   1000000|     100|0.78785383|  PASSED
      rgb_lagged_sum|  18|   1000000|     100|0.26379760|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  19|   1000000|     100|0.96151665|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  20|   1000000|     100|0.74272681|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  21|   1000000|     100|0.77203198|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  22|   1000000|     100|0.20024292|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  23|   1000000|     100|0.47653247|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  24|   1000000|     100|0.45179371|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  25|   1000000|     100|0.79676496|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  26|   1000000|     100|0.21967819|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  27|   1000000|     100|0.70138813|  PASSED
# The file file_input_raw was rewound 1 times
      rgb_lagged_sum|  28|   1000000|     100|0.37782906|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  29|   1000000|     100|0.12114665|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  30|   1000000|     100|0.85766920|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  31|   1000000|     100|0.82934211|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|  32|   1000000|     100|0.98902419|  PASSED
# The file file_input_raw was rewound 2 times
     rgb_kstest_test|   0|     10000|    1000|0.46591312|  PASSED
# The file file_input_raw was rewound 2 times
     dab_bytedistrib|   0|  51200000|       1|0.31983884|  PASSED
# The file file_input_raw was rewound 2 times
             dab_dct| 256|     50000|       1|0.50032067|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 2 times
        dab_filltree|  32|  15000000|       1|0.41546106|  PASSED
        dab_filltree|  32|  15000000|       1|0.50056801|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 2 times
       dab_filltree2|   0|   5000000|       1|0.72847591|  PASSED
       dab_filltree2|   1|   5000000|       1|0.59749122|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 2 times
        dab_monobit2|  12|  65000000|       1|0.00914153|  PASSED
```

ANNEX XIV

Checking bit sequences that appear at each hash position for the 3 functions:

1) SHA2-512
2) SHA3-512
3) VIKTORIA

In the first column we have the position where the bit sequence appears. In the body of the table we have the number of occurrences per position for the three algorithms.

In the footer of the table we have the total of occurrences (green), the total of occurrences that most appears considering all the positions (yellow) and the average of the total of occurrences for each algorithm (red).

The averages foreseen for all the bits occurrences are, from left to right: 1/256, 1/16, 1, 16, 256 and 4096.

For the three algorithms this pattern is observed in the test.

HEXADECIMAL 0																		
Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	1	1	0	12	18	10	265	260	220	4010	4001	3996
1	0	0	0	0	0	0	0	3	0	17	17	15	255	268	244	4152	4128	3961
2	0	0	0	0	0	0	0	1	0	12	22	13	241	269	282	4174	4161	4174
3	0	0	0	0	0	0	2	1	2	22	19	14	261	268	226	4103	4114	4101
4	0	0	0	0	0	0	1	0	0	22	19	19	292	267	225	4109	4135	4154
5	0	0	0	0	0	1	1	2	1	12	18	17	260	267	272	4144	4155	3942
6	0	0	0	0	0	0	0	1	3	14	14	19	256	282	255	4025	4107	4085
7	0	0	0	0	0	0	0	0	0	11	19	19	265	267	278	4141	4116	4171
8	0	0	0	0	0	0	2	0	2	7	14	21	238	242	271	4104	4047	4120
9	0	0	0	0	0	0	0	0	0	15	7	13	244	241	282	4087	4023	4116
10	0	0	0	0	0	0	1	0	0	12	15	15	263	234	240	4133	4086	4104
11	0	0	0	0	0	0	1	0	0	20	11	22	266	254	284	4124	4076	4113
12	0	0	0	0	0	0	1	1	0	21	9	18	266	243	251	4085	4053	4155
13	0	0	0	0	0	0	0	1	2	15	24	23	262	275	298	4100	4143	4189
14	0	0	0	0	0	0	3	0	0	17	22	13	266	280	261	4181	4168	4137
15	0	0	0	0	0	0	0	0	1	21	14	14	271	250	250	4147	4060	4104
16	0	0	0	0	0	0	1	1	1	21	25	13	304	264	246	4211	4160	4046
17	0	0	0	0	0	0	1	3	1	18	20	15	270	273	239	4141	4257	4148
18	0	0	0	0	0	0	1	0	1	17	13	15	270	243	234	4206	4125	4021
19	0	0	0	0	0	0	2	0	0	23	12	10	253	249	245	4114	4071	3989
20	0	0	0	0	0	0	0	1	3	19	16	15	256	256	240	3972	4100	4083
21	0	0	0	0	0	0	1	0	0	11	22	22	232	260	270	4078	4029	4055
22	0	0	0	0	0	0	0	0	2	7	10	16	240	238	253	4018	4037	4144
23	0	0	0	0	0	0	0	2	0	16	16	13	255	254	247	4143	4065	4089
24	0	0	0	0	0	0	1	0	1	14	19	8	256	220	283	4196	4094	4123
25	0	0	0	0	0	0	0	2	0	14	17	14	270	248	226	4140	4060	4061
26	0	0	0	0	0	0	1	1	0	13	14	13	264	251	266	4175	4085	4142
27	0	0	0	0	1	0	0	2	2	14	8	13	239	256	223	4173	4124	4098
28	0	0	0	0	0	0	2	1	2	26	13	15	266	258	235	3981	4121	3943
29	0	0	0	0	0	0	1	0	1	15	11	14	245	258	261	4114	4079	4073
30	0	0	0	0	1	1	1	3	3	15	13	17	262	270	274	4044	3994	4189
31	0	0	0	0	0	0	1	1	1	23	16	20	268	275	257	4181	4112	4098
32	0	0	0	0	0	0	0	1	1	14	13	18	252	255	272	4114	3997	4158
33	0	0	0	0	0	0	0	1	1	12	19	18	234	242	258	4089	4208	4144
34	0	0	0	0	0	0	0	0	1	25	18	16	269	289	244	4089	4080	4163
35	0	0	0	0	0	1	1	1	4	12	13	17	286	266	281	4135	4102	4060
36	0	0	0	0	0	1	0	1	2	11	11	18	242	225	242	4093	4060	4145
37	0	0	0	0	0	0	0	0	1	6	11	16	256	225	283	4170	3966	4133
38	0	0	0	0	1	0	3	3	2	13	12	14	261	242	290	4069	4146	4061
39	0	0	0	0	0	0	2	1	3	24	14	14	260	259	250	4137	4117	4060
40	0	0	0	1	0	0	1	1	2	16	15	14	240	285	235	4030	4082	3976
41	0	0	0	0	0	0	2	0	1	16	20	17	273	237	256	4167	4021	4004
42	0	0	0	0	0	0	4	2	1	22	20	16	288	278	289	4232	4089	4195
43	0	0	0	1	0	0	1	2	0	16	22	9	259	283	250	4048	4251	4203
44	0	0	0	0	1	0	1	1	0	9	16	19	263	258	246	4100	4087	4098
45	0	0	0	0	0	0	1	2	0	20	14	13	248	262	275	4190	4142	4031
46	0	0	0	0	0	0	0	0	0	13	12	16	220	256	279	4165	4175	4168
47	0	0	0	1	0	0	1	0	0	14	11	12	248	259	254	4075	4101	4162
48	0	0	0	0	0	0	1	0	2	15	15	18	258	261	251	4060	4071	4094
49	0	0	0	1	0	0	2	0	0	15	8	13	221	259	248	4072	4035	4049
50	0	0	0	0	0	0	2	0	1	12	15	11	257	248	225	4144	3987	4121
51	0	0	0	0	0	0	0	1	1	10	14	5	259	276	230	3915	4103	3987
52	0	0	0	0	1	0	0	1	0	8	20	15	239	236	242	4025	4065	4046
53	0	0	0	0	0	0	2	2	0	14	15	14	261	278	285	4051	4075	4105
54	0	0	0	0	0	0	5	2	0	24	20	16	240	258	252	4136	4250	4133
55	0	0	0	0	0	1	2	1	2	24	15	18	256	267	250	3996	4106	4149
56	0	0	0	0	0	0	2	2	1	10	15	12	273	265	254	4067	4149	4025
57	0	0	0	0	0	0	0	3	0	23	22	10	291	254	250	4163	4134	4129
58	0	0	0	0	0	0	1	1	3	13	14	21	292	256	260	4100	4101	4038
59	0	0	0	0	1	0	0	1	0	20	13	14	241	254	212	4075	4090	3999
60	0	0	0	0	0	0	0	1	1	18	17	15	298	219	251	4039	4101	4149
61	0	0	0	0	0	0	1	0	0	17	13	21	268	270	250	4171	4086	4080
62	0	0	0	0	0	0	2	0	0	21	14	13	269	255	232	4149	4107	4034
63	0	0	0	0	0	1	2	1	1	20	13	14	260	267	222	4086	4171	4030
64	0	0	0	0	0	0	0	1	2	15	15	16	268	256	250	4060	4205	4026
65	0	0	0	0	0	0	0	0	2	14	16	16	245	256	282	4049	4082	4032
66	0	0	0	0	0	1	1	1	3	16	12	17	257	261	279	4148	3986	4124
67	0	0	0	0	0	0	0	2	2	13	16	24	238	274	264	4096	4207	4021
68	0	0	0	0	0	0	0	0	0	17	9	18	262	266	263	3934	4070	4159
69	0	0	0	0	0	0	0	1	0	16	13	15	247	280	250	4192	4146	4088
70	0	0	0	0	0	0	2	5	0	19	17	15	286	259	247	4047	4117	4159
71	0	0	0	0	0	1	1	1	2	10	23	13	232	251	234	4083	4090	4037
72	0	0	0	0	0	0	1	2	1	22	16	21	242	263	276	4024	3993	4044
73	0	0	0	1	0	0	3	1	1	14	20	20	251	240	267	4128	4190	4056
74	0	0	0	0	0	0	2	1	2	17	15	18	261	249	237	4030	4108	4028
75	0	0	0	0	0	0	0	1	1	20	14	17	257	248	276	4180	4015	4150
76	0	0	0	0	0	0	0	0	2	14	20	14	263	257	244	4087	4049	4121
77	0	0	0	0	0	0	0	1	2	19	15	17	243	251	252	4177	4148	4138
78	0	0	0	0	0	0	2	2	1	14	18	18	282	260	259	4125	4083	4102
79	0	0	0	0	0	0	0	0	0	18	17	18	271	253	266	4134	4002	4110
80	0	0	0	0	0	1	0	0	1	9	17	12	256	248	246	4081	4095	4006
81	0	0	0	0	0	0	0	4	1	23	27	17	259	267	262	4090	4125	4159
82	0	0	0	0	0	0	1	0	1	19	15	9	273	266	259	4010	4028	4158
83	0	0	0	1	0	0	1	0	1	14	14	21	262	237	244	4184	4031	4288
84	0	1	0	0	1	0	2	1	1	17	22	11	263	234	250	4186	4141	4020
85	0	0	0	1	1	0	2	1	4	21	13	23	243	254	254	4054	4035	4135
86	0	0	1	0	0	1	2	1	3	27	13	22	273	256	268	4127	4021	4002
87	0	0	0	0	0	1	1	1	1	25	21	14	258	253	283	4019	4119	4166
88	0	0	0	0	0	0	1	1	1	12	18	12	254	259	238	4107	4060	4134
89	0	0	0	0	0	0	1	2	3	20	16	23	249	266	246	4038	4112	3949
90	0	0	0	1	0	0	1	1	0	16	16	13	253	257	290	4118	4157	4139
91	0	0	0	0	0	0	1	0	0	15	12	17	266	245	277	4155	4021	4067
92	0	0	0	0	0	0	2	1	0	16	16	12	248	245	294	4100	4038	4079
93	0	0	0	0	0	0	1	1	0	15	16	25	239	234	266	4191	4133	4166
94	0	0	0	0	0	0	1	0	1	17	16	18	232	235	292	4161	3979	4144
95	0	0	0	0	0	0	0	0	0	18	12	17	251	261	268	4072	4059	4066
96	0	0	0	0	0	0	1	1	1	18	8	15	237	226	239	4135	4107	4090
97	0	0	0	0	0	0	0	2	1	11	21	8	278	241	227	4071	4088	4034
98	0	0	0	0	0	0	3	0	0	23	14	13	263	246	223	4135	3964	3994
99	0	0	0	0	0	0	0	1	0	22	17	18	253	260	253	4017	4097	4121
100	0	0	0	0	0	0	1	0	2	14	9	9	263	268	253	4154	4240	4152
101	0	0	0	0	0	0	1	1	1	21	15	11	237	245	244	4048	3965	4179
102	0	0	0	0	0	0	2	0	0	21	19	9	263	229	247	4001	4068	4054
103	0	0	0	0	0	0	0	0	2	20	11	17	251	253	255	4125	4181	4070
104	0	0	0	0	0	0	0	3	0	10	21	8	263	245	258	4094	4039	4133
105	0	0	0	0	0	0	0	1	1	12	18	14	223	254	264	3916	4074	4152
106	0	0	0	0	0	0	0	0	0	9	14	15	267	276	240	4036	4040	4034
107	0	0	0	0	0	0	2	2	1	14	19	14	234	252	255	4108	4050	4053
108	0	0	0	0	0	0	1	0	1	17	17	21	246	246	253	4069	4100	4116
109	0	0	0	0	0	0	1	0	1	15	15	20	232	257	296	4019	4109	4021
110	0	0	0	0	0	1	2	3	2	13	19	14	235	267	264	4155	4162	4186
111	0	0	0	0	0	0	1	1	2	15	14	20	278	274	261	4058	4109	4027
112	0	0	0	0	0	1	0	1	3	16	16	17	243	260	263	4132	4127	4107
113	0	0	0	0	0	0	0	0	1	13	18	19	242	285	248	4124	4080	4059
114	0	0	0	0	0	0	0	2	2	13	15	15	279	279	276	4163	4114	4080
115	0	0	0	0	0	0	0	1	1	20	16	20	223	236	252	4087	4034	4254
116	0	0	0	0	0	0	0	2	0	17	16	9	261	231	220	4055	3944	4127
117	0	0	0	0	0	0	0	3	0	24	15	15	270	259	248	4088	4136	4044
118	0	0	0	0	0	0	0	1	2	15	19	17	267	225	263	4083	4146	4061
119	0	0	0	0	0	0	2	1	4	16	16	19	266	280	261	3996	4099	4112
120	0	0	0	0	0	0	0	0	1	15	21	15	245	230	266	4088	4014	4074
121	0	0	0	0	0	0	0	0	0	16	15	11	249	246	228	4091	4196	4026
122	0	0	0	0	0	0	1	0	0	18	13	22	236	237	250	4146	4114	4106
123	**0**	**1**	**1**	0	0	0	0	0	0	16	9	15	267	207	246	4163	4083	4057
124	**0**	**0**	**0**	**8**	**8**	**13**	0	0	1	21	10	14	281	263	244	4201	4116	4073
125	0	0,00813	0,00813	**0**	**0**	**0**	**112**	**119**	**129**	11	11	19	283	246	230	4112	4110	4043
126				0,064516	0,064516	0,104839	**0**	**1**	**0**	**2051**	**1972**	**1971**	259	242	241	4063	4032	4092
127							0,896	0,952	1,032	**14**	**15**	**15**	**32681**	**32390**	**32407**	4091	4092	3981
										16,27778	15,65079	15,64286	**263**	**267**	**250**	**524869**	**523816**	**523638**
													257,3307	255,0394	255,1732	**4100**	**4060**	**4021**
																4100,539	4092,313	4090,922

HEXADECIMAL 1																		
Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	0	0	0	20	15	15	276	238	236	4014	4159	4063
1	0	0	0	0	0	0	0	0	1	14	13	11	260	242	250	4228	4089	4083
2	0	0	0	0	0	0	2	1	0	20	19	13	254	264	242	4022	3993	4066
3	0	0	0	1	0	0	1	1	1	20	22	18	258	253	239	4099	4079	4065
4	0	0	0	0	0	0	2	2	0	18	18	10	273	253	246	4137	4094	4000
5	0	0	0	0	0	0	0	1	0	26	19	12	268	272	245	4113	4133	4099
6	0	0	0	0	0	0	2	1	1	15	12	14	277	259	231	4145	4072	3989
7	0	0	0	0	0	0	1	4	2	14	23	15	236	293	245	4079	4077	4117
8	0	0	0	0	0	0	1	2	2	18	20	12	266	268	265	4066	4195	4042
9	0	0	0	0	0	0	3	1	0	19	15	16	261	245	276	4120	4092	4114
10	0	0	0	0	0	0	0	1	0	16	18	15	265	260	266	4117	4091	4151
11	0	0	0	0	0	0	0	1	1	11	10	16	256	250	241	4039	4180	4061
12	0	0	0	0	0	0	0	0	0	23	15	13	263	245	262	4104	4083	4006
13	0	1	0	0	1	0	0	2	0	23	13	10	246	243	214	4025	4215	4020
14	0	0	0	0	1	0	0	3	0	17	21	12	264	244	256	4035	4078	4003
15	0	0	0	0	0	0	0	2	0	15	16	11	252	286	248	3971	4142	4030
16	0	0	0	0	0	0	0	1	1	11	19	17	250	275	253	4146	4063	4067
17	0	0	0	0	0	0	0	0	3	15	15	15	238	256	268	4087	4033	4204
18	0	0	0	0	0	0	3	1	1	17	17	25	261	248	260	4048	4111	4052
19	0	0	0	0	0	0	0	0	0	16	17	13	262	266	250	4019	4015	4085
20	0	0	0	0	0	0	0	0	1	10	16	11	260	265	260	4230	4179	4091
21	0	0	0	0	0	0	0	0	1	15	16	16	261	244	255	4032	4127	4046
22	0	0	0	0	0	0	1	1	0	8	15	9	240	241	259	4139	4213	4040
23	0	0	0	0	0	0	2	0	2	18	23	23	276	274	256	4202	4131	4021
24	0	0	0	0	0	0	1	0	0	18	14	18	259	273	237	4097	4094	4106
25	0	0	0	0	0	0	1	0	0	13	17	20	224	243	269	4045	4113	4144
26	0	0	0	0	0	0	1	2	0	17	26	13	267	262	236	4037	4135	4039
27	0	0	0	0	0	0	0	1	1	15	20	9	277	260	246	4190	4098	4026
28	0	0	0	0	0	0	1	0	1	13	26	16	261	279	237	4114	4109	4180
29	0	0	0	0	0	0	0	0	1	9	12	15	264	259	252	4127	4051	4055
30	0	0	0	0	0	0	3	2	1	19	15	15	247	222	266	4092	4107	4082
31	0	0	0	0	0	0	0	1	2	19	16	17	268	238	253	4137	3964	4106
32	0	0	0	0	0	0	0	2	1	15	16	12	292	234	238	4230	4058	4040
33	0	0	0	0	0	0	1	2	1	11	9	14	250	239	246	4074	4108	4107
34	0	0	0	0	0	0	1	2	1	22	20	11	257	246	229	4114	3992	4096
35	0	0	0	0	0	0	0	1	0	15	23	17	291	283	267	4160	4036	4057
36	0	0	0	0	0	0	2	5	0	12	23	14	267	278	261	4169	4128	4134
37	0	0	0	0	0	0	0	0	1	14	18	23	245	264	256	3994	4168	4112
38	0	0	0	0	0	0	0	0	1	16	12	16	294	267	288	4096	4201	4024
39	0	0	0	0	0	0	0	1	0	14	16	9	264	265	249	4030	4168	4148
40	0	0	0	0	0	0	1	1	0	17	23	13	266	254	246	4094	4158	4145
41	0	0	0	0	0	0	2	1	1	14	16	16	247	258	239	4077	4098	4073
42	0	0	0	0	0	0	1	2	1	18	21	23	218	267	283	4033	4151	4096
43	0	0	0	0	0	0	1	3	0	7	11	20	236	260	276	4023	4115	4156
44	0	0	0	0	0	0	2	0	1	15	14	14	259	247	288	4104	4158	4154
45	0	0	0	0	0	0	0	1	0	17	20	16	255	240	250	4138	4054	4185
46	0	0	0	0	0	0	0	1	1	19	18	16	260	254	252	4144	4007	4055
47	0	0	0	0	0	0	1	1	1	21	13	14	269	259	279	4136	4065	4052
48	0	0	0	0	0	0	0	1	0	13	17	13	240	246	239	4073	4163	4043
49	0	0	0	0	0	0	2	0	1	20	11	9	252	261	266	4064	4057	4072
50	0	0	0	0	0	0	0	2	1	17	22	17	270	252	252	4016	4179	4121
51	0	0	0	0	0	0	0	0	2	13	15	18	271	263	250	4157	4006	4123
52	0	0	0	0	0	0	1	0	1	11	17	13	253	267	241	3972	4095	4158
53	0	0	0	0	0	0	2	0	0	16	18	16	234	274	245	4067	4030	4047
54	0	0	0	0	0	0	0	1	1	18	23	10	273	279	232	4106	4202	4161
55	0	0	0	0	0	0	1	0	0	19	10	19	275	248	266	4096	4096	4072
56	0	0	0	0	0	0	1	0	2	19	12	13	278	239	258	4105	4081	4092
57	0	0	0	0	0	0	2	1	1	17	13	11	263	256	240	4022	4076	4087
58	0	0	0	0	0	0	0	2	0	17	17	13	256	239	263	4127	4112	4043
59	0	0	0	0	0	0	1	2	1	10	17	18	258	237	243	4124	4045	4144
60	0	0	0	0	0	0	1	1	1	14	20	14	215	271	249	4086	4046	4030
61	0	0	0	0	0	0	2	0	2	21	14	15	276	265	246	4079	4090	4058
62	0	0	0	0	0	1	2	1	3	19	14	28	267	250	270	4178	4160	4132
63	0	0	0	0	0	0	0	3	2	19	23	18	270	238	279	4133	4025	4091
64	0	0	0	0	0	0	0	0	3	12	15	16	283	236	254	4087	4080	4193
65	0	0	0	0	0	0	1	0	0	19	13	13	249	258	256	4125	4134	4103
66	0	0	0	0	0	0	2	1	2	15	10	15	274	267	259	4045	4098	4102
67	0	0	0	0	0	0	0	0	0	19	17	19	241	227	241	4099	4098	3971
68	0	0	0	0	1	0	1	2	0	19	15	15	240	239	276	4043	4050	4146
69	0	0	0	0	0	0	2	3	1	24	19	15	264	244	259	4141	4014	4098
70	0	0	0	0	0	0	1	2	2	15	15	13	278	290	233	4137	4225	4074
71	0	0	0	0	0	0	0	1	1	15	21	17	247	243	256	4141	4128	3975
72	0	0	0	0	0	0	1	0	0	18	26	15	259	242	253	4099	4043	4118
73	0	0	0	0	0	0	0	1	2	8	16	19	249	281	249	4029	4094	4058
74	0	0	0	0	0	0	2	0	1	9	14	17	216	245	268	4082	4050	4185
75	0	0	0	0	0	0	1	2	3	12	14	12	236	292	245	4095	4158	4204
76	0	0	0	0	1	0	1	2	2	16	15	22	231	273	247	4034	4173	4078
77	0	0	0	0	0	0	0	3	3	23	13	19	290	271	280	4107	4064	4074
78	0	0	0	0	0	0	1	1	1	12	12	18	271	250	278	4092	4074	4016
79	0	0	0	0	0	0	0	1	0	22	18	19	259	228	254	4195	3993	4170
80	0	0	0	0	0	0	2	0	2	21	14	19	284	258	259	4117	3957	4119
81	0	0	0	0	0	0	0	1	0	15	19	12	250	273	252	4102	4156	4003
82	0	0	0	1	0	0	2	1	0	14	23	19	255	277	246	4057	4204	4176
83	0	0	0	0	0	0	2	0	1	17	13	14	248	256	284	4050	4146	4123
84	0	0	0	0	0	0	1	1	0	24	17	14	257	291	245	4090	4006	4175
85	0	0	0	0	0	0	1	1	1	11	16	22	252	259	249	4031	4210	4139
86	0	0	0	0	1	0	1	3	3	15	17	12	238	260	239	4059	4088	4109
87	0	0	0	1	0	0	1	1	0	13	9	21	268	233	259	4113	4104	4041
88	0	0	0	0	0	0	2	1	0	20	12	15	236	246	255	4018	3996	3947
89	0	0	0	1	1	0	3	2	0	23	17	19	262	265	268	4118	4090	4087
90	0	0	0	0	0	0	3	4	0	21	16	11	268	269	287	4079	4073	4067
91	0	0	0	0	0	0	1	1	4	18	19	13	230	237	244	4070	4073	4175
92	0	0	0	0	1	0	2	2	0	19	15	19	241	231	256	4034	4037	4077
93	0	0	0	0	0	0	1	2	0	15	20	21	251	237	275	4054	3991	4220
94	0	0	0	0	0	0	0	1	1	17	14	14	256	280	267	4170	4108	4102
95	0	0	0	0	0	0	0	2	2	12	18	15	258	264	279	4123	4080	4194
96	0	0	0	0	0	0	1	0	0	14	17	13	259	266	254	4125	4135	4107
97	0	0	0	0	1	0	3	2	0	15	16	15	269	258	259	4185	4068	4168
98	0	0	0	0	0	0	4	3	2	24	15	17	261	260	225	4102	3974	4064
99	0	0	0	0	0	0	0	1	0	12	15	15	249	252	270	4028	4151	4171
100	0	0	0	0	0	0	2	1	0	15	9	12	262	264	254	4129	4090	4047
101	0	0	0	0	0	1	0	2	1	17	16	17	267	237	242	4169	4036	4075
102	0	0	0	0	1	0	2	1	1	18	17	22	251	263	266	4117	4025	4009
103	0	0	0	0	0	0	0	1	1	14	14	21	268	255	273	4139	4045	4075
104	0	0	0	0	0	0	1	0	2	16	11	16	245	242	236	4128	4077	4087
105	0	0	0	0	0	0	0	0	1	22	17	16	268	255	239	4019	4170	4095
106	0	0	0	0	0	0	0	0	1	13	10	19	298	294	267	4143	4112	3980
107	0	0	0	0	0	0	0	0	1	12	20	20	259	270	255	4187	4159	4144
108	0	0	0	0	0	0	0	1	0	10	13	13	240	257	258	4059	4280	4052
109	0	0	0	0	0	0	1	0	2	16	17	24	273	272	256	4101	4113	4153
110	0	0	0	0	0	0	0	0	2	12	10	23	235	241	258	4066	4172	3966
111	0	0	0	0	0	0	0	2	2	16	14	18	250	260	260	4052	4090	4238
112	0	0	0	0	0	0	5	1	1	13	20	22	269	262	282	4145	4092	4030
113	0	0	0	0	0	0	1	1	1	18	21	12	217	297	255	4140	4139	4193
114	0	1	0	0	1	0	1	1	1	18	17	13	229	220	247	3985	4164	4061
115	0	0	0	0	1	0	2	3	0	15	12	13	255	261	247	4069	3969	4108
116	0	0	0	0	1	0	1	4	3	14	14	19	270	253	274	4053	4110	4125
117	0	0	0	0	0	0	1	2	1	16	20	13	273	263	255	4171	4031	4184
118	0	0	0	0	0	0	1	1	1	12	17	23	257	259	274	4197	4091	4133
119	0	0	0	0	0	0	1	1	1	12	17	15	253	248	282	4030	4179	4257
120	0	0	0	0	0	0	0	1	0	13	14	14	246	248	238	4034	4147	4137
121	0	0	0	0	0	0	0	1	1	12	19	17	217	236	238	4032	3989	4064
122	0	0	0	0	0	0	2	2	3	15	16	24	253	268	262	4137	4108	4025
123	0	2	0	0	0	0	0	0	1	15	10	19	256	249	271	4117	4145	4131
124	0	0	0	4	12	2	2	1	0	20	17	14	235	247	248	3985	4078	4197
125	0	0,01626	0	0	0	0	118	143	116	16	21	18	264	239	274	4023	4069	4147
126				0,032258	0,096774	0,016129	0	1	1	2020	2067	1999	258	271	273	4070	4158	4041
127							0,944	1,144	0,928	15	17	13	32658	32609	32469	4091	4042	4092
										16,03175	16,40476	15,86508	268	260	256	523941	524446	524036
													257,1496	256,7638	255,6614	4137	4098	4030
																4093,289	4097,234	4094,031

ANNEX XV

Checking bit sequences that appear at each hash position for the 3 functions:

1) SHA2-512
2) SHA3-512
3) VIKTORIA

In the first column we have the position where the bit sequence appears. In the body of the table we have the number of occurrences per position for the three algorithms.

In the footer of the table we have the total of occurrences (green), the total of occurrences that most appears considering all the positions (yellow) and the average of the total of occurrences for each algorithm (red).

The averages foreseen for all the bits occurrences are, from left to right: 1/256, 1/16, 1, 16, 256 and 4096.

For the three algorithms this pattern is observed in the test.

HEXADECIMAL 2

Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	1	0	0	2	0	17	23	13	277	269	256	4166	4114	4138
1	0	0	0	0	0	0	0	2	1	17	13	12	252	263	225	4106	4050	4071
2	1	0	0	1	0	0	1	1	1	10	15	14	243	263	222	3991	4140	4027
3	0	0	0	2	0	0	2	0	0	25	13	13	241	229	244	4060	4089	4057
4	0	0	0	0	0	0	4	2	1	15	18	17	258	263	237	4035	4119	4229
5	0	0	0	0	0	0	1	2	0	18	13	16	247	256	270	4166	4046	4097
6	0	0	0	1	0	0	1	1	0	12	20	10	277	281	249	4088	4067	3968
7	0	0	0	0	0	0	1	0	0	9	15	15	244	250	237	4101	4027	4020
8	0	0	0	0	0	0	1	0	1	15	15	18	288	287	237	4082	4039	4053
9	0	0	0	0	0	0	0	0	0	8	9	9	224	235	272	4100	4065	4031
10	0	0	0	0	0	0	0	0	2	15	19	7	226	258	237	4017	4119	4121
11	0	0	0	0	0	0	0	0	0	13	16	21	258	277	229	4140	4121	3978
12	0	0	0	0	0	0	0	0	0	12	16	9	268	272	286	4080	4117	4094
13	0	0	0	0	0	0	1	1	1	17	11	9	242	260	231	4061	4094	4092
14	0	0	0	0	0	0	2	1	1	15	15	22	245	275	243	4152	4160	4094
15	0	0	0	0	0	0	2	2	2	23	21	14	260	266	296	4031	4105	4072
16	0	0	0	0	0	0	0	1	0	18	24	16	262	286	249	4076	4127	4105
17	0	0	0	0	0	0	2	0	0	11	21	16	252	280	256	4102	4219	4084
18	0	0	0	0	0	0	0	0	2	7	14	18	231	265	237	4094	4165	4099
19	0	0	0	0	0	0	0	1	1	14	14	17	232	232	253	3977	4078	4124
20	0	0	0	0	0	1	0	0	2	11	18	19	238	261	297	4065	3976	4142
21	0	0	0	0	0	0	2	0	2	25	14	13	275	265	242	4107	4108	4234
22	0	0	0	0	0	0	0	0	1	16	17	18	261	275	260	4064	4131	4018
23	0	0	0	0	0	0	0	0	1	18	12	10	253	263	252	4121	4108	4058
24	0	0	0	0	0	0	2	1	2	19	13	18	244	281	241	3985	4109	4199
25	0	0	0	0	0	0	1	0	1	14	17	13	240	240	241	4065	4040	4130
26	0	0	0	0	0	0	0	1	1	17	13	21	267	290	252	4026	4084	4110
27	0	0	0	0	0	0	1	1	2	18	24	16	266	268	242	4135	4084	4098
28	0	0	0	0	0	0	0	1	0	16	17	20	255	275	279	4029	4015	4134
29	0	0	0	0	0	0	0	2	0	8	16	13	244	267	262	4031	4144	4143
30	0	0	0	0	0	0	1	0	0	11	15	13	242	261	242	4037	4192	4074
31	0	0	0	0	0	0	0	0	1	22	13	21	235	265	243	4101	4176	4107
32	0	0	0	0	0	0	0	0	0	17	6	17	263	240	245	4114	4067	4035
33	0	0	0	0	0	0	0	3	1	12	17	15	233	278	244	4152	4082	3940
34	0	0	0	0	0	0	1	0	2	20	18	21	254	263	251	4096	4204	4143
35	0	0	0	0	0	0	0	1	1	18	11	19	277	243	262	4028	3952	4137
36	0	0	0	0	1	0	0	1	0	14	17	14	259	267	245	4105	4082	4183
37	0	0	0	0	0	0	0	1	1	14	10	14	252	251	268	4113	4072	4091
38	0	0	0	0	0	0	1	0	1	9	16	15	266	285	245	4123	4116	4113
39	0	0	0	0	0	0	0	2	1	20	18	19	247	248	244	4179	4143	4102
40	0	0	0	0	1	0	0	2	1	20	12	15	255	258	269	4054	4063	4008
41	0	0	0	0	0	0	1	1	1	17	18	15	263	259	245	4156	4069	4155
42	0	0	0	0	0	0	2	1	0	12	13	18	253	232	249	4137	4215	4061
43	0	0	0	0	0	0	0	0	1	18	14	15	243	272	268	4124	4074	4059
44	0	0	0	0	0	0	1	1	0	18	14	19	253	237	270	4026	4095	4039
45	0	0	0	0	0	0	0	2	0	16	19	19	254	278	266	4234	4183	4159
46	0	0	0	0	0	0	0	1	1	21	18	20	247	265	238	3963	4084	4069
47	0	0	0	0	0	1	1	1	3	16	14	19	288	237	253	4236	4140	3962
48	0	0	0	0	0	0	0	3	2	14	19	17	258	261	251	4118	3998	4107
49	0	0	0	0	0	0	2	1	1	19	13	22	262	255	252	3978	4090	4090
50	0	0	0	0	0	0	3	1	0	18	19	10	285	240	274	4068	4016	3971
51	0	0	0	0	0	0	1	0	0	26	8	11	247	243	231	4106	4090	4113
52	0	0	0	0	1	0	1	3	0	17	23	15	256	266	244	4182	4089	4000
53	0	0	0	0	0	0	1	5	2	12	24	19	250	243	246	4119	4036	4056
54	0	0	0	0	0	0	3	1	1	15	15	13	266	237	242	4080	3952	3920
55	0	0	0	0	0	0	0	2	0	9	19	11	232	283	269	4069	3970	4021
56	0	0	0	0	0	0	2	1	2	13	23	18	229	265	251	4011	4148	4022
57	0	0	0	0	0	0	1	1	1	20	18	20	271	266	250	4124	4224	4043
58	0	0	0	0	0	0	0	1	1	19	21	15	260	250	247	4044	4138	4020
59	0	0	0	1	0	0	1	0	0	10	20	25	257	255	259	4064	4046	4087
60	0	0	0	1	0	0	2	1	0	15	16	9	255	280	247	4196	4099	3942
61	0	0	0	1	0	0	2	0	0	12	16	22	233	245	269	4041	4118	4171
62	0	0	0	0	0	0	4	1	1	19	20	14	235	265	244	4078	4066	4156
63	0	0	0	0	0	0	0	1	1	18	14	20	245	250	252	4125	4101	4058
64	0	0	0	0	0	0	0	2	0	14	19	13	253	242	260	4081	4181	4098
65	0	0	0	0	0	0	1	0	0	9	26	14	238	273	257	4144	4112	4194
66	0	0	0	0	0	0	0	0	1	10	19	24	231	273	243	4051	4107	4149
67	0	0	0	1	0	0	1	1	2	12	16	16	239	265	280	3984	4101	4029
68	0	0	0	0	0	0	3	1	0	18	16	15	253	266	264	4180	4129	4170
69	0	0	0	0	0	0	2	0	0	20	19	12	227	241	259	3964	4137	4071
70	0	0	0	0	0	0	0	1	2	13	14	17	248	226	250	4060	4061	4125
71	0	0	0	1	0	0	2	1	2	19	16	17	283	268	241	4088	4132	4098
72	0	0	0	0	0	0	1	1	0	15	14	14	262	265	264	4105	3988	4074
73	0	0	0	0	0	0	0	1	1	19	22	9	264	252	253	4069	4000	4076
74	0	0	0	0	0	0	2	0	0	19	13	17	286	267	266	4095	4121	4197
75	0	0	0	0	0	0	0	1	1	16	20	8	257	261	253	4112	4069	4164
76	0	0	0	0	0	0	0	1	0	15	18	11	254	255	228	4019	4089	4058
77	0	0	0	0	0	0	0	0	3	20	10	20	236	243	257	4142	4171	4134
78	0	0	0	0	0	0	1	0	0	21	12	19	253	263	278	4068	4059	4152
79	0	0	0	0	0	0	0	1	0	15	17	17	244	263	255	4120	4171	4144
80	0	0	0	0	0	0	0	0	0	12	8	23	280	266	255	4108	4127	4020
81	0	0	0	0	0	0	1	1	1	11	15	16	248	238	248	4208	4047	4113
82	0	0	0	0	0	0	3	0	1	22	18	11	267	241	234	4132	4056	4049
83	0	0	0	0	0	0	1	0	0	14	15	19	275	254	219	4091	4102	4121
84	0	0	0	0	0	0	1	0	1	14	19	16	251	243	257	4110	4058	4077
85	0	0	0	0	0	0	1	1	0	19	14	17	247	272	263	4069	4206	4139
86	0	0	0	0	0	0	0	0	0	18	15	11	250	239	254	4061	4070	4155
87	0	0	0	0	0	0	0	0	0	16	14	10	237	278	250	4170	4079	4110
88	0	0	0	0	0	0	0	2	2	22	14	18	248	254	263	4042	4172	4043
89	0	0	0	0	0	0	2	1	2	14	17	12	270	246	254	4160	4081	4102
90	0	0	0	0	0	0	0	1	1	17	10	21	246	267	241	4092	4069	4017
91	0	0	0	0	0	0	0	1	0	10	17	14	246	261	252	3986	4057	4027
92	0	0	0	0	0	0	0	0	0	8	18	15	246	291	263	3999	4176	4116
93	0	0	0	0	0	1	0	2	2	14	18	15	245	267	218	3955	4157	4035
94	0	0	0	0	0	0	0	2	3	16	17	14	266	274	234	4080	4174	4039
95	0	0	0	0	0	0	3	1	1	15	15	17	262	256	260	4043	4139	4113
96	0	0	0	0	0	0	0	1	0	18	16	16	232	265	249	4035	4045	4183
97	0	0	0	0	0	0	2	0	0	12	10	20	231	247	274	4053	4061	3975
98	0	0	0	0	0	0	0	2	1	18	24	15	241	296	238	4033	4182	4055
99	0	0	0	0	0	0	2	1	0	12	16	10	255	267	230	4061	4239	3982
100	0	0	0	0	0	0	1	1	1	15	20	12	283	256	249	4171	4157	4116
101	0	0	0	0	0	0	1	1	2	21	19	16	233	282	270	4175	4187	4079
102	0	0	0	0	0	0	1	2	1	19	17	22	254	239	252	4071	4042	4092
103	0	0	0	0	1	0	1	1	1	15	22	15	269	263	259	4131	4110	4003
104	0	0	0	0	0	0	0	3	1	16	16	13	280	270	260	4273	4143	4129
105	0	0	0	0	0	0	1	0	0	17	14	11	262	301	247	4056	4074	4203
106	0	0	0	0	0	0	0	0	0	19	9	10	241	259	250	4089	4120	4053
107	0	0	0	0	0	0	1	1	2	16	15	22	254	249	274	4061	4029	4027
108	0	0	0	0	0	0	0	1	1	16	13	17	257	259	288	4091	4125	4108
109	0	0	0	0	0	0	1	1	0	12	23	12	291	244	246	4083	4141	4109
110	0	0	0	0	0	0	0	2	1	17	12	14	257	266	229	4021	4036	4129
111	0	0	0	0	0	0	0	0	1	17	15	21	246	247	275	4098	4178	4004
112	0	0	0	0	0	0	1	2	0	18	11	23	230	266	255	4005	4165	4197
113	0	0	0	1	0	0	2	1	2	12	20	23	247	239	266	3988	4177	4042
114	0	0	0	0	0	0	2	0	0	18	11	20	269	273	270	4080	4140	4199
115	0	0	0	0	0	0	2	1	0	12	12	14	242	236	243	4117	4202	4130
116	0	0	0	0	0	0	0	0	1	9	11	19	242	235	276	4071	4061	4140
117	0	0	0	1	0	0	1	1	1	15	11	9	228	222	261	4188	4090	4116
118	0	0	0	0	0	0	1	1	0	11	16	22	247	282	237	4094	4116	4154
119	0	0	0	0	0	0	2	4	0	16	18	19	240	260	281	4125	4021	3988
120	0	0	0	0	0	0	1	1	0	13	14	16	232	252	247	4010	4142	4068
121	0	0	0	0	0	0	2	3	0	14	22	19	252	240	274	4130	4169	4231
122	0	0	0	0	0	0	0	1	1	22	22	20	259	275	280	4158	4110	4209
123	1	0	0	1	0	0	4	0	0	11	19	11	245	285	268	4138	4117	4018
124	0	0	0	12	5	3	1	0	2	22	17	16	220	249	247	3987	4110	4035
125	0,00813	0	0	0	0	0	110	116	97	14	6	23	253	245	285	4088	4139	4214
126				0,096774	0,040323	0,024194	0	1	0	1968	2023	2008	266	263	247	4069	4209	4090
127							0,88	0,928	0,776	18	14	15	32065	32997	32209	4125	4093	4135
										15,61905	16,05556	15,93651	247	263	237	523097	525378	523456
													252,4803	259,8189	253,6142	4080	4140	4113
																4086,695	4104,516	4089,5

HEXADECIMAL 3																		
Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	0	2	1	14	7	16	271	235	248	4246	4074	4091
1	0	0	0	0	0	0	1	0	0	14	14	23	259	223	270	4097	4037	4099
2	0	0	0	2	0	0	2	0	0	21	11	15	235	241	271	3984	4087	4180
3	0	0	0	0	0	0	3	0	1	10	13	14	260	270	259	4164	4074	4116
4	0	0	0	0	0	1	1	1	3	19	16	12	277	243	255	4014	4123	4097
5	0	0	0	0	1	0	1	1	1	12	15	16	255	262	274	4073	4081	4102
6	0	0	0	0	0	0	0	1	1	13	12	12	244	284	280	4017	3999	4070
7	0	0	0	0	0	0	3	0	1	20	21	10	268	240	263	4112	4137	4159
8	0	0	0	1	0	0	2	1	2	14	11	16	249	244	240	3967	4050	3998
9	0	0	0	0	0	0	1	0	1	16	16	12	229	247	265	4119	4104	4206
10	0	0	0	0	0	0	0	1	2	9	13	16	259	261	241	4005	4175	3959
11	0	0	0	0	0	0	0	3	2	15	11	19	263	256	260	4052	4057	4002
12	0	0	0	0	0	0	0	1	3	17	22	22	268	245	237	4093	4055	4199
13	0	0	0	0	0	1	0	0	3	8	13	23	272	260	267	4057	4067	4047
14	0	0	0	0	0	0	1	0	2	11	11	19	252	234	251	4129	4031	4073
15	0	0	0	0	0	0	0	0	0	19	16	15	297	272	252	4196	4082	4094
16	0	0	0	0	0	0	0	0	0	10	12	13	263	233	212	4089	4160	4057
17	0	0	0	0	0	0	0	1	0	11	12	12	263	232	261	4163	4032	4030
18	0	0	0	0	0	0	1	0	0	14	11	14	283	217	239	4083	3992	4121
19	0	0	0	0	0	0	1	1	2	11	18	25	242	253	246	4219	4041	4056
20	0	0	0	0	0	0	2	1	1	20	10	22	256	241	250	4078	4089	4098
21	0	0	0	0	0	0	1	1	1	13	19	19	236	260	263	4061	4095	4183
22	0	0	0	0	0	0	0	1	1	15	23	16	244	274	279	3989	4083	4161
23	0	0	0	0	0	0	0	1	0	8	16	17	228	232	248	4011	4104	4123
24	0	0	0	0	0	0	2	1	0	20	19	16	222	262	266	3996	4056	4090
25	0	0	0	0	0	0	1	2	3	14	13	17	261	267	242	4096	4130	4028
26	0	0	0	0	0	0	0	0	1	12	13	20	229	261	263	4037	4130	4227
27	0	0	0	1	0	0	1	0	0	14	14	20	282	233	275	3957	4117	4091
28	0	0	0	0	0	0	2	0	0	11	11	20	238	241	246	4174	4094	4141
29	0	0	0	0	0	0	0	0	1	15	8	14	259	247	249	4082	3962	4046
30	0	0	0	0	0	0	2	2	1	14	13	17	245	248	251	4166	4126	4118
31	0	0	0	0	0	0	0	0	0	12	17	13	244	262	247	4080	4178	4065
32	0	0	0	0	0	0	0	0	4	15	21	13	260	263	243	4077	4154	4052
33	0	0	0	0	0	0	0	0	0	16	15	21	228	233	245	4149	4157	4048
34	0	0	0	0	0	0	0	0	0	9	12	20	254	251	278	4003	4085	4099
35	0	0	0	0	0	0	0	2	1	21	15	12	304	226	232	4123	4110	4120
36	0	0	0	0	0	0	0	2	0	10	20	17	242	269	229	4078	4086	4044
37	0	0	0	0	0	0	2	2	0	10	22	14	262	274	266	4115	4117	4071
38	0	0	0	0	0	0	0	0	3	18	17	15	239	258	235	4096	4125	4079
39	0	0	0	0	0	0	1	0	1	10	14	23	254	246	263	4072	4023	4126
40	0	0	0	0	0	0	1	2	2	17	18	12	244	274	252	4103	4074	4108
41	0	0	0	0	0	0	3	1	0	15	16	21	290	245	257	4119	4107	4058
42	0	0	0	0	0	0	1	3	1	15	21	13	250	288	256	4150	4246	4093
43	0	0	0	0	0	0	1	0	0	17	13	12	257	251	239	4081	4051	4107
44	0	0	0	0	0	0	1	0	0	14	12	12	265	240	249	4043	4111	4061
45	0	0	0	0	0	0	1	1	1	14	15	17	249	242	245	4059	4012	3975
46	0	0	0	0	0	0	1	1	0	15	20	14	240	272	243	4145	3991	4036
47	0	0	0	0	0	0	0	1	1	14	20	17	259	273	244	4124	4150	4138
48	0	0	0	0	0	0	0	0	2	10	10	12	238	269	256	4084	4138	4110
49	0	0	0	0	0	0	1	1	2	21	12	21	228	256	265	4089	4169	4133
50	0	0	0	0	0	0	1	0	0	12	19	22	276	235	260	4106	4140	4022
51	0	0	0	0	0	0	1	2	2	15	11	16	260	259	270	4261	4011	4111
52	0	0	0	0	0	1	1	0	1	14	11	22	253	255	257	4118	4148	4118
53	0	0	0	0	0	0	2	0	1	20	17	21	265	235	240	4135	4191	4091
54	0	0	0	0	0	0	1	1	0	16	11	15	246	239	243	4183	4060	4144
55	0	0	0	0	0	0	1	2	2	17	18	19	236	259	300	4034	4103	4186
56	0	0	0	0	0	0	3	0	0	16	18	20	259	267	247	4031	4132	4179
57	0	0	0	0	0	0	1	0	0	17	13	18	277	238	249	4027	4149	4063
58	0	0	0	0	0	0	4	0	1	19	16	14	272	249	256	4111	4052	4113
59	0	0	0	0	0	0	2	0	0	21	14	15	269	255	282	4178	4063	4179
60	0	0	0	1	1	0	2	1	1	10	14	9	242	261	250	4098	4057	4169
61	0	0	0	0	0	0	1	2	2	21	8	23	282	255	247	4109	3963	4062
62	0	0	0	0	0	0	0	1	0	17	18	15	258	245	274	4110	4147	4166
63	0	0	0	0	0	0	0	2	0	21	14	10	291	256	242	4059	4026	4044
64	0	0	0	0	0	0	2	0	1	17	20	19	249	248	234	4098	4065	4045
65	0	0	0	1	0	0	2	1	1	21	12	16	275	235	241	4149	3960	4018
66	0	0	0	1	0	0	3	0	0	15	22	15	242	247	247	3990	4069	4023
67	0	0	0	0	0	0	1	1	2	15	16	16	242	281	250	4034	4204	4078
68	0	0	0	0	0	0	0	1	4	11	18	19	260	266	251	4080	4147	4154
69	0	0	0	0	0	0	1	2	1	29	20	21	272	255	251	4167	4132	4073
70	0	0	0	0	0	0	3	1	1	21	19	13	270	247	273	4043	4130	4073
71	0	0	0	0	0	0	0	2	1	13	12	13	254	249	235	4107	4102	4135
72	0	0	0	0	0	1	2	1	4	19	19	20	257	241	234	3986	4036	3969
73	0	0	0	0	0	0	1	0	1	15	15	20	261	253	274	4112	4062	4137
74	0	0	0	0	0	0	1	0	1	16	11	10	231	254	264	4107	4172	4077
75	0	0	0	0	0	0	0	1	1	12	22	9	249	255	245	3990	4138	3953
76	0	0	0	0	0	0	0	1	1	14	20	8	257	264	231	4140	4160	4185
77	0	0	0	0	0	0	2	1	2	24	15	16	272	250	268	4059	4043	4087
78	0	0	0	0	1	0	1	1	1	18	13	18	273	268	256	4003	4127	4119
79	0	0	0	0	0	0	2	2	0	14	14	13	235	246	241	4079	4149	4045
80	0	0	0	0	0	0	1	1	1	18	16	17	238	228	247	4043	4105	4168
81	0	0	0	0	0	0	1	3	1	16	23	20	260	290	240	4067	4107	4081
82	0	0	0	0	0	0	2	0	1	14	17	15	240	287	278	4001	4193	4110
83	0	0	0	0	0	0	0	0	0	19	15	21	244	271	263	4066	4168	4118
84	0	0	0	0	1	0	1	2	2	15	16	14	253	252	273	3987	4131	4027
85	0	0	0	1	0	0	1	4	1	17	28	19	272	264	236	4101	4038	3992
86	0	0	0	0	0	0	2	0	0	16	15	14	273	311	245	4242	4134	3997
87	0	0	0	0	0	0	0	1	1	21	16	15	232	259	235	4075	4119	4111
88	0	0	0	0	0	0	1	0	2	18	15	12	283	243	260	4044	4083	4156
89	0	0	0	0	0	0	0	2	1	25	16	20	295	251	250	4203	4113	4185
90	0	0	0	0	0	0	1	0	0	15	14	10	269	237	238	4126	4098	4165
91	0	0	0	0	0	0	1	1	0	17	6	20	251	228	256	4223	4020	4074
92	0	0	0	0	0	1	0	0	2	18	13	18	255	254	248	4062	4115	3995
93	0	0	0	0	0	0	0	0	5	13	11	18	230	254	253	4125	4224	4055
94	0	0	0	0	0	0	0	2	3	10	17	26	238	263	257	4087	4138	4200
95	0	0	0	0	0	0	0	0	0	15	14	20	277	261	270	4112	4124	4131
96	0	0	0	0	0	0	0	0	1	16	11	13	274	268	253	4082	4130	4123
97	0	0	0	0	0	0	1	1	1	13	12	17	251	249	243	4037	4156	4071
98	0	0	0	0	0	0	1	1	0	19	12	24	270	247	237	4055	4092	4047
99	0	0	0	0	0	0	1	1	0	10	18	19	248	240	268	4114	4127	3983
100	0	0	0	1	0	0	3	1	1	21	20	9	291	286	267	4116	4202	4050
101	0	0	0	0	0	0	2	1	0	11	19	24	244	242	255	4132	4094	4064
102	0	0	0	0	0	1	0	0	3	19	15	15	253	260	263	4212	4204	4191
103	0	0	0	1	0	0	1	1	2	15	15	17	253	236	240	4141	4022	4033
104	0	0	0	0	0	0	3	0	0	23	16	21	260	269	296	4168	4058	4150
105	0	0	0	0	0	0	1	1	1	17	12	15	266	267	253	4171	4152	4144
106	1	0	0	1	0	0	2	1	3	17	14	23	237	210	245	4063	4002	4112
107	0	0	0	1	0	0	2	2	2	21	20	21	281	261	261	4131	4051	4074
108	0	0	0	0	0	0	1	1	0	13	17	19	249	274	269	4073	4036	4120
109	0	0	0	0	0	0	0	1	1	13	18	15	258	280	251	4121	4177	4055
110	0	0	0	0	0	0	1	1	0	18	16	14	235	225	243	4055	4027	4027
111	0	0	0	0	0	0	0	0	0	18	13	11	270	260	238	4072	4094	4014
112	0	0	0	0	0	0	0	0	1	11	14	18	257	252	239	4134	4084	4035
113	0	0	0	0	0	0	1	0	2	11	12	20	249	270	240	4083	4045	4118
114	0	0	0	0	0	0	0	0	2	16	14	15	230	289	238	4146	4105	4020
115	0	0	0	0	0	0	0	0	0	16	18	19	238	239	256	3977	4126	4122
116	0	0	0	0	0	0	2	0	1	9	22	13	270	288	257	4108	4086	4116
117	0	0	0	0	0	0	1	1	1	16	20	18	250	269	266	4131	4104	4072
118	0	0	0	1	0	0	1	1	0	13	11	17	266	256	257	4164	4039	4101
119	0	0	0	0	0	0	2	0	2	14	13	15	244	237	258	4030	4058	4089
120	0	0	0	0	0	0	3	1	0	17	15	14	264	243	258	4052	4122	4020
121	0	0	0	0	0	1	2	2	2	19	13	14	249	255	261	4136	4135	4066
122	0	0	0	0	0	0	2	1	2	14	26	19	257	228	261	4144	3978	4109
123	**1**	**0**	**0**	0	0	0	0	2	2	14	18	20	262	290	290	3971	4055	4201
124	**0**	**0**	**0**	**13**	**4**	**7**	1	1	1	21	12	17	264	260	253	4150	4062	4119
125	**0,00813**	**0**	**0**	**0**	**0**	**0**	**127**	**102**	**135**	16	13	21	261	241	260	4077	4115	4123
126				**0,104839**	**0,032258**	**0,056452**	**1**	**0**	**1**	**1955**	**1935**	**2102**	262	260	246	4158	4161	4075
127							**1,016**	**0,816**	**1,08**	**14**	**13**	**15**	**32574**	**32281**	**32250**	3977	4099	4186
										15,51587	**15,35714**	**16,68254**	**244**	**260**	**263**	**523765**	**524374**	**523798**
													256,4882	**254,1811**	**253,937**	**4112**	**4130**	**4118**
																4091,914	**4096,672**	**4092,172**

HEXADECIMAL 4																		
Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	1	0	0	13	15	16	233	254	286	4004	3998	4118
1	0	0	0	0	0	0	0	3	0	16	19	10	252	289	248	4095	4065	4040
2	0	0	0	0	0	0	1	0	1	13	14	18	285	279	267	4149	4112	4067
3	0	0	0	0	0	0	1	1	0	18	13	11	234	271	257	4261	4123	4123
4	0	0	0	0	0	0	2	0	1	19	23	15	251	245	247	3956	3993	4132
5	0	0	0	0	0	0	2	2	0	19	16	12	269	281	257	4152	4042	4075
6	0	0	0	0	0	0	3	0	0	29	14	13	239	260	250	4118	4208	4160
7	0	1	0	0	1	0	1	1	4	15	12	21	289	249	272	4116	4111	4166
8	0	0	0	0	1	0	0	3	0	9	11	21	229	259	228	4088	4111	4112
9	0	0	0	0	0	0	3	1	0	19	12	13	246	246	268	4011	4187	3965
10	0	0	0	0	0	0	1	0	2	20	19	21	253	263	270	4105	4181	4098
11	0	0	0	0	0	0	2	1	0	24	14	22	251	254	287	4179	4037	4154
12	0	0	0	0	0	1	0	2	1	13	7	13	243	258	257	4142	4073	4044
13	0	0	0	0	0	0	1	1	3	11	20	17	246	252	250	4025	4248	4152
14	0	0	0	0	0	0	0	2	1	14	22	15	231	279	251	4079	4246	4052
15	0	0	0	0	0	0	0	0	0	18	19	11	267	274	253	4110	4160	4159
16	0	0	0	0	0	0	1	1	0	24	18	11	269	238	236	4019	4042	4087
17	0	0	0	0	0	0	1	1	1	19	16	13	259	243	247	4158	4100	4074
18	0	0	0	0	0	0	2	1	1	21	13	19	260	254	250	4165	4027	4131
19	0	0	0	0	0	0	1	0	0	23	8	8	293	268	260	4132	4195	4197
20	0	0	0	0	0	0	1	1	1	12	11	18	268	249	234	4122	4013	4097
21	0	0	0	0	0	0	0	2	0	12	18	8	234	264	224	4106	4139	4053
22	0	0	0	0	0	0	1	1	0	20	20	12	284	261	257	4160	4104	4069
23	0	0	0	0	0	0	3	1	0	14	17	10	274	278	240	4166	4035	4148
24	0	0	0	0	0	0	1	2	0	20	13	9	236	241	232	4093	4108	4058
25	0	0	0	0	0	0	0	4	2	13	18	14	277	259	227	4155	4124	4000
26	0	0	0	0	0	0	0	0	0	18	19	17	261	229	264	4012	4098	4109
27	0	0	0	0	0	0	2	0	2	12	16	17	241	251	248	4177	4007	4048
28	0	0	0	0	0	0	0	0	3	11	10	18	244	266	250	4076	4113	4088
29	0	0	0	0	0	0	1	0	0	17	11	17	242	254	278	4213	4151	4023
30	0	0	0	0	0	0	2	2	0	19	12	13	258	224	255	4040	4030	4084
31	0	0	0	0	0	0	0	1	0	15	9	22	267	258	256	4042	4072	4101
32	0	0	0	0	0	0	1	0	0	12	11	18	269	249	253	4169	4009	4123
33	0	0	0	1	0	0	2	0	1	16	13	13	225	244	258	4182	4126	4110
34	0	0	1	0	0	1	1	1	2	15	19	14	241	241	245	3970	4060	4096
35	0	0	0	0	0	1	0	0	1	12	10	17	244	269	261	4047	4199	4126
36	0	0	0	0	0	0	2	1	1	16	14	12	248	245	250	4017	4056	4089
37	0	0	0	0	0	0	1	0	1	21	17	16	256	271	267	4133	4064	4131
38	0	0	0	0	0	0	0	0	1	20	19	15	249	265	262	4160	4192	4152
39	0	0	0	0	0	0	0	0	0	15	12	12	246	253	240	4082	4183	4083
40	0	0	0	0	0	0	3	1	2	12	18	11	226	240	252	4081	4095	3984
41	0	0	0	0	0	0	1	1	0	11	14	13	245	264	237	4004	4104	4069
42	0	0	0	0	0	0	1	0	0	21	21	16	234	241	228	4043	4200	4093
43	0	0	0	0	0	0	1	0	2	17	14	12	287	236	252	4127	3973	3944
44	0	0	0	0	0	0	0	1	0	20	9	17	268	242	245	4209	3981	4122
45	0	0	0	0	0	0	0	0	2	11	18	11	236	249	253	4079	4119	4091
46	0	0	0	0	0	0	1	1	0	14	18	12	263	259	262	4190	4105	4067
47	0	0	0	0	0	0	2	1	1	16	14	19	248	269	268	4045	4162	4090
48	0	0	0	0	0	0	0	0	3	14	15	14	251	247	264	4125	4152	4122
49	0	0	0	0	0	0	0	1	1	10	22	19	273	259	266	4075	4076	4137
50	0	0	0	0	0	0	2	0	1	24	16	12	244	265	253	4034	4218	4088
51	0	0	0	0	0	0	3	0	0	16	18	21	265	253	252	4116	4002	4049
52	0	0	0	0	0	0	2	0	0	20	11	10	254	242	231	4068	4077	4120
53	0	1	0	0	1	0	2	2	1	22	13	15	249	242	249	4028	4158	3988
54	0	0	0	0	1	0	0	2	0	23	15	13	263	255	233	4185	4010	4029
55	0	0	0	0	0	0	1	2	0	16	9	18	257	241	246	4030	4092	4062
56	0	0	0	0	0	0	1	0	2	21	17	16	245	225	247	4181	4089	4024
57	0	0	0	0	0	0	0	0	1	14	6	17	237	243	257	4158	3939	3975
58	0	0	0	0	0	0	3	0	2	23	16	24	279	237	259	4097	4074	4046
59	0	0	0	0	0	0	2	1	0	26	16	20	265	223	257	4069	4057	4142
60	0	0	0	0	1	0	2	1	1	17	16	21	288	267	245	4162	3988	4117
61	0	0	0	0	0	0	0	2	0	22	17	19	274	274	264	4074	4235	4245
62	0	0	0	0	0	0	0	0	2	13	11	10	239	239	253	4075	4062	4126
63	0	0	0	0	0	0	1	0	0	12	17	18	254	265	245	4132	4213	4227
64	0	0	0	0	1	0	1	1	0	15	16	15	268	275	255	4050	4146	4000
65	0	0	0	0	0	0	0	2	0	13	15	22	292	261	265	4186	4061	4126
66	0	0	0	0	0	0	0	2	1	9	17	11	238	252	243	4087	4123	4161
67	0	0	0	0	0	0	0	0	0	14	15	14	253	228	227	4077	4053	4108
68	0	0	0	0	0	0	0	0	1	15	11	10	250	273	255	4079	4103	4008
69	0	0	0	0	0	1	0	0	1	11	17	19	247	251	267	4132	4129	4077
70	0	0	0	0	0	0	3	2	2	21	12	20	274	249	259	4099	4022	4251
71	0	0	0	0	0	0	1	2	2	18	20	24	258	280	270	4120	4077	4134
72	0	0	0	0	0	0	1	0	1	20	17	20	236	259	270	4057	4238	4187
73	0	0	0	0	0	0	1	0	0	16	11	18	270	243	267	4011	4079	4102
74	0	0	0	0	0	0	0	2	1	24	16	18	268	267	244	4138	4063	4073
75	0	0	0	0	0	1	2	1	1	14	16	19	263	229	259	4138	4091	4024
76	0	0	0	0	0	0	3	0	3	15	17	18	244	277	265	4078	4198	4171
77	0	0	0	0	0	0	1	0	0	22	18	19	248	249	235	4138	4127	4121
78	0	0	0	0	0	0	1	0	0	15	11	9	266	250	247	4089	4052	4100
79	0	0	0	0	0	0	0	0	0	14	14	17	260	259	267	4068	4143	4190
80	0	0	0	0	0	0	0	0	4	11	13	17	231	277	239	4008	4121	4124
81	0	0	0	0	0	0	0	0	0	25	17	17	261	230	229	3996	4027	4001
82	0	0	0	0	0	0	2	0	0	18	9	8	254	248	246	4048	3991	4057
83	0	0	0	0	0	0	0	1	0	20	13	15	279	244	256	4161	4079	4071
84	0	0	0	0	1	0	1	3	1	16	15	9	278	206	228	3985	4061	4135
85	0	0	0	0	0	0	3	1	1	19	14	20	274	254	233	4106	4031	4011
86	0	0	0	0	0	0	0	0	0	18	16	18	260	245	270	4176	4068	4077
87	1	0	0	1	0	0	1	0	0	15	12	10	235	263	234	4017	4023	4114
88	0	0	0	2	0	0	2	1	0	13	19	16	240	263	244	4063	4144	4000
89	0	0	0	0	0	0	2	2	0	13	10	11	259	265	280	3997	4084	4219
90	0	0	0	0	0	0	0	3	0	15	17	10	245	246	236	4016	4114	4159
91	0	0	0	1	0	0	2	0	0	15	21	15	273	256	272	4137	4035	4045
92	0	0	0	0	0	0	2	0	0	18	20	17	238	296	246	4133	4098	4114
93	0	0	0	0	0	0	2	1	1	14	13	16	265	267	258	4008	4164	4057
94	0	0	0	0	0	0	3	1	0	20	17	15	289	256	246	4092	3968	4022
95	0	0	0	0	0	0	0	3	0	18	10	12	255	235	249	4036	4075	3946
96	0	0	0	1	0	0	1	2	3	13	21	19	260	245	245	4162	4015	4081
97	0	0	0	1	0	0	4	1	1	13	21	26	273	306	289	4025	4248	4114
98	0	0	0	0	0	0	2	0	2	19	10	15	215	271	279	4044	4122	4034
99	0	0	0	0	0	0	1	2	2	21	11	20	300	213	252	4018	4004	4226
100	0	0	0	0	0	0	0	1	1	16	13	22	255	225	261	4097	3948	4080
101	0	0	0	0	0	0	0	0	0	13	12	16	256	252	238	4075	4053	3987
102	0	0	0	0	0	0	0	0	1	11	15	15	253	271	250	4168	4114	4238
103	0	0	0	0	0	0	2	0	1	11	12	12	286	271	252	4193	4041	4089
104	0	0	0	0	0	0	1	0	0	12	20	16	275	257	233	4084	4199	4175
105	0	0	0	0	0	0	0	1	3	11	9	17	257	241	244	4070	4170	3995
106	0	0	0	0	0	0	0	0	1	23	16	18	262	235	255	4216	4059	4187
107	0	0	0	0	0	0	1	0	1	17	15	13	271	247	253	4087	4107	4176
108	0	0	0	0	0	0	0	2	0	12	23	17	242	257	237	4106	4032	4063
109	0	0	0	0	0	0	1	0	1	16	14	16	267	233	249	4040	4070	4074
110	0	0	0	0	0	0	2	2	1	22	11	10	283	237	242	4217	4109	4078
111	0	0	0	0	0	0	1	1	1	16	16	16	256	244	239	4112	4060	4080
112	0	0	0	0	0	0	1	2	1	15	22	17	272	291	256	4178	4001	4023
113	0	0	0	0	0	0	0	2	2	13	23	15	231	267	271	4147	4200	3987
114	0	0	0	0	0	0	0	0	0	12	18	16	219	261	237	4076	4090	4004
115	0	0	0	0	0	0	1	0	0	13	11	20	239	236	248	3962	4094	4136
116	0	0	0	0	0	0	0	0	2	12	13	12	249	235	251	4175	4099	4000
117	0	0	0	0	0	0	0	2	0	17	13	19	265	231	261	4157	4116	4088
118	0	0	0	0	0	0	0	2	1	12	22	20	241	267	283	4093	4050	4171
119	0	0	0	0	0	0	0	1	2	8	16	22	277	273	288	4102	4047	4061
120	0	0	0	0	0	0	2	0	2	13	14	14	237	238	262	4125	4100	4145
121	0	0	0	0	0	0	0	2	0	12	15	19	271	239	249	4238	4060	4209
122	0	0	0	0	1	1	0	1	1	16	16	17	279	279	268	4119	4164	4183
123	1	2	1	0	0	0	0	2	2	15	19	11	256	249	217	4114	4066	4062
124	0	0	0	7	8	6	1	0	1	13	20	16	235	260	230	3924	4030	4089
125	0,00813	0,01626	0,00813	0	0	0	124	106	104	16	17	14	226	244	262	4057	4195	4053
126				0,056452	0,064516	0,048387	0	0	0	2039	1912	1969	251	282	243	4072	4103	4146
127							0,992	0,848	0,832	13	16	17	32537	32274	32065	4185	4173	4117
										16,18254	15,1746	15,62698	251	249	257	524537	524050	523987
													256,1969	254,126	252,4803	4079	4060	4000
																4097,945	4094,141	4093,648

HEXADECIMAL 5

Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	0	0	0	12	13	17	224	279	255	4173	4037	4109
1	0	0	0	0	0	0	0	0	0	8	14	12	266	252	256	4002	4139	4197
2	0	0	0	0	1	0	1	2	0	9	11	22	220	248	251	4107	4038	4022
3	0	0	0	0	0	0	2	1	0	16	17	13	242	246	254	4016	4126	4091
4	0	0	0	0	0	0	1	1	3	14	13	15	247	262	271	4106	4147	4144
5	0	0	0	0	0	0	0	1	1	14	19	16	269	281	241	4050	4086	4139
6	0	0	0	0	0	0	0	1	1	13	15	17	221	240	257	4058	4046	4083
7	0	0	0	0	0	0	0	0	1	20	13	15	260	225	247	4103	4095	4105
8	0	0	0	0	0	0	0	0	1	14	18	17	232	290	273	4084	4152	4112
9	0	0	0	0	0	1	0	1	2	19	12	17	263	254	262	4034	4075	4100
10	0	0	0	0	0	0	3	1	1	16	16	19	271	253	239	4036	4073	4080
11	0	0	0	0	0	0	2	2	2	18	12	18	265	270	270	4009	4147	4150
12	0	0	0	0	0	0	3	2	2	19	13	16	251	285	269	4160	4231	4103
13	0	0	0	0	0	0	2	0	3	19	13	16	252	244	254	4102	4079	4070
14	0	0	0	0	1	1	2	1	1	23	16	18	259	258	276	4131	4154	4158
15	0	0	0	0	0	0	0	3	1	16	12	20	247	283	289	4105	4163	4177
16	0	0	0	0	0	1	2	0	1	10	14	13	249	235	283	4078	4086	3936
17	0	0	0	0	0	0	2	1	1	21	13	22	253	258	260	4041	4064	4055
18	0	0	0	0	0	1	0	0	3	18	22	18	258	249	288	4034	4124	4135
19	0	0	0	0	0	0	2	1	4	12	13	27	263	247	271	4119	4148	4047
20	0	0	0	0	0	0	0	0	0	12	9	22	277	251	271	4131	4039	4079
21	0	0	0	0	0	0	0	1	0	8	17	15	245	255	246	4108	3969	4173
22	0	0	0	0	0	0	2	0	1	18	14	13	244	260	262	4102	4201	4090
23	0	0	0	0	0	0	3	3	0	21	20	14	242	257	254	4140	4122	4100
24	0	0	0	0	0	0	0	0	1	15	10	11	235	245	272	4028	4234	4182
25	0	0	0	0	0	0	1	2	1	17	11	20	279	226	277	4011	4083	4142
26	0	0	0	0	0	0	1	2	0	16	12	12	264	245	256	4077	4160	4092
27	0	0	0	0	0	0	0	0	1	12	13	14	227	250	245	4161	4013	4166
28	0	0	0	0	0	0	0	0	0	15	17	13	229	258	235	4015	4135	4051
29	0	0	0	0	0	0	0	0	2	12	15	11	248	251	264	4147	4106	4087
30	0	0	0	0	0	0	2	0	1	14	13	20	267	249	250	4084	4078	4074
31	0	0	0	0	0	0	1	0	0	24	9	13	249	231	233	4041	4130	4060
32	0	0	0	0	0	0	0	1	0	12	10	12	254	228	263	4123	3990	4245
33	0	0	0	0	0	0	1	0	0	17	15	10	238	251	260	4070	4027	4111
34	0	0	0	1	0	0	2	1	0	19	14	17	275	268	239	4106	4188	4057
35	0	0	0	0	0	0	1	0	1	19	14	15	242	244	294	4068	4185	4118
36	0	0	0	1	0	0	1	1	1	12	21	11	299	270	255	4090	4071	4088
37	0	0	0	0	0	0	1	1	0	19	11	17	277	269	264	4266	4121	4029
38	0	0	0	0	0	0	0	1	1	17	12	11	274	275	255	4012	4035	4154
39	0	0	0	0	0	0	1	2	2	13	19	13	252	242	247	4089	4195	4070
40	0	0	0	0	0	0	0	2	2	12	23	23	262	274	226	3979	4120	4054
41	0	0	0	0	0	0	1	0	0	13	15	17	271	239	267	4211	4084	4002
42	0	0	0	0	0	0	0	0	0	12	13	14	247	273	284	4082	4108	4128
43	0	0	0	0	1	0	1	3	0	15	27	7	230	269	254	3966	4097	4057
44	0	0	0	0	0	0	1	2	0	20	17	10	260	287	239	4044	4152	4057
45	0	0	0	0	0	0	0	1	0	20	16	15	272	275	236	4162	4029	4080
46	0	0	0	1	0	0	4	1	2	16	14	13	271	258	252	4048	4170	4141
47	0	0	0	0	0	0	1	1	0	22	13	11	261	267	257	4074	4130	4025
48	0	0	0	0	0	0	1	0	0	19	22	17	250	255	270	4054	4167	4201
49	0	0	0	0	0	0	0	1	1	12	15	19	266	259	242	4117	4142	4101
50	0	0	0	1	0	0	2	1	1	22	12	17	218	258	224	4083	4024	4159
51	0	0	0	0	0	0	2	0	0	16	24	15	261	232	268	3987	4068	4121
52	0	0	0	1	0	0	1	1	2	19	14	18	269	267	307	4077	4010	4127
53	0	0	0	0	0	0	2	1	0	12	15	19	244	262	224	4088	4171	4145
54	0	0	0	0	0	0	0	0	1	19	16	13	241	272	262	4110	4196	4054
55	0	0	0	0	0	0	1	2	0	16	16	11	272	254	245	4054	4051	4073
56	0	0	0	0	0	0	1	1	1	15	19	10	255	263	253	4165	4139	4089
57	0	0	0	0	0	0	0	2	3	14	15	19	260	246	249	4041	4113	4079
58	0	0	0	2	0	0	2	1	1	17	23	20	271	268	268	4163	4172	4119
59	0	0	0	0	0	0	3	3	1	19	18	13	240	284	247	4106	4233	4086
60	0	0	0	0	0	0	1	1	0	21	19	15	236	254	235	4075	4148	4062
61	0	0	0	0	0	0	1	1	2	17	23	12	266	254	261	4072	4026	4150
62	0	0	0	0	1	0	1	4	1	18	18	19	243	268	284	4011	4112	4053
63	0	0	0	0	0	0	0	2	1	13	23	15	247	267	253	3989	4177	4125
64	0	0	0	0	1	0	0	2	0	20	15	11	276	240	226	4201	4026	4029
65	0	0	0	0	0	0	1	3	3	9	28	19	234	267	239	4106	4132	4053
66	0	0	0	0	0	0	0	1	3	16	22	19	256	274	263	4040	4213	4043
67	0	0	0	0	0	0	0	0	2	13	15	25	256	256	278	4062	4064	4153
68	0	0	0	0	0	0	2	0	1	13	12	11	242	246	261	4199	4152	4072
69	0	0	0	0	0	0	1	0	3	20	9	20	247	253	245	4015	4088	4103
70	0	0	0	0	0	0	0	1	1	18	15	15	274	260	242	4125	4070	4034
71	0	0	0	0	0	0	0	3	1	15	17	16	260	276	248	4124	4175	4092
72	0	0	0	0	0	0	0	1	1	11	18	14	282	272	251	4142	4199	4128
73	0	0	0	0	0	0	1	1	1	11	9	10	265	240	230	4175	4138	4116
74	0	0	0	0	0	0	0	1	1	20	20	16	280	259	248	4065	4102	4048
75	0	0	0	0	0	0	2	1	1	18	16	14	277	262	251	4227	4099	4028
76	0	0	0	0	0	0	1	1	1	16	14	19	252	239	243	4048	4090	4082
77	0	0	0	0	0	0	1	0	0	17	13	15	263	253	252	4070	4013	4070
78	0	0	0	0	0	0	2	0	1	16	8	17	260	247	268	4111	4109	4142
79	0	0	0	0	0	0	0	0	0	15	16	21	254	250	270	4069	4062	4112
80	0	0	0	0	0	0	0	2	2	17	12	12	270	252	245	4142	4069	4096
81	0	0	0	0	0	0	0	1	1	14	16	17	238	246	262	4056	3946	3978
82	0	0	0	0	0	0	1	0	0	17	10	21	261	236	284	4088	4155	4106
83	0	0	0	0	0	0	0	1	0	30	12	24	278	260	261	4132	4022	4053
84	0	0	0	0	0	0	1	2	0	11	14	10	272	267	248	4112	4231	4007
85	0	0	0	0	0	0	0	1	2	9	23	19	234	229	229	4053	4100	4114
86	0	0	0	0	0	0	1	0	1	16	17	21	240	256	251	4090	4122	4077
87	0	0	0	0	0	0	0	0	1	14	13	18	253	275	264	4039	4119	4118
88	0	0	0	0	0	0	1	1	2	20	20	19	256	283	264	4053	4161	4123
89	0	0	0	0	0	0	0	3	2	15	18	17	265	265	276	4037	4169	4145
90	0	0	0	0	0	0	1	2	1	11	13	19	259	234	243	4154	4056	4097
91	0	0	0	0	0	0	1	1	1	24	15	15	267	231	267	4045	4116	4058
92	0	0	0	0	0	1	1	2	2	16	12	15	258	256	259	4007	4151	4085
93	0	0	0	0	0	0	2	0	2	15	21	22	260	254	253	4120	4110	4116
94	0	0	0	0	1	0	0	1	0	14	11	13	252	251	241	4096	4084	4077
95	0	0	0	0	0	0	0	1	0	9	13	5	260	249	238	4029	4157	4113
96	0	0	0	0	0	0	0	1	0	21	23	14	264	287	250	4067	4026	4181
97	0	0	0	0	0	0	0	2	1	15	18	24	268	271	247	4105	4262	4077
98	0	0	0	0	0	0	0	0	1	20	15	13	254	247	263	4088	4171	4231
99	0	0	0	0	0	0	1	2	0	12	18	22	268	240	260	4150	4033	4203
100	0	0	0	0	0	0	3	1	0	21	19	18	254	259	257	4201	3950	4164
101	0	0	0	0	0	1	0	1	1	18	14	12	253	259	268	4087	4090	4090
102	0	0	0	0	0	0	1	0	2	25	18	22	279	281	256	4098	4207	4096
103	0	0	0	0	0	1	2	2	4	19	18	16	261	258	262	4115	4203	4002
104	0	0	0	0	0	0	2	1	1	21	16	21	262	255	246	4063	4098	4016
105	0	0	0	0	0	0	0	0	2	18	8	18	257	249	280	4185	4090	4086
106	0	0	0	0	0	0	1	5	0	10	26	17	252	245	250	4056	4139	4104
107	0	0	0	0	0	0	0	3	2	17	16	15	243	255	247	4067	4229	4006
108	0	0	0	0	0	0	2	1	1	20	24	27	280	261	250	4128	4055	4193
109	0	0	0	0	0	0	1	2	1	22	10	18	252	266	290	4089	4185	4017
110	0	0	0	0	0	0	0	1	1	13	25	20	236	242	278	4021	4173	4162
111	0	0	0	0	0	0	1	0	1	16	12	16	228	257	246	4051	4056	4186
112	0	0	0	0	0	0	0	1	0	9	22	12	231	309	256	3963	4184	4009
113	0	0	0	0	0	0	3	2	0	18	14	14	259	265	264	4022	4191	4162
114	0	0	0	1	0	0	2	0	1	20	17	18	205	271	266	4151	4057	4105
115	0	0	0	0	0	0	1	2	1	20	18	10	255	248	250	4021	4122	3994
116	0	0	0	0	0	0	1	1	0	12	8	20	241	265	268	3949	4088	4094
117	0	0	0	0	0	0	0	1	0	13	13	12	233	223	274	4091	4117	4153
118	0	0	0	0	0	0	1	2	1	15	16	19	286	267	263	4073	4168	4031
119	0	0	0	0	1	0	1	3	1	22	13	14	283	262	275	4176	4099	4159
120	0	0	0	0	0	0	0	1	0	12	23	12	262	256	256	4105	4189	4125
121	0	0	0	0	0	0	0	1	0	12	21	9	253	237	236	4088	4093	4026
122	0	0	0	0	0	0	2	1	1	18	23	16	246	269	243	4056	4085	4001
123	0	0	0	1	0	0	1	0	1	14	18	15	273	273	248	4098	4143	4097
124	0	0	0	9	7	7	1	1	0	16	15	15	267	245	254	4211	4143	4111
125	0	0	0	0	0	0	112	135	120	16	11	22	254	235	246	4067	4044	4080
126				0,072581	0,056452	0,056452	0	1	1	2027	1994	2024	228	271	264	4070	3991	4050
127							0,896	1,08	0,96	12	13	15	32430	32655	32628	4086	4055	4091
										16,0873	15,8254	16,06349	260	258	264	523005	526367	524239
													255,3543	257,126	256,9134	4106	4139	4080
																4085,977	4112,242	4095,617

HEXADECIMAL 6

Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	1	0	1	1	5	15	21	15	249	249	244	4042	4176	4007
1	0	0	0	0	0	0	3	2	0	20	23	20	254	254	274	4056	4043	4046
2	0	0	0	0	0	0	0	2	0	17	18	17	251	251	263	4168	4080	4009
3	0	0	0	0	0	0	0	1	0	14	20	19	241	241	259	4122	4104	4072
4	0	0	0	1	0	0	3	0	1	23	15	14	262	262	276	4017	4150	4159
5	0	0	0	0	0	1	1	0	1	15	14	20	258	258	275	4099	4048	4090
6	0	0	0	0	0	0	2	0	1	19	10	19	247	247	256	4062	4132	4036
7	0	0	0	0	0	0	1	2	0	20	12	13	259	259	243	4113	4113	4110
8	0	0	0	0	0	0	1	1	1	21	11	16	249	249	269	4099	4096	4034
9	0	0	0	0	0	0	0	1	1	19	18	22	280	280	247	4170	4104	4067
10	0	0	0	0	0	0	0	3	0	14	18	16	217	217	255	4061	4045	4002
11	0	0	0	0	0	0	2	1	1	20	15	16	275	275	250	4071	3965	4123
12	0	0	0	0	0	0	3	3	0	15	15	8	259	259	261	4092	4082	4127
13	0	0	0	0	0	0	2	2	0	13	16	12	260	260	260	4038	3984	4125
14	0	0	0	0	0	0	2	2	0	19	18	13	254	254	266	4156	4012	4154
15	0	0	0	0	0	0	3	1	0	22	19	11	257	257	280	4082	4196	4085
16	0	0	0	0	0	0	0	2	2	15	11	13	265	265	246	4150	4159	4081
17	0	0	0	0	0	0	2	1	3	17	19	17	243	243	254	4060	4125	4050
18	0	0	0	0	0	0	3	0	0	21	16	12	255	255	253	4208	4098	4092
19	0	0	0	0	0	0	1	1	1	16	19	19	248	248	242	4110	4171	4118
20	0	0	0	0	0	0	0	2	0	19	11	17	283	283	288	4101	4080	4180
21	0	0	0	0	0	0	1	0	0	14	19	13	225	225	239	4168	4143	4004
22	0	0	0	0	0	0	1	1	0	13	12	5	251	251	213	4110	4030	4115
23	0	0	0	0	0	0	1	0	1	16	19	15	266	266	249	4146	4056	4060
24	0	0	0	0	0	0	0	2	1	13	13	18	231	231	255	4033	4104	4069
25	0	0	0	0	0	0	0	1	0	13	16	14	252	252	240	3977	4090	4111
26	0	0	0	0	1	0	1	1	1	15	13	9	275	275	234	4113	4075	3983
27	0	0	0	0	1	0	0	2	1	14	10	19	267	267	288	4041	4066	4038
28	0	0	0	0	0	0	1	3	0	16	14	14	250	250	244	4053	4074	4176
29	0	0	0	0	0	0	1	0	0	17	16	19	255	255	264	4014	4147	4194
30	0	0	0	0	0	0	2	2	2	18	20	15	255	255	248	4053	3966	4253
31	0	0	0	0	0	0	0	1	2	12	19	17	245	245	251	4052	4019	4107
32	0	0	0	0	0	0	1	1	0	20	8	23	287	287	278	4143	4151	4196
33	0	0	0	0	0	0	2	0	0	23	13	17	247	247	256	4047	4089	4179
34	0	0	0	0	0	0	1	3	0	21	22	15	268	268	277	4146	4154	4079
35	0	0	0	0	0	0	1	0	0	12	18	18	252	252	264	4061	4032	4150
36	0	0	0	0	0	0	0	1	2	13	11	16	263	263	260	4181	4201	4089
37	0	0	0	0	0	0	0	3	1	15	23	22	255	255	257	4099	4032	4064
38	0	0	0	0	0	0	1	0	1	16	16	13	269	269	269	4176	4151	4049
39	0	0	0	0	1	0	0	2	1	13	14	14	251	251	247	4036	4105	4170
40	0	0	0	0	0	0	1	2	0	17	15	11	254	254	236	4138	4036	4154
41	0	0	0	0	0	0	1	1	1	15	17	14	253	253	249	4106	4144	3918
42	0	0	0	0	0	0	0	0	1	21	21	11	266	266	236	4133	3982	3950
43	0	0	0	1	0	0	1	0	0	17	9	17	266	266	290	4071	4095	4137
44	0	0	0	0	0	0	1	0	1	5	16	18	240	240	249	4144	4073	4026
45	0	0	0	0	0	0	0	0	1	16	14	20	255	255	255	3894	4067	4184
46	0	0	0	0	0	0	1	0	1	17	11	15	263	263	234	4066	4114	4058
47	0	0	0	0	0	1	1	3	2	16	19	17	265	265	240	4040	4151	3993
48	0	0	0	0	0	0	1	0	1	19	11	11	241	241	274	4189	4069	4072
49	0	0	0	1	0	0	2	1	0	18	13	21	256	256	254	4074	4154	4143
50	0	0	0	0	0	0	3	0	2	17	19	19	270	270	269	4160	4144	4022
51	0	0	0	0	0	0	1	4	1	19	25	19	262	262	241	4119	4162	4072
52	0	0	0	0	1	0	1	2	0	12	17	18	263	263	267	4149	4147	3972
53	0	0	0	0	0	0	2	3	1	20	20	11	263	263	239	4124	4115	4009
54	0	0	0	1	0	0	2	0	2	23	13	17	273	273	224	4106	3960	4059
55	0	0	0	0	0	0	1	0	1	20	20	19	259	259	273	4120	4063	4097
56	0	0	0	0	0	0	0	3	1	18	14	15	245	245	251	4008	4110	4044
57	0	0	0	0	0	0	0	3	1	13	18	10	277	277	263	4113	4091	4145
58	0	0	0	0	0	0	0	2	0	20	22	21	223	223	290	4083	4147	4109
59	0	0	0	0	0	0	1	2	0	13	16	17	260	260	275	4119	4075	4252
60	0	0	0	0	0	0	0	1	1	15	18	17	251	251	244	4076	4101	4108
61	0	0	0	1	0	0	2	1	1	21	16	22	258	258	296	4195	4243	4039
62	0	0	0	0	0	1	2	2	2	21	19	22	266	266	254	4091	4133	4148
63	0	0	0	0	0	0	1	3	1	8	20	10	220	220	253	4032	4127	4065
64	0	0	0	0	0	0	0	0	2	14	20	16	231	231	273	4070	4015	4141
65	0	0	0	0	0	0	1	0	1	15	24	16	228	228	278	4038	4063	4112
66	0	0	0	0	0	0	0	0	0	19	16	20	256	256	266	3980	4061	4080
67	0	0	0	0	0	0	0	1	0	13	15	14	277	277	250	4192	4022	4138
68	0	0	0	0	0	0	0	2	1	17	13	17	272	272	244	4053	4003	4086
69	0	0	0	0	0	0	0	0	1	11	12	20	239	239	241	4144	4156	4038
70	0	0	0	1	0	0	3	2	2	21	16	13	238	238	269	4122	4079	4082
71	0	0	0	0	0	1	2	1	2	19	21	20	250	250	256	4180	4029	4116
72	0	0	0	0	0	0	1	1	2	19	25	13	246	246	247	4137	4030	4101
73	0	0	0	0	0	0	0	0	0	17	20	17	256	256	237	3991	4174	4058
74	0	0	0	0	0	0	2	0	1	11	17	12	231	231	248	4211	4142	4196
75	0	0	0	0	0	0	1	2	2	14	14	17	264	264	249	4039	4063	4026
76	0	0	0	0	0	0	1	2	2	21	15	16	289	289	237	4202	4061	3984
77	0	0	0	0	0	0	1	2	2	19	20	20	254	254	280	4082	4235	4152
78	0	0	0	0	0	0	2	1	1	15	20	16	269	269	262	4153	4147	4101
79	0	0	0	0	1	0	0	1	3	12	21	23	245	245	269	4030	4089	4059
80	0	0	0	0	1	0	1	4	1	18	20	14	243	243	241	4032	4049	4086
81	0	0	0	0	1	0	1	3	1	15	21	19	287	287	286	4046	4115	4098
82	0	0	0	0	0	0	0	1	1	11	16	20	243	243	256	4185	4153	4042
83	0	0	0	0	0	0	0	0	1	9	24	17	215	215	268	4039	4065	4088
84	0	0	0	0	0	0	1	0	0	16	14	14	242	242	256	4055	4050	4153
85	0	0	0	0	0	0	0	0	1	19	15	17	230	230	242	4055	4164	4119
86	0	0	0	1	0	0	1	1	0	19	13	15	267	267	271	4011	4020	4042
87	0	0	0	0	0	0	1	0	0	12	8	5	248	248	248	3985	4105	4087
88	0	0	0	0	0	0	0	0	1	10	17	21	244	244	277	4117	4092	4155
89	0	0	0	0	0	0	1	0	1	19	14	12	267	267	244	4107	4099	4130
90	0	0	0	0	0	1	2	0	3	23	10	21	249	249	273	4088	3986	4079
91	0	0	0	0	0	0	0	1	2	18	19	16	259	259	275	4209	4198	4136
92	0	0	0	0	0	0	2	0	0	20	17	20	277	277	239	4112	4039	4132
93	0	0	0	0	1	0	0	1	0	15	7	8	241	241	268	4102	3983	4083
94	0	0	0	0	0	0	1	1	1	10	13	12	254	254	238	4024	4136	3985
95	0	0	0	0	0	0	1	0	0	22	17	13	283	283	266	4153	4101	4187
96	0	0	0	0	0	0	1	0	1	11	12	14	255	255	255	4022	4110	4097
97	0	0	0	0	0	0	1	1	2	15	15	17	243	243	229	4119	3990	4103
98	0	0	0	0	0	0	1	0	2	24	22	20	257	257	262	3982	4143	4090
99	0	0	0	0	0	0	1	1	1	16	19	14	261	261	276	4091	4078	4135
100	0	0	0	0	0	0	2	0	0	15	11	9	252	252	255	3995	4117	4133
101	0	0	0	0	0	0	0	0	0	13	15	14	259	259	229	4097	4040	3996
102	0	0	0	0	0	0	1	1	1	17	20	13	261	261	229	4094	4205	4080
103	0	0	0	0	0	1	0	0	2	14	16	15	273	273	234	4010	3998	4027
104	0	0	0	0	0	0	4	2	1	23	13	11	252	252	252	3977	4033	4055
105	0	0	0	0	1	0	1	3	0	15	23	18	247	247	261	4129	4056	4071
106	0	0	0	0	0	0	0	2	1	13	18	12	254	254	269	4093	4113	4057
107	0	0	0	0	0	0	1	0	0	16	12	12	296	296	251	4070	4099	4104
108	0	0	0	0	0	0	1	0	1	17	13	9	258	258	253	4146	4155	4024
109	0	0	0	0	0	0	2	3	0	13	17	22	242	242	250	4159	4151	4000
110	0	0	0	0	0	0	3	1	1	21	21	17	261	261	280	4158	4141	4094
111	0	0	0	0	1	0	2	4	2	20	18	10	267	267	220	4116	4081	4047
112	0	0	0	0	0	0	0	1	1	19	17	16	267	267	243	4181	3991	4001
113	0	0	0	0	0	0	0	1	0	14	15	20	268	268	271	4077	3945	4118
114	0	0	0	0	0	0	0	2	0	12	19	15	252	252	277	4138	4113	4124
115	0	0	0	0	0	0	1	1	0	20	21	12	270	270	244	4245	4107	4062
116	0	0	0	0	0	0	2	1	2	19	23	15	266	266	246	4158	4162	4153
117	0	0	0	1	0	0	1	1	1	24	13	20	302	302	243	4161	4064	4112
118	0	0	0	0	0	0	3	1	1	16	13	17	253	253	256	4173	4103	4078
119	0	0	0	0	0	0	1	0	2	28	19	17	280	280	278	4104	4126	4160
120	0	0	0	0	1	0	0	1	1	16	15	16	284	284	262	4123	4025	4066
121	0	0	0	0	0	0	0	1	1	14	12	16	255	255	231	4172	4073	4043
122	0	0	0	0	0	0	0	2	1	10	24	13	228	228	229	4102	4096	4003
123	**0**	**0**	**0**	0	0	0	0	1	1	12	22	17	258	258	244	4060	4226	4072
124	**0**	**0**	**0**	**8**	**12**	**6**	0	1	2	16	15	21	244	244	270	4150	4224	4152
125	**0**	**0**	**0**	**0**	**0**	**0**	**124**	**143**	**115**	14	14	18	249	249	268	4155	4097	4183
126				**0,064516**	**0,096774**	**0,048387**	**1**	**1**	**1**	**2079**	**2079**	**1991**	265	265	253	4078	4088	4142
127							**0,992**	**1,144**	**0,92**	**15**	**19**	**17**	**32552**	**32552**	**32524**	4134	4024	4073
										16,5	**16,5**	**15,80159**	**255**	**255**	**244**	**524489**	**523849**	**523356**
													256,315	**256,315**	**256,0945**	**4099**	**4147**	**4072**
																4097,57	**4092,57**	**4088,719**

HEXADECIMAL 7

Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	1	1	1	17	18	16	268	281	275	4143	4065	4170
1	0	0	0	0	0	0	2	0	1	15	11	16	234	234	255	4096	4171	4114
2	0	0	0	0	0	0	2	0	3	14	14	24	240	272	271	4092	4155	4134
3	0	0	0	0	1	0	2	4	1	18	20	17	246	257	279	4053	4093	4117
4	0	0	0	0	0	0	1	1	3	9	17	14	227	261	274	4149	4130	4084
5	0	0	0	0	0	0	0	1	1	7	13	21	246	261	261	4021	4190	4189
6	0	0	0	0	0	0	0	1	0	17	20	15	265	251	248	4141	4138	4187
7	0	0	0	0	0	0	1	0	0	18	10	17	264	248	249	4122	4009	4195
8	0	0	0	0	0	0	0	2	1	12	15	15	259	261	259	4140	4150	4141
9	0	0	0	0	0	0	0	1	2	18	16	18	247	231	252	4207	4098	4196
10	0	0	0	0	0	0	1	1	0	17	14	17	264	247	260	4066	3995	4041
11	0	0	0	0	0	0	0	0	0	13	16	14	268	250	285	4223	3988	4166
12	0	0	0	0	0	0	1	1	1	13	17	17	250	253	267	3958	4122	4115
13	0	0	0	0	0	1	2	2	1	18	13	12	244	254	265	4148	4103	4129
14	0	0	0	0	0	0	3	0	3	20	13	24	238	277	247	4015	4126	4084
15	0	0	0	0	0	1	0	0	1	9	22	15	234	256	270	4071	4121	4094
16	0	0	0	0	0	0	2	1	1	17	20	12	250	277	246	4056	4111	4102
17	0	0	0	0	0	0	1	3	0	27	18	14	255	240	242	4117	4057	4041
18	0	0	0	0	0	0	0	0	1	18	14	12	274	259	253	4094	4158	4094
19	0	0	0	1	0	0	1	1	0	15	16	13	252	248	247	4143	4125	4088
20	0	0	0	0	0	0	2	1	0	14	16	17	267	251	263	4138	4000	4160
21	0	0	0	0	0	0	0	2	0	9	18	18	229	256	247	4195	4090	4177
22	0	0	0	0	0	0	0	1	0	16	14	14	234	269	248	4136	4096	4127
23	0	0	0	0	0	0	0	1	1	14	15	16	247	282	272	4027	4101	4078
24	0	0	0	0	0	0	2	0	1	19	17	16	261	233	275	4174	4046	4046
25	0	0	0	0	0	0	1	0	0	21	9	11	278	246	227	4170	4024	4006
26	0	0	0	0	0	0	0	0	1	12	9	13	249	239	289	4134	4069	4125
27	0	0	0	0	0	0	2	0	1	16	10	26	247	251	275	4121	4028	4203
28	0	0	0	0	0	0	1	0	1	17	11	20	227	237	279	4063	4052	4045
29	0	0	0	0	0	0	3	0	1	24	16	17	268	247	256	4150	4104	4114
30	0	0	0	0	0	0	0	1	0	15	11	11	244	299	238	4027	4116	4028
31	0	0	0	0	0	0	0	0	1	13	11	10	253	240	249	4166	3991	4161
32	0	0	0	0	0	0	0	1	2	22	16	18	221	238	252	4062	4092	4057
33	0	0	0	0	0	0	1	1	0	17	22	15	262	236	258	3973	4074	4135
34	0	0	0	0	0	0	4	2	1	11	17	13	252	276	268	4151	4116	4079
35	0	0	0	0	0	0	1	0	1	22	15	18	264	246	272	4105	4136	4114
36	0	0	0	0	1	0	1	1	2	21	12	15	253	240	267	4037	4129	4031
37	0	0	0	0	0	1	1	2	1	14	22	19	259	238	239	4132	4148	4070
38	0	0	0	0	0	0	1	0	3	22	14	17	279	244	281	4068	4083	4050
39	0	0	0	0	0	0	3	2	2	17	14	15	253	271	298	3996	4082	4169
40	0	0	0	0	0	0	2	0	1	19	13	19	236	275	262	4196	4134	4148
41	0	0	0	0	0	0	0	0	1	22	24	13	264	234	264	4095	4019	4090
42	0	0	0	0	0	0	0	1	1	10	7	15	262	272	271	4024	4021	4181
43	0	0	0	0	0	0	0	0	2	17	19	14	239	261	239	4144	4117	4038
44	0	0	0	0	0	0	1	1	2	10	20	17	237	255	240	3929	4036	4046
45	0	0	0	0	0	0	2	1	0	15	17	22	246	280	271	4115	4122	4070
46	0	0	0	0	0	0	1	1	1	21	23	16	255	252	273	4073	4133	4100
47	0	0	0	0	0	0	0	0	1	14	20	19	253	269	264	4124	4031	4131
48	0	0	0	1	0	0	2	0	0	11	9	12	236	260	246	4120	4080	4132
49	0	0	0	0	0	0	2	1	1	11	14	10	238	258	254	4071	4109	4121
50	0	0	0	0	0	0	0	1	0	18	15	17	241	264	265	4167	4160	4165
51	0	0	0	1	0	0	2	0	1	14	8	14	240	247	236	4182	4247	4131
52	0	0	0	0	0	1	1	2	3	9	12	19	236	248	262	3978	4028	4075
53	0	0	0	0	0	0	3	2	1	18	10	23	278	246	280	4084	4031	4158
54	0	0	0	1	0	0	3	0	1	27	15	15	288	255	242	4157	4045	4041
55	0	0	0	0	0	0	1	0	1	20	13	15	260	238	228	4200	4163	4098
56	0	0	0	0	0	0	1	3	0	9	15	21	279	231	234	4138	4092	4080
57	0	0	0	0	0	0	0	1	0	10	21	14	246	268	258	4127	4071	4074
58	0	0	0	0	0	0	3	0	2	15	13	18	278	267	242	4017	4125	4087
59	0	0	0	0	0	1	1	1	2	18	22	18	242	271	247	4144	4159	4092
60	0	0	0	0	0	0	0	1	2	23	13	11	258	241	282	4120	4067	4152
61	0	0	0	1	0	0	2	0	1	15	15	13	252	249	244	3984	4127	4137
62	0	0	0	0	0	0	1	0	1	16	7	15	263	246	264	4053	4074	4105
63	0	0	0	0	0	0	1	0	0	13	15	15	267	268	248	4260	4156	4184
64	0	0	0	0	0	0	0	0	1	18	18	20	287	265	278	4125	4110	4067
65	0	0	0	0	0	1	0	0	2	21	23	11	288	246	278	4226	4076	4103
66	0	0	0	0	0	0	1	1	2	15	19	20	282	265	274	4170	4048	4145
67	0	0	0	0	0	0	1	0	0	13	22	12	222	247	296	4173	4022	4120
68	0	0	0	0	0	0	0	0	0	13	11	20	255	242	256	3980	4052	4177
69	0	0	0	0	0	0	0	0	1	13	18	15	256	246	263	4029	3985	4082
70	0	0	0	0	0	0	2	1	0	17	15	14	249	246	249	4045	4076	4094
71	0	0	0	0	0	0	2	0	0	14	14	14	261	244	240	4068	4003	4087
72	0	0	0	0	0	0	0	1	1	17	13	22	272	240	242	4135	4021	4109
73	0	0	0	0	0	0	3	0	1	19	18	21	256	261	267	4012	4071	4222
74	0	0	0	0	0	0	1	0	0	18	17	11	260	255	257	4058	4013	4133
75	0	0	0	0	0	0	0	0	0	18	13	15	292	257	243	4127	4066	4018
76	0	0	0	0	0	0	1	1	0	13	16	15	210	257	247	4089	4035	4117
77	0	0	0	0	1	0	1	2	1	14	16	11	268	249	276	3992	4168	4060
78	0	0	0	0	0	0	2	2	0	20	12	13	260	248	233	4213	4068	3997
79	0	0	0	0	0	0	0	2	1	21	13	20	263	257	261	4168	4122	4037
80	0	0	0	0	0	1	0	1	2	14	20	18	252	256	256	4134	4196	4079
81	0	0	0	0	0	0	0	1	1	13	15	14	258	261	234	4045	4181	4090
82	0	0	0	1	0	0	1	1	0	16	21	11	256	268	234	4091	4123	4055
83	0	0	0	0	0	0	1	1	0	15	19	14	259	249	238	4060	4187	4098
84	0	0	0	0	0	0	0	1	0	15	17	18	250	255	267	4020	4132	4153
85	0	0	0	0	0	0	1	4	3	10	17	18	273	244	294	4172	4054	4178
86	0	0	0	0	1	0	1	2	0	20	14	11	245	238	252	4173	4130	4136
87	0	0	0	0	0	0	2	2	0	16	16	11	266	282	246	4155	4037	4144
88	0	0	0	0	0	0	0	0	3	13	12	13	277	254	240	4146	4109	4121
89	0	0	0	0	0	0	0	0	0	14	12	20	239	243	256	4115	4104	4000
90	0	0	0	0	0	0	0	0	1	14	17	13	256	273	253	4066	4164	4099
91	0	0	0	0	0	0	1	2	1	12	16	10	269	271	230	4111	4206	4203
92	0	0	0	0	0	0	3	1	0	19	17	13	267	271	259	4134	4179	4096
93	0	0	0	0	0	0	1	2	0	17	22	10	263	253	262	4183	4134	4133
94	0	0	0	0	0	0	0	0	3	13	20	13	297	267	259	4080	4084	4116
95	0	0	0	0	0	0	2	1	1	17	20	11	276	267	270	4202	4039	4221
96	0	0	0	0	0	0	0	1	2	14	23	16	261	243	233	4247	4146	3986
97	0	0	0	0	0	0	0	2	2	15	18	11	245	268	225	4094	4076	4121
98	0	0	0	0	0	0	0	1	1	17	20	16	252	248	254	4186	4061	4166
99	0	0	0	0	0	0	1	3	1	13	17	17	256	257	258	4051	4028	4107
100	0	0	0	0	0	0	0	1	1	10	19	21	235	266	261	4031	4098	4147
101	0	0	0	0	0	0	0	0	1	11	12	17	240	251	275	4027	4102	4167
102	0	0	0	0	0	0	0	0	1	9	9	23	254	237	252	3951	4078	4196
103	0	0	0	0	1	0	2	3	0	19	20	10	256	247	222	4169	3998	4101
104	0	0	0	0	0	0	1	1	2	14	17	16	257	279	239	4101	3945	4076
105	0	0	0	0	0	0	0	1	0	17	22	17	260	260	252	4069	4175	4063
106	0	0	0	0	1	0	1	2	1	17	23	10	273	264	254	4113	4070	4097
107	0	0	0	0	0	0	0	1	0	13	15	8	254	292	274	4059	4179	4064
108	0	0	0	0	0	0	1	1	1	10	14	14	231	260	250	4005	4082	4160
109	0	0	0	0	0	0	2	1	0	9	15	16	249	247	260	4137	4035	4040
110	0	0	0	0	0	0	0	0	0	18	22	11	245	249	255	4116	4005	4204
111	0	0	0	0	0	0	1	1	1	18	18	16	267	266	232	4083	4114	4061
112	0	0	0	0	0	1	0	2	1	17	19	12	250	244	245	4064	4004	4152
113	0	0	0	0	0	0	1	0	2	13	13	17	263	260	257	4098	3960	4050
114	0	0	0	0	0	0	2	0	0	21	16	15	232	278	247	4106	4073	4112
115	0	0	0	0	0	0	0	1	1	11	16	11	251	260	255	4133	4173	4008
116	0	0	0	0	0	0	0	1	2	12	16	15	271	238	266	4096	4177	4081
117	0	0	0	0	0	0	0	1	0	10	18	12	248	242	252	4020	4038	4119
118	0	0	0	0	0	0	1	0	3	18	16	18	240	296	225	4104	4142	4078
119	0	0	0	0	0	0	1	0	0	12	24	16	254	269	262	4119	4040	4035
120	0	0	0	0	0	0	0	1	0	6	16	20	233	266	271	4159	4087	4184
121	0	0	0	0	0	0	1	0	0	8	16	14	239	271	240	4068	4166	4101
122	0	0	0	0	0	0	3	0	1	17	4	20	238	267	249	4078	4127	4196
123	0	0	0	0	0	0	0	0	1	15	14	16	243	252	242	4130	4075	4166
124	0	0	0	6	6	8	0	1	1	16	17	15	241	301	249	4131	4213	4025
125	0	0	0	0	0	0	117	106	116	17	14	12	260	246	242	4103	4179	4132
126				0,048387	0,048387	0,064516	0	0	1	1942	2002	1951	233	245	257	4074	4032	4253
127							0,936	0,848	0,928	17	16	15	32281	32558	32539	4147	4073	4051
										15,4127	15,88889	15,48413	256	246	252	525148	523775	526081
													254,1811	256,3622	256,2126	4027	4122	4114
																4102,719	4091,992	4110,008

HEXADECIMAL 8																		
Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	1	2	2	4	14	17	22	248	227	264	3947	3992	4124
1	0	0	0	2	0	0	2	0	2	21	16	18	255	259	264	4234	4045	4102
2	0	0	0	0	0	0	4	1	0	19	16	23	272	277	265	3999	4155	4161
3	0	0	0	0	0	0	0	1	0	20	18	14	267	251	248	4232	4102	4054
4	0	0	0	0	0	0	1	0	2	16	12	15	284	251	266	4270	4188	4090
5	0	0	0	0	0	1	1	1	1	15	15	12	259	253	238	4197	4113	4070
6	0	0	0	0	0	0	0	0	4	20	11	18	263	269	251	4056	4162	4193
7	0	0	0	0	0	0	1	1	3	7	16	20	293	247	260	4137	4139	4031
8	0	0	0	0	0	0	0	0	1	18	11	14	248	251	255	4218	4063	4055
9	0	0	0	0	0	0	3	3	1	14	14	19	262	254	248	4101	4104	4101
10	0	0	0	0	0	0	3	1	0	17	22	14	275	236	279	4234	4066	4172
11	0	0	0	0	0	1	2	0	2	24	11	13	233	298	275	4044	4121	4188
12	0	0	0	0	1	0	1	1	3	12	22	18	239	236	280	4002	4035	4169
13	0	0	0	0	0	0	4	1	0	25	13	17	271	284	257	4079	3999	4108
14	0	0	0	0	0	0	1	4	0	24	14	14	278	281	227	4184	4009	4124
15	0	0	0	0	0	0	0	1	3	21	19	14	281	273	256	4094	4009	4127
16	0	0	0	0	0	0	0	1	2	10	16	22	269	258	280	4255	4109	4161
17	0	0	0	0	0	0	4	1	0	16	13	15	224	260	261	4205	4108	4125
18	0	0	0	0	0	0	0	2	1	19	18	10	264	248	241	4084	4164	4111
19	0	0	0	1	0	0	2	1	2	14	17	17	258	269	255	4077	4202	4128
20	0	0	0	0	0	0	1	1	1	13	14	21	258	233	250	4064	4146	4226
21	0	0	0	0	0	0	1	0	1	7	16	17	234	251	247	4076	4079	3961
22	0	0	0	0	0	0	1	0	1	17	13	12	261	258	248	4141	4047	4183
23	0	0	0	0	0	0	2	1	0	17	20	17	265	256	259	4164	4089	4065
24	0	0	0	0	0	0	0	0	1	18	12	23	259	279	239	4088	4188	4023
25	0	0	0	0	0	0	0	3	0	16	10	16	284	237	256	4126	4203	4006
26	0	0	0	1	0	0	1	3	1	9	21	14	256	256	270	4069	4112	4007
27	0	0	0	0	0	0	1	0	2	8	16	15	216	241	227	4120	4006	4081
28	0	0	0	0	0	0	0	1	1	12	19	20	239	250	254	4150	4054	4004
29	0	0	0	0	0	0	1	0	1	15	17	13	271	250	252	4066	4053	4060
30	0	0	0	0	0	0	1	1	0	9	19	17	254	265	289	4093	4094	4112
31	0	0	0	0	0	0	1	1	1	20	16	14	285	246	268	4082	4131	4065
32	0	0	0	0	0	1	1	2	1	16	16	14	285	267	256	4142	4154	4165
33	0	0	0	0	0	1	3	0	3	22	16	20	263	225	252	4073	4111	4051
34	0	0	0	1	0	0	3	2	2	22	16	19	265	249	270	4133	4046	4043
35	0	0	0	0	0	0	1	1	1	16	16	17	234	263	260	4118	4170	4072
36	0	0	0	0	0	0	2	3	1	19	15	15	261	260	237	4169	4067	4119
37	0	0	0	0	0	0	0	0	1	17	11	14	263	229	269	4124	3993	4003
38	0	0	0	0	0	0	1	2	1	13	17	13	250	227	244	4064	4045	4084
39	0	0	0	0	0	0	1	3	0	19	17	16	288	251	253	4123	4125	4127
40	0	0	0	0	0	0	1	1	0	9	17	15	250	295	275	4210	4146	4102
41	0	0	0	0	1	1	0	2	3	9	13	12	265	260	252	4118	4133	4214
42	0	0	0	0	0	0	0	1	2	11	11	15	244	258	258	4117	4035	4086
43	0	0	0	0	0	0	0	1	0	8	12	19	255	240	246	4169	4220	4035
44	0	0	0	1	0	0	1	0	1	18	5	16	269	254	249	3993	4156	4035
45	0	0	0	0	0	0	1	0	0	22	10	22	259	265	314	4249	4096	4195
46	0	0	0	0	0	0	0	0	0	16	18	22	269	226	297	4077	4030	4095
47	0	0	0	0	0	0	1	4	0	19	22	11	266	300	262	4148	4045	4161
48	0	0	0	0	0	0	2	1	1	18	19	17	291	273	250	4058	4123	4044
49	0	0	0	0	0	1	0	1	1	5	12	28	234	278	282	4122	4085	4115
50	0	0	0	0	0	0	2	0	2	18	17	16	283	258	287	4195	4124	4133
51	0	0	0	0	0	0	0	0	2	13	14	16	263	275	242	4198	4093	4212
52	0	0	0	0	0	0	0	2	1	8	17	16	236	261	279	4134	4201	4020
53	0	0	0	0	0	0	0	0	1	15	16	20	227	227	240	3911	4072	4097
54	0	0	0	0	0	0	0	0	2	6	20	19	217	251	272	4047	4011	4073
55	0	0	0	0	0	0	0	1	0	12	22	13	247	264	267	4072	4077	4048
56	0	0	0	0	0	0	2	1	0	13	18	10	252	261	265	3986	4190	4019
57	0	0	0	0	0	0	1	1	2	15	11	11	264	246	249	4001	4088	4029
58	0	0	0	1	0	1	1	1	2	11	19	16	251	247	237	4138	4022	3987
59	0	0	0	0	0	1	3	3	2	16	16	14	231	269	240	4093	4146	3993
60	0	0	0	0	0	0	0	1	1	19	16	13	254	247	226	4018	4161	4104
61	0	0	0	0	0	0	0	2	1	17	20	16	286	274	258	4094	3992	3961
62	0	0	0	0	0	0	1	0	1	18	15	20	267	252	279	4113	4200	4154
63	0	0	0	0	0	0	0	1	0	23	11	17	288	252	269	4114	4057	4046
64	0	0	0	0	0	0	1	0	0	15	19	15	246	256	231	4097	4133	4081
65	0	0	0	0	0	0	0	0	1	14	18	16	261	254	274	3961	4074	4149
66	0	0	0	0	0	0	2	2	2	16	21	16	237	264	256	4079	4125	4102
67	0	0	0	0	0	0	1	1	0	19	14	17	245	229	236	4086	4102	4187
68	0	0	0	0	0	0	0	1	2	19	14	8	255	252	256	4170	4125	4049
69	0	0	0	0	0	0	3	1	3	12	17	15	237	260	245	4110	4119	4129
70	0	0	0	0	1	0	2	2	0	22	14	16	254	260	251	4184	4149	4063
71	0	0	0	0	0	0	0	2	1	12	13	17	241	249	248	4212	4137	4044
72	0	0	0	0	0	0	3	0	1	16	16	11	237	256	249	4093	4172	4067
73	0	0	0	0	0	0	1	1	1	19	11	22	252	247	216	4117	4189	4098
74	0	0	0	1	0	0	1	3	3	15	14	22	269	266	241	4094	4149	4064
75	0	0	0	0	0	0	1	0	1	12	22	16	255	251	289	4042	4052	4091
76	0	0	0	1	0	0	2	0	0	15	17	13	237	250	245	4080	4014	4169
77	0	0	0	0	0	0	3	1	1	16	15	18	249	251	248	4146	3960	4006
78	0	0	0	0	0	0	0	1	1	16	10	26	248	229	254	4084	4092	4168
79	0	0	0	0	0	0	3	3	0	16	21	11	255	247	298	4132	3982	4148
80	0	0	0	0	0	0	3	1	1	23	16	11	236	241	261	4055	4158	4140
81	0	0	0	0	0	1	0	2	1	20	12	13	253	258	238	4072	4086	4008
82	0	0	0	0	1	0	0	1	3	8	22	18	263	286	253	4138	4115	4111
83	0	0	0	0	0	0	2	2	0	19	18	16	261	243	243	4175	4025	4117
84	0	0	0	0	0	0	1	1	0	16	18	6	248	258	228	4006	4109	4072
85	0	0	0	0	0	0	0	2	0	11	13	16	262	250	248	4102	4146	4022
86	0	0	0	0	0	0	0	2	0	21	17	8	257	228	235	4101	3987	4215
87	0	0	0	0	0	0	0	1	0	11	22	20	263	253	257	4136	4038	4070
88	0	0	0	1	0	0	1	1	1	19	15	15	249	291	260	4090	4186	4081
89	0	0	0	0	0	0	2	0	1	10	12	13	240	269	269	4126	3981	4155
90	0	0	0	0	0	0	1	1	0	17	9	12	263	285	260	4234	4120	4040
91	0	0	0	1	0	0	1	1	0	14	14	9	245	240	244	4005	4162	4105
92	0	0	0	0	0	1	3	1	2	14	15	15	270	247	221	4108	4042	4034
93	0	0	0	0	0	0	2	1	1	20	20	19	259	241	244	4078	4113	4089
94	0	0	0	0	0	0	0	1	2	18	10	20	255	252	243	4052	3998	4063
95	0	0	0	0	0	0	0	0	0	12	13	17	258	247	253	4227	4056	3964
96	0	0	0	0	0	0	0	3	0	10	15	7	223	262	245	4060	4198	4066
97	0	0	0	0	0	0	1	2	1	20	12	6	288	258	247	4076	4152	4200
98	0	0	0	1	0	0	1	0	1	15	15	12	279	266	254	4096	4107	4066
99	0	0	0	0	0	0	1	0	0	14	15	12	263	253	238	4126	4109	4103
100	0	0	0	0	0	0	2	0	3	13	13	17	263	265	228	4113	4078	4103
101	0	0	0	0	0	0	0	0	1	17	16	22	287	244	252	4203	4090	4118
102	0	0	0	0	0	0	0	0	0	20	11	18	259	266	286	4075	4129	4088
103	0	0	0	0	0	0	3	1	0	20	16	14	249	224	279	3922	4056	4141
104	0	0	0	0	0	0	0	1	0	14	20	14	252	251	250	4071	4066	4178
105	0	0	0	0	0	0	3	1	0	15	13	19	234	297	259	3969	4144	4113
106	0	0	0	0	0	1	1	1	1	20	12	22	274	251	275	3979	4106	4136
107	0	0	0	0	0	0	1	1	2	23	19	16	282	268	286	4074	4057	4166
108	0	0	0	0	0	0	0	0	1	10	11	18	253	266	266	4124	4101	4143
109	0	0	0	0	0	0	2	1	1	21	13	21	243	283	255	4149	4059	4210
110	0	0	0	0	0	0	1	3	0	14	25	20	282	282	259	4062	4245	4073
111	0	0	0	0	0	0	0	3	0	12	19	13	248	274	247	4166	4141	3986
112	0	0	0	0	0	0	1	1	0	17	19	11	296	247	242	4171	4235	4123
113	0	0	0	0	0	0	1	0	0	19	16	18	257	253	252	4126	4097	4064
114	0	0	0	0	0	0	1	2	1	17	18	16	250	244	251	4011	4090	3997
115	0	0	0	0	0	0	0	2	0	19	13	14	256	249	250	4025	4104	4070
116	0	0	0	0	0	0	0	2	2	18	20	18	255	278	249	4196	4051	4081
117	0	0	0	0	0	0	2	4	3	23	18	19	245	237	261	4045	4125	4114
118	0	0	0	0	0	1	1	0	2	17	14	16	280	251	283	4085	4071	4103
119	0	0	0	0	0	0	1	2	1	18	10	14	265	250	258	4107	4136	4161
120	0	0	0	1	0	0	1	1	0	13	11	17	269	257	258	4181	4125	4083
121	0	0	0	1	1	0	2	1	1	10	15	13	234	257	232	4129	4124	4030
122	0	0	0	0	0	0	2	1	3	19	18	13	232	261	219	3906	4113	4028
123	**0**	**0**	**0**	0	0	0	1	0	1	21	10	26	268	232	258	4021	4065	4080
124	**0**	**0**	**0**	**14**	**5**	**13**	0	1	2	14	15	12	282	231	238	4142	4080	4013
125	**0**	**0**	**0**	**0**	**0**	**0**	**137**	**140**	**132**	18	20	27	248	241	245	4108	4057	4078
126				**0,112903**	**0,040323**	**0,104839**	**1**	**1**	**1**	**1998**	**1963**	**2022**	254	261	264	4114	4140	4032
127							**1,096**	**1,12**	**1,056**	**19**	**16**	**16**	**32732**	**32477**	**32445**	4012	4031	4082
										15,85714	**15,57937**	**16,04762**	**263**	**251**	**248**	**525245**	**524679**	**523739**
													257,7323	**255,7244**	**255,4724**	**4126**	**4125**	**4161**
																4103,477	**4099,055**	**4091,711**

HEXADECIMAL 9																		
Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	0	1	0	14	17	13	241	258	219	4072	4103	4065
1	0	0	0	0	0	0	2	1	0	10	20	16	241	252	258	4051	4121	4109
2	0	0	0	0	0	0	1	1	2	20	13	17	243	263	254	4105	4067	3992
3	0	0	0	0	0	0	0	0	1	12	13	23	266	279	257	4093	4171	4143
4	0	0	0	0	0	0	1	1	0	13	17	20	225	261	266	3973	3989	4082
5	0	0	0	0	0	0	1	1	1	13	13	14	264	240	269	4072	4044	4135
6	0	0	0	0	0	0	0	0	0	8	18	15	241	278	295	4027	4117	4211
7	0	0	0	0	0	0	0	0	3	13	10	17	228	233	259	4045	4094	4069
8	0	0	0	0	0	0	0	3	2	10	15	19	238	262	294	4064	4086	4063
9	0	0	0	0	0	0	1	1	0	16	18	14	273	252	221	4110	4092	4055
10	0	0	0	0	0	0	2	1	1	14	12	19	226	260	248	4077	4232	3981
11	0	0	0	0	0	0	0	1	0	12	13	16	257	256	243	4019	4112	3966
12	0	0	0	0	0	0	1	0	0	15	18	14	259	256	235	4221	4161	4052
13	0	0	1	0	0	1	1	0	1	14	13	16	264	241	263	4149	4090	4088
14	0	0	0	0	0	1	0	0	1	18	12	13	258	233	251	4051	4036	4015
15	0	0	0	0	0	0	2	0	2	14	14	15	261	262	264	4159	4099	4057
16	0	0	0	0	0	0	0	3	1	19	19	14	234	250	241	4089	3978	4108
17	0	0	0	0	0	1	4	1	3	17	15	21	248	249	262	4074	4170	4073
18	0	0	0	0	0	0	1	1	3	20	11	20	263	272	281	4131	3993	4086
19	0	0	0	0	0	0	3	0	0	10	15	14	241	265	272	3943	4209	4136
20	0	0	0	0	0	0	0	0	2	22	14	20	231	270	247	4133	4142	3966
21	0	0	0	0	0	0	1	0	1	15	13	21	259	270	261	4092	4095	4070
22	0	0	0	0	0	0	0	1	0	18	10	17	279	262	260	4162	4081	4214
23	0	0	0	1	0	0	1	0	1	19	13	17	279	235	242	4114	4119	4106
24	0	0	0	0	0	0	4	0	0	18	12	12	264	248	254	4248	3949	4086
25	0	0	0	0	0	0	0	0	1	14	20	13	224	266	249	4143	4163	4181
26	0	0	0	0	0	0	0	0	0	16	10	15	274	254	231	4016	4027	4075
27	0	0	0	0	0	1	4	2	1	16	19	11	245	247	232	4064	4183	4079
28	0	0	0	0	0	0	2	3	3	19	17	19	271	251	233	4254	4019	4090
29	0	0	0	0	0	0	1	1	0	21	26	15	260	267	270	4042	4164	4118
30	0	0	0	0	0	0	0	0	0	15	21	18	266	269	263	4134	4139	4028
31	0	0	0	0	0	0	0	0	0	13	16	11	258	265	266	4067	4122	4195
32	0	0	0	1	0	0	1	3	0	12	18	11	237	278	242	4064	4089	4014
33	0	0	0	0	0	0	2	2	1	17	15	16	261	270	256	4048	4083	3944
34	0	0	0	0	0	0	2	1	3	18	15	17	269	257	271	4010	4151	4144
35	0	0	0	0	0	0	0	1	1	16	17	20	241	268	253	4106	4018	4133
36	0	0	0	0	0	0	2	1	1	10	17	13	255	288	260	4055	4118	3995
37	0	0	0	0	1	0	1	2	1	16	13	12	243	242	256	4065	4198	4063
38	0	0	0	0	0	0	0	1	0	18	13	14	267	241	243	4144	4127	4142
39	0	0	0	0	0	0	2	0	1	20	22	23	283	264	277	4155	4075	4056
40	0	0	0	0	0	0	1	1	0	17	14	13	258	259	263	4013	4180	4108
41	0	0	0	0	0	0	0	2	0	15	24	9	256	266	247	4095	4183	4260
42	0	0	0	0	0	0	1	1	0	9	12	16	255	241	258	4057	4068	4158
43	0	0	0	0	0	0	0	1	0	19	8	10	251	240	273	4129	4008	4131
44	0	0	0	1	0	0	1	1	0	13	18	18	255	263	263	4081	4059	4190
45	0	0	0	0	0	0	1	3	1	17	24	19	255	284	233	4077	4188	3989
46	0	0	0	0	0	0	1	1	0	20	19	18	251	248	285	4056	4115	4173
47	0	0	0	0	1	0	0	4	0	17	21	8	241	263	234	4155	4095	4042
48	0	0	0	0	0	0	1	1	0	15	27	18	261	260	267	4084	4195	4141
49	0	0	0	1	0	0	1	0	0	16	10	11	249	277	269	4081	3966	4141
50	0	0	0	1	0	0	3	2	1	17	21	11	252	270	246	4006	4174	4142
51	0	0	0	0	1	0	1	1	0	14	18	21	241	270	277	4108	4123	4140
52	0	0	0	1	0	0	3	1	0	14	15	14	286	283	265	4090	4047	4204
53	0	0	0	0	0	0	2	1	2	18	13	19	261	261	255	4149	4133	4176
54	0	0	0	0	0	0	2	0	0	17	15	22	240	253	263	4042	4031	4128
55	0	0	0	0	0	0	1	1	1	14	13	17	260	257	251	4087	4123	4031
56	0	0	0	0	0	0	0	1	1	18	12	11	253	271	240	4116	4151	4055
57	0	0	0	0	0	0	1	0	0	15	22	15	266	258	255	4073	4120	4104
58	0	0	0	0	0	0	3	2	0	17	19	9	255	252	255	4095	4074	4171
59	0	0	0	0	0	0	1	2	0	26	20	11	233	264	220	4014	4049	4002
60	0	0	0	0	0	0	0	1	0	19	22	12	256	262	258	4091	4129	4157
61	0	0	0	0	0	0	2	0	1	8	14	13	257	250	249	4098	4034	4016
62	0	0	0	0	0	0	0	0	1	17	9	16	243	228	247	4019	4000	4120
63	0	0	0	0	0	0	1	0	0	7	11	16	256	226	261	3992	4044	4128
64	0	0	0	0	0	0	2	0	0	20	11	25	233	265	272	4090	4005	4205
65	0	1	0	0	1	0	1	2	0	18	18	15	244	266	289	4088	4126	4049
66	0	1	0	0	1	0	0	1	2	14	19	15	283	257	248	4148	4083	4067
67	0	0	0	0	1	0	1	1	2	11	13	17	230	250	264	4247	4198	4055
68	0	0	0	0	0	0	1	1	1	13	19	21	233	276	256	4006	4102	4151
69	0	0	0	0	1	0	0	1	1	17	11	19	266	266	267	3984	4038	4168
70	0	0	0	0	0	0	1	2	2	11	16	13	245	249	237	4009	4083	4149
71	0	0	0	0	0	0	1	0	2	20	20	19	257	259	251	4137	4020	4136
72	0	0	0	0	1	0	0	4	0	11	16	17	246	265	257	4131	4123	4084
73	0	0	0	0	0	0	2	1	1	16	23	20	238	253	239	4089	4054	4037
74	0	0	0	0	0	0	1	1	1	16	17	30	236	253	267	4061	4029	4194
75	0	0	0	0	0	0	2	2	0	17	17	11	257	241	277	4145	4088	4128
76	0	0	0	0	0	0	1	0	4	13	16	16	265	291	269	4070	4131	4114
77	0	0	0	0	0	0	1	2	0	15	17	16	227	258	243	4078	4098	4155
78	0	0	0	0	0	0	0	0	0	11	17	13	279	272	255	4094	4154	4079
79	0	0	0	0	0	0	1	0	1	15	12	12	259	250	238	4019	4178	4022
80	0	0	0	0	0	0	2	2	0	17	15	24	260	249	250	4072	4074	4045
81	0	0	0	0	0	0	1	0	3	16	16	16	270	264	288	4153	4206	4173
82	0	0	0	0	0	0	2	0	0	17	22	18	257	278	278	4076	4104	4149
83	0	0	0	0	0	0	0	1	1	12	14	21	243	236	255	4022	4137	3969
84	0	0	0	0	0	0	3	0	1	15	14	16	290	243	250	4167	4064	4088
85	0	0	0	0	0	0	1	1	2	24	15	13	229	267	264	4132	4001	4144
86	0	0	0	0	0	0	1	0	0	10	12	13	232	260	232	3987	4155	4086
87	0	0	0	0	0	1	0	0	1	11	12	26	250	250	279	4103	4121	4095
88	0	0	0	0	0	0	1	0	3	13	18	12	257	249	257	4075	4173	4117
89	0	0	0	0	0	0	1	0	1	19	12	18	274	248	261	4116	4049	4208
90	0	0	0	0	0	0	2	1	0	17	22	13	247	257	256	4175	4084	4195
91	0	0	0	0	0	1	1	0	1	19	10	18	226	244	257	4048	4097	4086
92	0	0	0	0	0	0	0	1	5	9	12	20	271	235	282	4214	4121	4118
93	0	0	0	0	0	0	0	2	1	17	15	27	264	262	266	4167	3974	4230
94	0	0	0	0	0	1	1	1	2	23	12	19	249	228	284	4109	4093	4107
95	0	0	0	0	0	0	0	1	1	18	15	22	288	253	269	4122	4063	4146
96	0	0	0	0	0	0	1	1	0	19	18	12	253	245	214	4029	4099	4189
97	0	0	0	0	0	0	1	1	2	12	10	13	268	256	244	4145	4104	3967
98	0	0	0	0	0	0	4	2	0	22	15	17	278	255	249	4196	4091	4170
99	0	0	0	0	0	0	3	1	1	20	13	23	283	248	255	4300	4067	4009
100	0	0	0	0	0	0	2	1	2	16	20	22	269	272	266	4146	3990	4122
101	0	0	0	0	0	0	0	2	3	17	20	20	261	285	271	4181	4156	4132
102	0	0	0	0	1	1	1	1	1	13	21	15	247	244	261	4132	4094	4120
103	0	0	0	1	0	0	2	1	1	24	16	14	254	279	252	4088	4173	4180
104	0	0	0	0	0	0	2	0	1	19	11	23	248	264	249	4003	4208	4120
105	0	0	0	0	0	0	0	1	0	18	9	14	252	238	241	4105	4036	3999
106	0	0	0	0	0	0	0	3	2	19	18	19	262	242	235	4197	4130	4052
107	0	0	0	0	0	0	0	3	1	11	12	10	254	251	261	4045	4100	3930
108	0	0	0	0	0	0	1	1	0	16	15	14	267	241	223	4145	4063	4066
109	0	0	0	0	0	0	0	2	0	16	17	9	267	260	250	4062	4077	3994
110	0	0	0	0	0	0	0	1	0	15	24	15	255	252	254	4108	4120	3951
111	0	0	0	0	0	0	1	1	1	16	11	16	266	273	240	4142	4026	4078
112	0	0	0	0	0	0	0	1	1	12	17	18	239	241	276	4085	4095	4134
113	0	0	0	0	0	0	1	0	0	16	20	16	244	231	274	3973	4139	4171
114	0	0	0	0	0	0	2	1	1	11	13	10	254	264	270	4114	4129	4085
115	0	0	0	0	0	0	2	0	1	20	15	21	277	250	235	4104	4077	4196
116	0	0	0	0	0	0	0	0	0	15	18	21	262	261	268	4000	4075	4087
117	0	0	0	0	0	1	1	1	3	13	22	18	255	275	257	4069	4174	4206
118	0	0	0	0	0	0	0	2	2	13	23	14	258	257	263	4100	4156	4107
119	0	0	0	0	0	0	0	1	0	17	21	9	263	250	257	4207	4110	4137
120	0	0	0	0	0	0	0	0	1	16	21	11	279	258	233	4069	4055	4028
121	0	0	0	0	0	0	0	0	1	12	15	15	230	242	266	4054	3993	4063
122	0	0	0	0	0	0	0	0	0	15	17	21	276	280	268	4150	4062	4218
123	0	2	1	0	0	0	0	0	1	15	11	13	270	246	251	4144	4070	4072
124	0	0	0	7	9	9	2	1	0	22	8	13	259	247	264	4099	4043	4089
125	0	0,01626	0,00813	0	0	0	127	118	111	16	17	14	242	265	237	4055	4085	4120
126				0,056452	0,072581	0,072581	1	1	0	1975	2004	2037	223	267	263	4163	4055	4124
127							1,016	0,944	0,888	17	15	13	32348	32673	32556	4070	4120	4100
										15,6746	15,90476	16,16667	241	250	263	524063	524313	524857
													254,7087	257,2677	256,3465	4072	4121	4086
																4094,242	4096,195	4100,445

HEXADECIMAL a

Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	0	1	1	10	13	18	249	256	284	4057	4060	4105
1	0	0	0	0	0	0	1	0	1	7	15	16	254	259	267	4043	4196	4174
2	0	0	0	0	0	0	0	1	3	18	18	18	262	269	271	4116	4098	4115
3	0	0	0	0	0	0	0	0	0	17	15	14	257	284	247	4030	4056	4123
4	0	0	0	0	0	0	3	1	0	16	15	14	244	233	239	4091	4028	3968
5	0	0	0	0	0	0	0	2	0	18	16	10	277	260	236	4047	3977	3947
6	0	0	0	0	0	0	1	1	2	10	18	18	251	259	259	4122	4029	4063
7	0	0	0	0	0	0	1	1	3	12	14	17	255	265	252	4041	4138	4068
8	0	0	0	0	0	0	0	0	0	14	20	11	253	245	216	4204	4126	4087
9	0	0	0	1	0	0	3	0	0	23	15	16	262	243	242	4056	3980	4110
10	0	0	0	0	0	0	2	3	3	12	15	17	235	235	259	4063	3977	4043
11	0	0	0	0	0	0	1	0	0	14	9	16	254	236	239	4111	4095	4114
12	0	0	0	0	0	0	0	1	0	21	19	11	244	258	266	4041	4039	4056
13	0	0	0	0	0	0	1	3	1	15	16	18	272	271	264	4141	4169	4145
14	0	0	0	0	0	0	2	2	0	14	19	14	257	239	246	4056	3977	4067
15	0	0	0	0	0	0	0	0	0	14	18	11	259	263	264	4153	4147	4120
16	0	0	0	0	0	0	0	2	1	17	9	13	243	251	257	4046	4077	4062
17	0	0	0	0	0	0	1	2	0	16	19	14	276	237	271	4096	4081	4138
18	0	0	0	1	0	1	1	2	2	11	13	17	270	241	253	4090	4117	4173
19	0	0	0	0	0	0	1	1	1	18	17	14	248	250	260	4037	4008	4094
20	0	0	0	0	0	0	1	2	0	14	15	10	274	270	287	4044	4113	4018
21	0	0	0	0	0	0	0	0	1	14	15	11	261	266	266	4165	4077	4102
22	0	0	0	0	0	0	1	0	2	19	22	16	278	261	262	4010	4068	4124
23	0	0	0	0	0	0	0	1	2	16	13	18	252	242	271	4079	4080	4166
24	0	0	0	0	1	0	1	1	1	14	15	14	227	248	258	4093	4153	4039
25	0	0	0	0	0	0	1	1	1	13	17	17	261	252	263	4098	4090	4164
26	0	0	0	0	0	0	1	0	1	15	14	14	265	267	253	4087	4056	4012
27	0	0	0	0	1	0	0	2	2	12	11	21	225	236	293	4032	4203	4162
28	0	0	0	0	0	0	1	3	0	10	19	23	232	251	281	4164	4203	4109
29	0	0	0	0	0	0	1	1	2	13	19	11	273	252	273	4014	4074	4112
30	0	0	0	0	0	0	2	1	0	22	18	11	270	270	243	4200	4082	4114
31	0	0	0	0	0	0	0	1	0	18	17	18	252	235	259	4067	4095	4034
32	0	0	0	0	0	0	1	2	1	15	18	20	246	252	261	4054	4130	4183
33	0	0	0	0	0	1	1	1	3	13	23	19	243	242	241	4119	4010	4094
34	0	0	0	0	0	0	0	0	2	14	21	19	234	285	252	4032	4185	4067
35	1	0	0	1	0	0	3	1	0	18	14	16	268	240	253	4096	4148	4087
36	0	0	0	1	0	0	1	0	0	17	24	18	248	278	283	4023	4139	4178
37	0	0	0	0	0	0	3	2	0	9	18	15	260	258	234	4000	4152	4219
38	0	0	0	0	0	0	1	1	1	19	12	16	255	247	230	4109	3931	4016
39	0	0	0	0	0	0	2	1	0	16	27	10	268	270	267	4092	4085	4062
40	0	0	0	0	0	1	1	0	1	14	16	19	241	253	246	4078	4111	4033
41	0	0	0	0	0	0	0	0	2	13	16	18	245	226	261	3991	3985	4077
42	0	0	0	0	1	0	0	2	0	17	13	17	258	226	247	3992	3981	4092
43	0	0	0	0	0	0	0	2	1	11	13	15	256	242	277	4151	4063	4087
44	0	0	0	0	0	0	1	1	0	15	21	6	268	255	233	4194	4058	4227
45	0	0	0	0	0	0	1	0	1	13	14	19	231	247	253	4127	4041	4142
46	0	0	0	0	0	1	0	1	1	18	15	15	238	263	271	4101	4099	4182
47	0	0	0	0	0	0	0	1	1	14	15	8	247	247	256	4092	4032	4203
48	0	0	0	0	0	0	0	1	3	23	10	15	241	243	225	4017	4021	4095
49	0	0	0	0	0	0	0	2	0	22	16	17	264	252	257	4147	4087	3945
50	0	0	0	0	0	0	2	3	0	16	18	13	265	238	249	4182	4043	4024
51	0	0	0	0	0	0	1	1	0	16	23	15	256	276	243	4067	4073	3974
52	0	0	0	0	0	0	1	1	3	18	8	22	236	243	254	4123	4085	4137
53	0	0	0	0	0	1	1	2	1	14	19	12	244	261	273	4156	4015	3977
54	0	0	0	0	0	0	0	0	1	10	13	17	234	250	267	3980	4205	4125
55	0	0	0	0	0	0	1	0	2	19	14	20	264	262	277	4242	4103	4120
56	0	0	0	0	1	0	0	2	3	14	12	18	240	247	254	4018	4059	4127
57	0	0	0	0	0	0	1	3	0	11	17	17	249	244	268	4087	4059	4122
58	0	0	0	0	0	0	2	1	4	20	9	22	238	245	285	4096	4069	4085
59	0	0	0	0	0	0	0	1	0	16	23	19	257	236	274	4104	4120	4088
60	0	0	0	0	0	0	0	0	0	11	22	14	234	267	251	4120	4108	4115
61	0	0	0	1	0	0	3	2	1	23	19	15	236	258	258	4012	4113	4085
62	0	0	0	0	0	0	2	2	2	26	13	16	282	235	261	4169	4042	3991
63	0	0	0	1	0	0	2	2	0	25	17	17	282	278	246	4071	4145	4127
64	0	0	0	0	0	0	2	0	0	17	14	12	258	253	227	4051	4185	4106
65	0	0	0	0	0	0	1	0	0	15	12	18	250	267	270	3956	4139	3926
66	0	0	0	0	0	0	1	1	0	12	12	14	265	262	277	4185	4155	4066
67	0	0	0	0	0	0	1	0	2	22	12	27	245	253	288	4071	4098	4190
68	0	0	0	0	0	0	2	2	3	16	9	17	259	220	254	4077	4076	4214
69	0	0	0	0	0	0	1	1	0	20	20	19	273	249	275	4122	4003	4074
70	0	0	0	0	0	0	3	1	3	21	18	13	271	274	252	4210	4064	4110
71	0	0	0	0	1	0	2	1	0	15	20	23	256	283	266	4010	4177	4047
72	0	0	0	0	0	0	0	1	0	18	12	12	255	268	269	4022	4065	4085
73	0	0	0	0	0	0	2	1	1	9	13	19	272	236	250	4222	4030	4095
74	0	0	0	0	0	0	0	2	2	25	17	21	271	264	248	4198	3986	3992
75	0	0	0	0	0	0	1	1	0	22	15	21	267	281	236	4080	4118	4161
76	0	0	0	0	0	0	1	3	0	13	13	13	254	230	263	4144	4023	4071
77	0	0	0	0	0	0	1	0	1	20	19	13	248	241	276	4217	4091	4119
78	0	0	0	0	0	0	2	1	0	18	21	16	266	253	247	4214	4162	4127
79	0	0	0	0	0	0	2	0	0	21	16	20	263	251	247	4131	4071	3945
80	0	0	0	0	0	0	0	1	0	13	16	20	261	218	279	4200	4122	4127
81	0	0	0	0	0	0	1	1	1	10	16	10	254	287	283	4137	4098	4280
82	0	0	0	0	0	0	1	0	0	16	13	17	255	244	236	4147	4048	3988
83	0	0	0	0	0	1	2	1	3	16	9	19	247	239	270	4014	4033	4216
84	0	0	0	0	0	0	1	1	1	13	18	12	253	218	252	4115	4049	4162
85	0	0	0	0	0	0	0	0	1	18	12	22	246	249	245	4101	4066	4010
86	0	0	0	0	0	0	0	1	1	10	12	14	249	230	245	4008	4017	4023
87	0	0	0	0	0	0	2	1	1	21	19	18	226	265	248	4166	4143	4072
88	0	0	0	0	0	0	1	0	3	18	14	16	231	273	265	4161	4053	4245
89	0	0	0	0	0	0	2	1	0	14	12	24	279	239	257	4155	4184	3988
90	0	0	0	0	1	0	2	2	1	21	23	12	230	289	250	4063	4262	4123
91	0	0	0	0	0	0	1	2	0	13	18	20	276	269	277	4119	4030	4136
92	0	0	0	0	0	0	1	1	1	17	17	13	240	234	245	4005	4116	4076
93	0	0	0	0	0	0	1	1	1	21	22	16	258	276	244	4141	4081	4036
94	0	0	0	0	0	0	0	0	0	12	15	18	262	256	256	4108	4088	4010
95	0	0	0	0	0	0	0	0	1	7	14	13	226	252	268	4088	4060	4055
96	0	0	0	0	0	0	3	2	0	18	18	13	253	259	243	3927	4122	3977
97	0	0	0	0	0	0	1	0	0	20	21	12	276	259	286	4060	4068	4117
98	0	0	0	0	0	0	1	0	1	11	15	14	263	277	247	4080	4117	4050
99	0	0	0	0	0	0	2	0	1	11	10	21	255	252	226	4115	4148	4075
100	0	0	0	0	0	0	0	1	0	16	20	15	250	258	265	4006	4060	4037
101	0	0	0	0	0	0	1	0	0	16	7	11	264	264	266	4072	4178	4081
102	0	0	0	0	0	0	2	3	0	13	19	17	248	249	247	4120	4074	4068
103	0	0	0	0	0	0	0	1	0	18	15	15	269	261	255	4105	4078	4046
104	0	0	0	0	0	0	1	0	0	12	15	14	257	253	229	4135	4132	4184
105	0	0	0	0	0	0	2	1	0	20	17	14	241	256	256	4139	4058	4136
106	0	0	0	0	0	0	0	0	0	14	11	12	256	268	235	4102	4111	4018
107	0	0	0	1	0	0	1	1	0	15	22	11	271	256	243	4043	4134	4097
108	0	0	0	0	0	1	1	0	3	15	14	18	263	271	249	4170	4063	4027
109	0	0	0	0	0	0	2	2	1	24	17	12	275	250	248	4149	4143	4122
110	0	0	0	0	0	0	1	1	1	19	16	15	257	283	235	4109	4101	4089
111	0	0	0	0	0	0	0	1	0	9	16	17	254	258	274	4102	4170	4088
112	0	0	0	0	0	0	1	2	0	21	16	11	247	259	233	4160	4056	4150
113	0	0	0	1	0	0	1	1	0	18	14	16	250	256	213	4180	4058	4038
114	0	0	0	0	0	0	1	1	1	17	12	11	244	249	258	3936	4108	4059
115	0	0	0	0	0	0	2	1	1	24	16	15	280	230	271	4168	4074	3981
116	0	0	0	0	0	0	1	0	0	15	16	13	244	268	255	4144	4007	4126
117	0	0	0	0	0	0	0	2	2	22	11	15	266	241	256	4042	4106	3999
118	0	0	0	0	0	0	1	1	1	17	20	20	257	249	256	4010	4045	4150
119	0	0	0	0	1	0	2	2	0	16	16	15	240	257	272	4207	4130	3983
120	0	0	0	0	0	0	0	2	0	14	25	9	249	258	266	4107	4084	4078
121	0	0	0	0	0	0	0	1	2	19	14	12	242	238	294	4029	3968	4184
122	0	0	0	0	0	0	1	2	2	13	15	16	260	265	237	4090	4172	4082
123	**1**	**0**	**0**	0	0	0	2	1	1	18	19	18	250	263	250	4125	4166	4105
124	**0**	**0**	**0**	**8**	**7**	**7**	2	1	1	16	10	16	271	242	278	4227	4071	4087
125	**0,00813**	**0**	**0**	**0**	**0**	**0**	**127**	**132**	**109**	18	11	16	262	259	231	4105	4022	4059
126				**0,064516**	**0,056452**	**0,056452**	**1**	**1**	**0**	**2020**	**2002**	**1976**	266	249	274	4175	4187	4053
127							**1,016**	**1,056**	**0,872**	**18**	**15**	**18**	**32366**	**32250**	**32621**	4145	4097	4229
										16,03175	**15,88889**	**15,68254**	**257**	**259**	**247**	**524458**	**523167**	**523464**
													254,8504	**253,937**	**256,8583**	**4096**	**4060**	**4087**
																4097,328	**4087,242**	**4089,563**

HEXADECIMAL b

Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	1	1	0	10	14	17	260	240	280	4120	4158	4125
1	0	0	1	0	1	2	2	1	2	15	17	19	225	247	266	4021	3984	4142
2	0	0	0	0	0	1	3	1	2	15	27	12	253	267	230	4021	4175	4078
3	0	0	0	0	0	0	6	3	2	27	29	25	288	275	254	4081	4124	4051
4	0	0	0	0	0	0	2	0	1	23	13	19	271	250	269	4125	4010	4098
5	0	0	0	0	0	0	0	1	0	17	9	14	272	259	273	4127	4106	4088
6	0	0	0	0	0	0	0	0	0	10	11	13	237	268	225	4043	4031	4027
7	0	0	0	0	0	0	1	0	0	15	13	11	249	236	263	4002	4023	4056
8	0	0	0	0	1	0	1	3	1	20	16	15	246	236	233	4051	4044	4217
9	0	0	0	0	0	0	0	1	0	13	17	15	279	263	247	4022	4103	4069
10	0	0	0	1	0	0	2	2	0	11	14	19	273	285	282	4156	4154	4113
11	0	0	0	0	0	0	1	1	1	20	19	12	243	266	221	4083	4089	4107
12	0	0	0	0	0	1	0	0	1	17	13	13	245	261	244	4038	4122	4012
13	0	0	0	0	0	0	1	0	3	20	10	18	266	250	256	4125	4045	4037
14	0	0	0	0	0	0	4	1	2	19	12	16	278	252	259	4024	4132	4131
15	0	0	0	0	0	0	1	0	1	16	13	17	283	244	265	4008	4029	4066
16	0	0	0	0	0	0	0	0	1	17	13	15	254	221	286	4158	4145	4228
17	0	0	0	0	0	0	2	2	1	13	17	19	229	266	264	4057	4064	4082
18	0	0	0	0	0	0	3	1	1	17	14	10	240	234	271	3894	4136	4040
19	0	0	0	0	0	0	0	1	0	23	16	13	272	274	243	4183	4108	4086
20	0	0	0	0	1	0	1	3	3	9	20	14	252	282	232	4118	4054	4055
21	0	0	0	0	0	0	0	2	1	17	15	22	249	239	263	4138	4046	4054
22	0	0	0	0	0	0	1	0	1	11	16	13	234	254	267	4070	4136	4126
23	0	0	0	0	0	0	1	2	2	17	18	18	273	255	255	4032	4037	4043
24	0	0	0	0	0	0	0	1	2	18	11	20	255	264	251	4113	4057	4122
25	0	0	0	0	0	0	1	1	2	12	17	22	257	248	267	4125	4155	4090
26	0	0	0	0	0	0	1	0	0	12	20	18	248	245	285	4164	4146	4068
27	0	0	0	0	0	0	0	1	3	19	16	17	261	270	259	4064	4177	4008
28	0	0	0	0	0	0	2	0	0	17	9	20	251	250	275	3965	4122	4198
29	0	0	0	0	0	0	0	0	1	10	20	12	266	251	250	4152	4153	4156
30	0	0	0	0	0	1	1	1	2	12	23	20	226	263	291	4136	4208	4086
31	0	0	0	0	0	0	0	1	3	15	21	23	241	265	266	4065	3991	4071
32	0	0	0	0	0	0	1	1	2	14	20	17	242	256	245	4072	4196	4042
33	0	0	0	0	0	0	0	1	1	14	24	17	250	267	275	4114	4126	4116
34	0	0	0	0	0	0	2	0	2	18	10	27	286	252	239	4120	4102	4068
35	0	0	0	0	0	0	2	0	1	11	7	18	275	239	258	4066	4121	4062
36	0	0	0	0	0	0	1	2	1	15	13	23	247	236	249	4070	4088	4079
37	0	0	0	1	0	0	3	3	0	18	15	10	253	218	267	4062	4012	4052
38	0	0	0	0	0	0	1	2	0	21	23	19	264	230	254	4114	4127	4090
39	0	0	0	0	0	0	1	0	1	15	14	27	250	266	276	4130	4070	4112
40	0	0	0	0	0	0	2	0	0	14	13	20	244	230	242	4120	4047	4038
41	0	0	0	0	0	0	0	0	0	14	10	12	258	266	270	4096	4073	4131
42	0	0	0	1	1	0	2	3	0	19	15	14	242	256	290	4092	4116	4131
43	0	0	0	0	0	0	2	1	2	19	17	12	263	231	236	4038	4104	4105
44	0	0	0	0	0	0	0	0	2	15	15	16	272	255	244	4204	4071	3995
45	0	0	0	0	0	0	2	2	0	20	22	11	258	270	260	4098	4191	3993
46	0	0	0	0	0	0	1	0	1	19	14	11	262	268	221	4138	4264	4198
47	0	0	0	0	0	0	0	1	1	16	11	13	227	242	236	3945	4226	4010
48	0	0	0	0	0	0	0	0	4	10	23	24	270	255	277	4126	4088	4034
49	0	0	0	0	0	0	2	5	1	11	17	18	251	277	277	4249	4102	4151
50	0	0	0	0	0	1	0	2	1	15	23	17	250	273	257	3961	4037	4146
51	0	0	0	0	0	0	2	0	1	16	14	18	255	266	265	4054	4102	4182
52	0	0	0	0	0	0	0	0	2	16	9	17	248	251	246	4182	4135	4067
53	0	0	0	0	0	0	1	0	0	15	9	15	238	228	272	4114	4124	4176
54	0	0	0	0	0	0	1	1	1	17	13	11	251	267	239	4051	4073	4196
55	0	0	0	0	0	0	2	0	0	13	10	13	270	247	262	4098	4131	4094
56	0	0	0	0	0	0	1	0	1	18	14	15	263	205	274	4176	3943	4275
57	0	0	0	1	0	0	4	1	1	18	19	16	245	267	292	4147	3993	4114
58	0	0	0	0	0	0	1	3	0	14	18	17	235	255	258	4123	4040	4203
59	0	0	0	0	0	0	0	0	1	8	17	14	247	283	258	4081	4181	4093
60	0	0	0	0	0	0	1	0	5	7	20	19	226	265	243	4041	4209	4166
61	0	0	0	1	0	0	1	0	2	11	15	20	280	254	244	4132	4092	4029
62	0	0	0	0	0	0	3	1	0	21	17	10	247	260	260	4099	4082	4016
63	0	0	0	0	0	1	1	0	3	18	17	18	303	264	227	4157	4157	4050
64	0	0	0	0	0	0	2	2	2	17	13	15	275	236	237	4112	4102	4104
65	0	0	0	0	0	1	0	1	1	19	20	15	284	252	283	4260	3989	4109
66	0	0	0	0	0	0	2	2	1	15	11	15	268	243	260	4179	4076	4080
67	0	0	0	0	0	0	1	0	0	16	17	21	252	267	273	4044	4137	4160
68	0	0	0	0	0	0	0	0	3	16	13	21	271	246	297	4076	4081	4133
69	0	0	0	0	0	0	1	2	0	22	17	22	277	258	281	4141	4242	4199
70	0	0	0	0	0	0	1	1	2	16	26	11	249	276	259	4107	4146	4118
71	0	0	0	0	0	0	0	0	1	11	18	21	227	270	258	4076	4195	4154
72	0	0	0	0	0	0	0	0	0	22	9	13	266	262	269	4130	4217	4044
73	0	0	0	0	0	0	3	2	0	14	18	13	253	247	245	4156	4063	4154
74	0	0	0	0	0	0	3	0	1	20	9	13	242	247	238	4030	4114	4071
75	0	0	0	0	0	0	0	1	1	15	9	15	263	267	240	4054	4050	4129
76	0	0	0	0	0	0	1	1	0	18	17	8	259	266	246	4079	4093	4009
77	0	0	0	0	0	0	1	0	1	21	18	19	257	247	304	4085	4146	4091
78	0	0	0	0	0	0	1	2	1	11	23	21	259	266	258	4092	4046	4065
79	0	0	0	0	0	1	3	0	1	13	21	17	237	269	245	3966	4093	4119
80	0	0	0	0	0	0	1	1	2	14	20	13	261	271	274	4054	4066	4219
81	0	0	0	0	0	0	3	1	1	12	18	18	291	283	232	4174	4160	4083
82	0	0	0	0	0	0	0	2	0	19	16	11	255	245	251	4146	4040	4080
83	1	0	0	1	0	0	2	1	0	16	17	15	250	237	228	4114	4104	4109
84	0	0	0	2	0	0	3	0	1	13	17	16	239	249	275	4098	4114	4131
85	0	0	0	0	0	0	2	1	2	18	17	16	258	253	266	4084	4145	4132
86	0	0	0	0	0	0	2	1	0	18	12	15	258	235	253	4105	4158	4123
87	0	0	0	0	0	0	1	0	1	21	18	17	268	260	268	4200	4113	4112
88	0	0	0	0	0	0	2	1	0	17	16	11	261	268	239	4197	4175	4024
89	0	0	0	0	0	0	1	1	2	12	9	15	247	261	219	4049	4192	4043
90	0	0	0	0	0	0	0	0	1	11	17	20	278	267	223	4077	4169	4067
91	0	0	0	0	0	0	1	1	1	15	19	18	266	261	258	4095	4036	4064
92	0	0	0	0	0	0	0	0	1	11	12	16	249	260	269	4167	3990	4153
93	0	0	0	0	0	1	1	0	2	14	10	16	252	269	271	4018	4070	4149
94	0	0	0	0	0	0	3	1	3	14	18	21	232	241	253	4065	4195	4167
95	0	0	0	0	0	0	1	1	0	17	18	9	246	266	248	4049	4180	4044
96	0	0	0	0	0	1	1	0	1	16	14	19	266	259	250	4157	4101	4171
97	0	0	0	0	1	0	2	2	2	20	12	9	263	276	278	4175	4211	4161
98	0	0	0	0	0	0	0	1	3	16	16	18	261	234	271	4080	4125	4249
99	0	0	0	0	1	0	0	1	1	15	13	20	256	289	268	4180	4071	4118
100	0	0	0	0	0	0	1	2	1	9	21	24	238	234	279	4039	4063	4077
101	0	0	0	0	0	0	2	0	2	16	20	9	235	261	284	4059	4078	4150
102	0	0	0	0	0	0	1	1	1	18	16	17	218	274	259	4173	4101	4037
103	0	0	0	0	0	0	1	0	0	19	27	14	271	255	245	4096	4223	4162
104	0	0	0	0	0	0	3	0	0	15	16	13	248	275	239	4011	4149	4080
105	0	0	0	0	0	0	1	1	0	16	11	16	286	246	253	4236	4103	4087
106	0	0	0	0	0	0	1	1	0	13	10	13	227	222	276	4072	3998	4150
107	0	0	0	0	0	0	0	1	1	12	16	21	238	243	274	4055	4035	4174
108	0	0	0	0	0	0	2	0	1	16	14	18	242	244	255	4062	4133	4080
109	0	0	0	0	0	0	2	0	2	8	12	18	207	240	264	4058	4039	4067
110	0	0	0	0	0	0	0	1	0	20	11	18	247	247	270	4037	4045	4136
111	0	0	0	0	0	0	1	0	3	12	16	17	267	272	259	4090	4084	4145
112	0	0	0	0	0	0	2	1	0	14	17	14	243	252	256	3933	4198	4095
113	0	0	0	0	0	0	0	0	3	11	15	15	254	231	286	4147	4031	4136
114	0	0	0	0	0	0	3	1	0	15	16	9	238	262	264	4138	4115	4251
115	0	0	0	0	0	0	0	0	1	18	9	12	262	236	214	4138	4050	4029
116	0	0	0	0	0	0	0	1	1	11	17	18	238	222	290	4144	4010	4108
117	0	0	0	0	0	0	1	1	4	9	23	17	272	261	267	4028	4040	4107
118	0	0	0	0	0	0	0	1	2	16	19	16	256	277	231	4057	4056	4102
119	0	0	0	0	0	0	1	1	1	15	19	20	255	269	251	3947	4172	4092
120	0	0	0	0	0	0	2	2	0	15	14	13	275	234	250	4098	4052	4053
121	0	0	0	0	1	0	1	1	2	19	20	18	257	257	259	4083	3965	4100
122	0	0	0	0	0	0	1	1	0	16	10	17	245	288	264	4042	4086	4105
123	**1**	**0**	**1**	0	0	0	3	0	1	22	22	12	267	232	257	4065	4177	4121
124	**0**	**0**	**0**	**8**	**7**	**11**	0	0	2	15	15	18	254	284	231	4021	4089	4099
125	**0,00813**	**0**	**0,00813**	**0**	**0**	**0**	**152**	**107**	**143**	7	15	12	265	231	260	4111	4026	4162
126				**0,064516**	**0,056452**	**0,08871**	**1**	**0**	**1**	**1949**	**2000**	**2042**	268	257	244	4132	3980	4045
127							**1,216**	**0,856**	**1,144**	**15**	**17**	**18**	**32417**	**32388**	**32773**	4133	4158	4053
										15,46825	**15,87302**	**16,20635**	**266**	**266**	**259**	**523827**	**524948**	**525285**
													255,252	**255,0236**	**258,0551**	**4138**	**4102**	**4131**
																4092,398	**4101,156**	**4103,789**

HEXADECIMAL c																		
Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	2	2	1	22	15	10	269	242	284	4213	4006	4115
1	0	0	0	0	0	0	0	1	0	17	15	18	287	247	270	4048	4018	4160
2	0	0	0	0	0	0	3	1	0	16	14	13	260	231	240	4194	4114	4075
3	0	0	0	0	0	1	1	2	4	18	23	15	251	229	235	4087	4041	4076
4	0	0	0	0	0	0	0	2	2	11	18	14	253	240	234	4097	4006	4128
5	0	0	0	0	0	0	0	0	2	14	18	18	257	260	278	4120	4120	4174
6	0	1	0	0	1	0	1	2	3	15	22	22	287	270	234	4261	4043	4090
7	0	0	0	0	1	0	0	2	1	15	19	12	244	281	271	4113	4086	4007
8	0	0	0	0	0	0	1	1	1	20	16	18	251	240	251	4050	4168	4220
9	0	0	0	0	0	0	0	2	1	12	21	12	273	316	248	4087	4141	4141
10	0	0	0	0	0	0	1	0	0	17	14	10	275	246	256	4262	4113	4166
11	0	0	0	0	0	0	3	3	0	10	19	12	253	254	234	4195	4112	4052
12	0	0	0	0	0	0	1	0	2	16	11	12	274	243	237	4144	4164	4007
13	0	0	0	0	0	0	0	1	0	11	16	17	250	259	267	4121	3978	4073
14	0	0	0	0	0	0	1	1	2	14	14	20	255	239	257	4023	4059	4165
15	0	0	0	0	0	0	0	1	0	19	16	9	228	221	248	4131	3996	4059
16	0	0	0	0	0	0	0	0	0	16	13	12	253	237	234	3983	4026	4107
17	0	0	0	0	0	0	0	2	2	16	21	13	250	242	252	4065	4054	4103
18	0	0	0	0	0	0	0	2	0	12	22	20	234	275	284	4088	4169	4075
19	0	0	0	0	0	0	1	0	1	16	16	21	237	232	268	4167	3979	4096
20	0	0	0	0	0	0	0	1	0	22	19	15	252	266	268	4060	4198	4129
21	0	0	0	0	0	0	2	1	0	18	15	16	251	254	251	4178	4146	4067
22	1	0	0	1	1	0	2	2	3	20	16	16	254	254	267	4127	4039	3960
23	0	0	0	1	0	0	1	2	2	13	16	20	258	264	266	4077	4103	4107
24	0	0	0	0	0	0	1	0	1	22	19	18	236	258	261	4055	4043	4116
25	0	0	0	0	0	0	0	3	0	19	19	14	260	239	266	4081	4120	4118
26	0	0	0	0	0	0	0	0	0	11	16	14	259	256	245	4063	4041	4136
27	0	0	0	0	0	0	0	0	1	15	17	13	274	249	279	4118	4023	4109
28	0	0	0	0	0	0	0	1	1	10	18	29	254	267	255	4190	4097	4033
29	0	0	0	0	0	0	1	0	1	23	12	20	260	247	268	4122	4094	4139
30	0	0	0	0	0	0	0	1	0	17	15	16	285	239	276	4158	4085	4098
31	0	0	0	0	0	0	0	3	0	18	21	9	279	260	230	4246	4144	4151
32	0	0	0	0	0	0	0	0	2	19	16	17	251	258	254	4060	4177	3993
33	0	0	0	0	0	0	0	0	2	11	15	17	253	263	279	4151	4119	4121
34	0	0	0	0	0	0	0	1	0	6	15	14	210	259	260	4175	4043	4115
35	0	0	0	1	0	0	1	1	1	14	10	17	255	233	236	4094	4000	4138
36	0	0	0	0	0	0	1	2	1	14	19	11	235	245	253	4141	4048	4087
37	0	0	0	0	0	0	3	0	3	13	24	18	253	266	267	3986	4180	4075
38	0	0	0	0	0	0	1	1	2	20	21	15	241	252	259	4072	4098	4084
39	0	0	0	0	0	0	1	0	1	16	14	17	271	243	275	4160	4064	4011
40	0	0	0	0	0	0	2	0	2	16	17	27	249	256	283	4069	4029	4092
41	0	0	0	0	0	0	0	1	0	17	22	11	250	264	234	4011	4109	4143
42	0	0	0	0	0	0	0	1	0	15	14	17	254	262	249	4115	3979	4016
43	0	0	0	0	0	0	1	0	3	12	12	16	259	249	250	4171	4042	4082
44	0	0	0	0	0	1	1	0	2	18	16	21	244	244	259	4066	4133	4113
45	0	0	0	0	0	0	0	0	2	16	16	10	243	256	263	3928	4114	4138
46	0	0	0	0	0	0	0	0	2	15	11	10	270	226	223	4039	4063	4093
47	0	0	0	0	0	0	2	0	0	20	9	12	233	262	242	4110	4028	4121
48	0	0	0	0	1	0	0	2	0	14	24	19	257	269	245	4003	4043	3978
49	0	0	0	0	0	0	1	1	0	16	27	12	254	259	268	4068	4121	4020
50	0	0	0	0	0	0	0	2	1	16	18	12	267	246	248	4101	4193	4225
51	0	0	0	0	0	0	0	0	2	11	16	15	280	266	264	4109	4079	4090
52	0	0	0	0	0	0	0	1	1	13	19	16	259	251	229	4148	4100	3990
53	0	0	0	0	0	1	1	1	1	11	16	21	249	242	256	4385	4113	4161
54	0	0	0	0	0	0	0	1	2	16	10	24	205	246	289	4139	4135	4096
55	0	0	0	0	0	0	1	0	3	20	16	21	262	248	315	4093	4104	4173
56	0	0	0	1	0	0	1	0	0	17	15	16	261	260	264	4125	3974	4249
57	0	0	0	0	0	0	2	0	1	13	17	14	267	234	266	4088	4071	4043
58	0	0	0	0	0	0	0	2	2	19	18	20	252	259	278	4051	4089	4200
59	0	0	0	1	0	0	2	1	0	20	13	16	268	261	245	4123	4105	4168
60	0	0	0	0	0	0	3	0	0	20	17	7	279	237	249	4162	4109	3977
61	0	0	0	0	0	0	1	0	0	14	16	10	244	253	233	4140	4114	4129
62	0	0	0	0	0	0	4	0	0	16	13	19	275	261	253	4162	4078	4111
63	0	0	0	0	0	0	2	1	2	19	13	9	268	249	239	4071	4062	4174
64	0	0	0	0	0	0	0	0	0	28	13	13	260	250	241	4129	3997	3963
65	0	0	0	0	0	0	0	1	2	5	17	16	234	261	267	3974	4153	4139
66	0	0	0	0	0	0	1	0	1	10	13	11	265	231	251	4120	4060	4087
67	0	0	0	0	0	0	1	1	1	18	8	16	243	237	271	4106	4059	4135
68	0	0	0	0	0	0	0	0	0	22	19	12	259	276	234	4080	4057	3953
69	1	0	0	1	0	0	1	2	0	20	9	13	258	258	249	4116	4048	4010
70	0	0	0	1	0	0	2	2	2	13	14	12	244	234	229	4078	4031	3989
71	0	0	0	0	0	0	1	0	2	21	15	18	258	257	249	4007	4079	4092
72	0	0	0	0	0	0	0	1	0	8	18	14	269	241	236	4137	4160	4065
73	0	0	0	0	0	0	1	1	0	11	9	12	247	258	267	4121	4147	4165
74	0	0	0	0	0	0	0	5	1	17	20	15	253	269	247	4072	4095	4001
75	0	0	0	0	0	0	1	1	1	15	20	13	259	257	298	4082	4014	4192
76	0	0	0	0	0	0	1	2	1	19	11	17	288	274	258	4260	4196	4145
77	0	0	0	0	0	0	0	0	0	12	21	16	265	238	251	4130	4116	4126
78	0	0	0	0	0	0	0	1	0	11	19	20	268	276	259	4075	4181	4069
79	0	0	0	0	0	0	0	0	2	9	8	14	246	245	257	4169	4062	4092
80	0	0	0	0	0	0	0	0	0	9	12	15	244	222	263	4020	4140	4062
81	0	0	0	0	0	0	0	1	0	21	17	12	275	262	253	4136	4021	4165
82	0	0	0	0	0	0	2	2	1	20	18	16	264	261	268	4078	4211	4203
83	0	0	0	0	0	0	1	2	0	11	15	18	253	234	279	4075	4033	4107
84	0	0	0	0	0	0	1	0	1	10	17	12	265	247	256	4143	4118	4079
85	0	0	0	0	0	0	0	2	0	9	19	18	248	261	268	4135	4041	4020
86	0	0	0	0	0	0	2	1	0	18	16	11	241	265	245	3947	4139	4122
87	0	0	0	0	0	0	2	1	1	14	12	22	261	263	273	4090	4004	3983
88	0	0	0	0	0	0	1	1	1	16	12	12	258	240	242	4036	4073	4083
89	0	0	0	0	0	0	0	1	3	13	9	16	223	233	262	3935	4010	4062
90	0	0	0	0	0	0	0	1	1	17	14	22	271	219	265	4095	4047	3951
91	0	0	0	0	0	0	2	3	2	23	22	12	297	257	279	4113	4152	4179
92	0	0	0	0	0	0	0	0	0	23	19	16	284	243	263	4073	4179	4152
93	0	0	0	0	0	0	1	0	0	17	14	13	281	285	259	4047	4148	4027
94	0	0	0	0	1	0	0	4	1	20	20	17	273	260	260	4166	4172	4083
95	0	0	0	0	0	0	1	3	1	11	15	6	264	246	252	4024	4104	4060
96	0	0	0	0	0	0	3	0	0	22	16	13	269	238	227	4163	4135	4075
97	0	0	0	1	0	0	2	1	1	21	18	13	273	238	247	4079	4042	4012
98	0	0	0	0	0	0	1	1	0	19	20	13	271	260	245	4105	4118	4008
99	0	0	0	0	0	0	1	1	0	19	18	10	281	233	243	4129	4078	4082
100	0	0	0	0	1	0	0	1	2	15	12	18	248	255	259	4050	4178	4154
101	0	0	0	0	0	0	0	1	0	10	17	18	252	242	268	4069	4014	4088
102	0	0	0	0	0	0	0	3	1	18	16	17	248	264	274	4086	4057	4128
103	0	0	0	0	0	0	1	1	1	12	17	19	245	268	252	4101	4091	4217
104	0	0	0	0	0	0	0	0	0	19	21	12	244	279	247	4008	4037	3948
105	0	0	0	0	0	0	2	1	0	20	15	19	268	263	253	4192	4067	4066
106	0	0	0	0	0	2	0	3	2	23	22	13	295	255	276	4060	4099	4134
107	0	0	0	0	1	0	0	1	5	11	14	21	250	233	258	4161	4092	4025
108	0	0	0	0	0	0	2	1	1	14	14	20	252	226	240	4115	4085	4046
109	0	0	0	0	0	0	1	1	0	21	18	13	263	247	260	4175	3981	4122
110	0	0	0	0	0	0	0	1	1	17	15	11	271	266	268	4167	3943	4154
111	0	0	0	0	0	0	1	0	2	17	10	12	270	244	244	4106	4068	4226
112	0	0	0	0	0	0	0	0	1	9	9	11	260	216	225	4130	4099	4113
113	0	0	0	0	0	0	1	3	4	21	16	23	239	242	258	4071	4183	4089
114	0	0	0	0	0	0	2	0	0	21	13	24	247	259	264	4117	4089	4098
115	0	0	0	0	0	0	0	2	1	15	18	11	257	261	225	4067	4182	4149
116	0	0	0	0	0	0	1	0	0	21	17	14	271	278	249	4113	4089	4041
117	0	0	0	1	0	0	3	0	0	23	21	10	270	248	253	4105	4226	4096
118	0	0	0	0	1	0	1	1	0	15	22	17	293	280	261	4064	4056	4042
119	0	0	0	0	1	0	2	2	1	20	16	16	268	251	240	4152	4172	4084
120	0	0	0	0	0	0	1	1	0	10	26	19	226	280	251	4110	4020	4100
121	0	0	0	0	0	0	2	3	0	12	23	7	227	287	244	3988	4178	3984
122	0	0	0	0	0	0	3	2	1	15	22	15	244	272	270	4118	4102	4142
123	2	1	0	0	0	0	3	0	0	16	12	8	256	265	253	4104	4115	4131
124	0	0	0	9	9	5	0	0	2	11	12	12	239	240	254	4072	4103	4075
125	0,01626	0,00813	0	0	0	0	108	128	120	12	13	19	233	264	252	4047	4259	3921
126				0,072581	0,072581	0,040323	0	1	0	2002	2053	1917	281	258	244	4098	4162	4162
127							0,864	1,024	0,96	16	16	12	32710	32108	32496	4118	4104	4080
										15,88889	16,29365	15,21429	253	261	268	525249	523513	523811
													257,5591	252,8189	255,874	4113	4043	4075
																4103,508	4089,945	4092,273

HEXADECIMAL d

Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	1	3	1	1	17	15	17	242	274	265	4079	4294	4019
1	0	0	0	0	0	0	1	2	1	21	15	11	250	242	267	4121	4170	4082
2	0	0	0	0	0	0	1	0	0	13	18	17	270	228	257	4162	4083	4137
3	0	0	0	0	1	0	0	2	0	8	13	10	247	252	262	4104	4073	4123
4	0	0	0	0	0	0	0	4	1	8	28	12	252	266	250	4113	4175	4053
5	0	0	0	0	0	0	0	0	0	13	15	19	250	282	300	4037	4081	4213
6	0	0	0	0	0	0	1	2	4	18	17	17	261	269	251	4209	4183	4236
7	0	0	0	0	0	0	1	2	2	22	16	21	274	256	249	4161	4156	4078
8	0	0	0	1	0	0	2	1	1	18	15	13	254	275	263	4138	4169	4043
9	0	0	0	1	0	0	2	0	2	15	11	15	255	236	234	4144	4064	4015
10	0	0	0	0	0	0	4	1	3	22	17	15	282	248	285	4155	3955	4177
11	0	0	0	0	0	0	0	0	0	17	11	22	262	258	259	4084	4057	4175
12	0	0	0	0	0	1	0	2	2	19	9	14	269	242	276	4051	3980	4137
13	0	0	0	1	0	0	4	1	1	17	15	12	261	240	239	4140	4176	4111
14	0	0	0	0	0	0	1	1	0	18	15	13	253	278	257	4164	4156	4113
15	0	0	0	0	0	0	1	0	3	11	17	14	253	233	265	3985	4189	4140
16	0	0	0	1	0	0	3	0	0	14	23	13	245	260	247	4119	3996	4079
17	0	0	0	0	0	0	3	0	2	20	11	14	242	263	250	4175	4041	4160
18	0	0	0	0	0	0	0	1	0	21	25	18	254	243	286	4152	4119	4068
19	0	0	0	0	0	0	0	0	2	14	13	19	266	284	257	4118	4052	4177
20	0	0	0	0	0	0	3	0	0	24	14	12	264	265	268	4146	4138	4029
21	0	0	0	0	0	0	0	1	0	13	21	14	272	257	247	3950	4134	4227
22	0	0	0	0	0	0	1	1	0	23	20	12	272	237	258	4164	4148	4014
23	0	0	0	0	0	0	1	1	1	18	14	18	256	237	234	4101	4107	4153
24	0	0	0	0	0	0	1	0	2	11	19	21	226	252	261	3997	4069	4084
25	0	0	0	0	0	0	0	1	2	15	10	21	245	249	272	4038	4024	4097
26	0	0	0	0	0	0	0	0	0	11	14	19	251	260	265	4146	4112	4146
27	0	0	0	0	0	0	1	4	1	11	18	12	247	243	280	4094	4151	4091
28	0	0	0	0	0	0	2	1	0	17	19	14	246	263	275	4119	4177	4177
29	0	0	0	0	0	0	3	1	1	27	22	11	246	283	220	4015	4103	4062
30	0	0	0	0	0	0	0	0	0	14	19	15	248	278	242	4071	4078	4087
31	0	0	0	0	0	0	0	0	0	18	14	16	275	236	242	4049	4188	3991
32	0	0	0	0	0	1	1	1	2	10	16	18	276	258	260	4067	4110	4030
33	0	0	0	0	0	0	1	0	2	14	17	18	243	294	241	4162	4142	4095
34	0	0	0	0	0	0	1	3	0	21	18	18	233	251	275	4044	3995	4105
35	0	0	0	0	0	0	0	0	1	13	14	18	254	233	281	4037	4055	4051
36	0	0	0	0	0	0	0	0	1	14	13	20	239	250	270	4151	4082	4156
37	0	0	0	0	0	0	0	1	0	12	15	15	245	257	294	4023	4087	4079
38	0	0	0	1	0	0	2	1	0	15	10	15	239	245	240	4064	3990	4160
39	0	0	0	1	0	0	2	0	2	12	16	21	228	233	252	4101	4024	4035
40	0	0	0	0	0	0	2	2	0	25	23	19	258	259	280	4095	4035	4236
41	0	0	0	0	0	0	1	1	3	15	16	13	284	234	224	4113	4150	4205
42	0	0	0	0	0	0	0	1	1	16	14	20	233	266	240	4088	4137	4034
43	0	0	0	0	0	0	0	1	0	16	14	14	244	268	257	4119	4125	4144
44	0	0	0	0	0	0	2	0	0	22	19	6	289	260	246	4232	4117	4054
45	0	0	1	0	0	1	0	1	2	17	26	13	266	290	245	4137	4154	4058
46	0	0	0	0	0	1	0	0	1	16	19	12	241	279	257	4153	4056	4001
47	0	0	0	0	0	0	0	2	1	11	12	16	266	233	281	4055	4056	4139
48	0	0	0	0	0	0	0	2	1	15	18	20	236	280	287	4105	4096	4141
49	0	0	0	0	0	1	1	0	2	9	23	18	211	271	252	4046	4197	4169
50	0	0	0	0	0	0	1	1	2	20	17	11	260	272	217	4112	4001	3983
51	0	0	0	0	0	0	0	0	0	15	16	15	245	255	254	4160	4145	3996
52	0	0	0	0	0	0	0	1	0	7	21	15	255	266	293	4035	4105	4160
53	0	0	0	0	0	0	1	0	1	18	14	17	248	236	268	4140	4161	4168
54	0	0	0	0	0	0	2	0	2	12	19	12	266	236	288	4013	4110	4257
55	0	0	0	0	0	0	0	2	0	18	13	10	254	305	223	4149	4150	4134
56	0	0	0	0	0	0	0	2	1	7	15	25	241	285	239	4166	4101	4092
57	0	0	0	0	0	0	0	1	1	8	19	12	240	264	269	4106	4142	4170
58	0	0	0	0	0	0	0	1	3	13	17	15	247	246	256	4108	4125	4159
59	0	0	0	0	0	0	0	0	0	13	12	11	224	241	258	4031	4025	4058
60	0	0	0	0	0	0	0	1	0	16	14	15	244	261	264	4008	4014	4141
61	0	0	0	0	0	0	1	0	0	18	10	13	285	247	221	4074	4165	4138
62	0	0	0	0	0	0	0	3	1	17	19	19	290	258	280	4170	4115	4041
63	0	0	0	0	0	0	0	0	0	14	11	12	262	269	251	4111	4054	4061
64	0	0	0	0	0	0	1	4	2	14	21	17	250	243	278	4128	4168	4174
65	0	0	0	0	0	1	1	0	2	13	21	23	263	279	286	4045	4231	4149
66	0	0	0	0	0	0	0	0	4	9	23	20	251	241	251	4149	4094	4282
67	0	0	0	0	0	0	1	0	0	27	11	20	265	250	267	4108	4049	3974
68	0	0	0	0	0	1	1	0	4	12	17	22	277	269	281	4185	4092	4116
69	0	0	0	0	0	2	0	1	4	17	14	22	239	252	242	4179	4086	4073
70	0	0	0	0	0	0	0	3	2	12	24	21	260	242	260	3924	4103	4065
71	0	0	0	0	0	0	0	1	0	13	13	19	239	258	280	4112	4077	4116
72	0	0	0	0	0	0	1	1	2	12	12	18	276	228	261	4166	4126	4198
73	0	0	0	0	0	0	1	0	2	20	20	19	248	269	237	4061	4119	4024
74	0	0	0	0	0	0	0	0	0	16	15	14	261	269	245	4096	4188	3983
75	0	0	0	0	0	0	1	2	1	17	15	14	245	263	222	4135	4199	4095
76	0	0	0	0	0	0	0	5	1	6	14	11	271	258	230	4112	4094	3991
77	0	0	0	0	0	0	0	0	0	10	17	15	242	240	247	4058	3970	4013
78	0	0	0	0	0	0	1	0	3	20	16	17	254	271	264	3969	4146	4112
79	0	0	0	0	0	0	2	0	1	18	17	17	258	257	278	4085	4105	4055
80	0	0	0	0	0	0	0	2	0	14	14	13	247	242	248	4192	4140	4118
81	0	0	0	0	0	0	0	1	1	17	23	12	249	244	225	4060	4074	4009
82	0	0	0	0	0	0	0	3	2	14	12	20	239	224	252	4116	4020	4063
83	0	0	0	0	0	0	0	1	1	8	20	15	237	280	262	4114	4108	3969
84	0	0	0	0	0	0	1	2	2	19	22	20	256	246	265	4156	4120	4089
85	0	0	0	0	1	0	0	2	1	20	15	15	265	265	284	4111	4151	4160
86	0	0	0	1	0	1	2	2	1	15	16	15	266	277	222	4076	4175	4087
87	0	0	0	0	0	0	4	1	4	18	17	13	252	239	251	4085	4035	4108
88	0	0	0	0	0	0	1	0	1	12	28	16	248	270	244	4083	3992	4112
89	0	0	0	0	0	0	2	1	2	13	11	16	242	262	275	4126	4049	4030
90	0	0	0	0	0	0	1	0	0	15	19	19	236	238	250	4069	4129	4135
91	0	0	0	0	0	0	0	0	0	16	10	10	229	278	264	4127	4108	4058
92	0	0	0	0	0	0	1	0	0	15	11	11	251	259	261	4108	4143	4160
93	0	0	0	0	0	0	1	0	0	20	14	12	269	262	237	4093	4139	4069
94	0	0	0	0	0	0	3	1	0	16	20	13	275	276	261	4113	4110	4197
95	0	0	0	0	0	0	1	2	0	12	17	14	236	237	233	3974	4128	4159
96	0	0	0	0	0	0	0	0	0	21	24	12	265	245	263	4029	4018	4117
97	0	0	0	0	0	0	1	0	1	13	22	12	272	240	263	4025	3982	4186
98	0	0	0	0	0	0	1	1	1	15	12	19	245	272	245	4088	4164	4007
99	0	0	0	0	0	0	0	2	0	18	17	14	252	250	236	4014	4029	4023
100	0	0	0	0	0	0	0	0	0	12	11	12	256	266	232	4131	4172	4129
101	0	0	0	0	0	0	2	0	2	18	16	15	239	244	251	4107	4171	4143
102	0	0	0	0	0	0	2	1	2	22	13	13	280	241	234	4070	4067	3964
103	0	0	1	0	0	1	1	2	2	22	20	23	274	274	257	4073	4069	4100
104	0	0	0	0	0	1	0	0	1	15	13	18	247	226	259	4182	4206	4050
105	0	0	0	0	0	0	2	1	1	21	14	11	267	235	259	4034	3993	4172
106	0	0	0	0	0	0	3	0	1	23	11	20	245	260	257	4141	4149	4211
107	0	0	0	0	0	0	3	1	2	28	13	20	272	239	256	4230	4039	4068
108	0	0	0	0	0	0	0	2	0	15	23	17	273	267	251	4054	4119	4079
109	0	0	0	0	0	0	1	0	1	20	16	17	266	274	295	4009	4044	4042
110	0	0	0	0	0	0	1	3	0	13	11	13	259	240	252	4036	4034	4100
111	0	0	0	0	0	0	4	2	0	16	27	15	259	269	276	4011	4125	4112
112	0	0	0	0	0	0	1	0	0	15	14	19	250	263	250	4016	4030	4158
113	0	0	0	0	0	0	1	1	1	16	17	12	230	260	256	4071	4165	4092
114	0	0	0	0	0	0	0	2	1	11	16	16	240	305	250	4138	4108	4116
115	0	0	0	0	0	0	2	1	4	13	14	23	262	274	260	4184	4151	4087
116	0	0	0	0	0	0	0	0	0	11	10	17	249	247	247	4186	4220	4117
117	0	0	0	0	0	0	2	1	0	18	14	15	263	220	232	4068	4011	3961
118	0	0	0	0	0	0	2	3	0	13	16	5	237	246	234	4035	4064	4099
119	0	0	0	0	0	0	2	0	2	16	21	19	252	249	268	4127	4033	4028
120	0	0	0	0	0	0	0	4	2	13	16	20	233	245	292	4104	4162	4209
121	0	0	0	0	0	0	1	0	2	15	22	21	261	275	241	4030	4153	4224
122	0	0	0	0	0	0	0	2	0	14	15	14	277	236	254	4150	4056	3997
123	**0**	**0**	**2**	0	0	0	1	2	0	13	15	11	245	253	255	4116	4029	4055
124	**0**	**0**	**0**	**7**	**2**	**13**	0	1	0	14	20	14	237	263	248	4029	4108	4133
125	**0**	**0**	**0,01626**	**0**	**0**	**0**	**117**	**125**	**131**	15	13	12	272	257	263	4239	4032	4079
126				**0,056452**	**0,016129**	**0,104839**	**0**	**0**	**0**	**1966**	**2068**	**1979**	240	252	250	4057	4122	4112
127							**0,936**	**1**	**1,048**	**13**	**14**	**12**	**32251**	**32566**	**32595**	4093	4182	4092
										15,60317	**16,4127**	**15,70635**	**245**	**269**	**257**	**524468**	**525094**	**524949**
													253,9449	**256,4252**	**256,6535**	**4113**	**4108**	**4160**
																4097,406	**4102,297**	**4101,164**

HEXADECIMAL e																		
Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	0	0	0	3	0	3	19	15	18	256	279	266	4070	4165	4017
1	0	0	0	0	0	0	0	1	0	19	13	21	248	245	269	4152	4179	4115
2	0	0	0	0	1	0	0	1	0	11	18	13	263	277	244	4135	4022	4176
3	0	0	0	0	0	0	0	3	3	19	12	14	257	247	255	4081	4095	4095
4	0	0	0	0	0	0	0	1	4	9	21	15	252	285	266	4161	4035	4015
5	0	0	0	0	0	0	2	0	1	17	23	16	249	249	243	4092	4196	4086
6	0	0	0	0	0	0	0	1	0	18	17	14	244	248	236	4062	4077	4015
7	0	0	0	0	0	0	2	1	0	18	19	12	259	231	242	4084	4124	4076
8	0	0	0	0	0	0	1	1	0	25	16	11	263	306	252	4105	4095	4111
9	0	0	0	0	0	0	0	0	1	14	18	17	259	277	276	4070	4145	4161
10	0	0	0	0	0	0	1	1	1	13	18	20	232	274	242	4029	4167	4182
11	0	0	0	0	0	0	0	1	1	17	15	24	267	261	264	4165	4264	4072
12	0	0	0	0	0	1	3	0	3	20	18	22	280	265	261	4174	4105	4181
13	0	0	0	0	0	0	0	1	1	15	17	12	251	272	265	4153	4073	4022
14	0	0	0	0	0	0	2	5	0	18	18	9	244	275	226	4066	4044	4064
15	0	0	0	0	0	0	2	1	0	11	24	21	228	259	275	4141	4012	4135
16	0	0	0	0	0	0	1	0	1	14	9	18	231	265	256	4017	4105	4181
17	0	0	0	0	0	0	0	2	0	12	19	15	245	257	277	3946	4061	4084
18	0	0	0	0	0	0	0	1	0	16	20	13	237	240	276	4068	4012	4213
19	0	0	0	0	0	0	0	1	0	8	23	15	255	305	244	4064	4017	4066
20	0	0	0	0	0	0	2	1	0	13	18	12	261	261	237	4091	4095	4010
21	0	0	0	0	0	0	1	1	0	14	15	14	209	268	261	3994	4161	4053
22	0	0	0	0	1	0	0	1	0	13	19	16	257	256	234	4039	4111	4100
23	0	0	0	0	0	0	1	1	1	14	12	18	243	233	264	4015	4075	4018
24	0	0	0	0	0	0	2	1	3	22	13	21	263	239	271	4124	4144	4117
25	0	0	0	0	0	0	1	1	2	10	12	16	268	274	262	4096	4044	4097
26	0	0	0	0	0	0	1	0	0	14	15	19	243	259	275	4186	4220	4085
27	0	0	0	1	0	0	1	0	0	14	10	14	270	284	246	4088	4129	4020
28	0	0	0	0	0	0	2	1	0	17	16	16	265	258	266	4069	4087	4068
29	0	0	0	0	0	0	0	1	1	17	15	8	250	234	258	4086	4017	4044
30	0	0	0	0	0	0	3	0	1	19	11	15	265	255	242	4181	4182	4071
31	0	0	0	0	0	0	0	0	1	12	15	18	246	241	270	4062	4170	4169
32	0	0	0	0	1	0	1	2	1	17	12	15	250	272	248	4023	4083	4101
33	0	0	0	0	0	0	1	1	0	9	11	8	230	223	257	4084	4016	4169
34	0	0	0	0	0	0	1	1	1	12	13	8	261	249	235	4203	4092	4057
35	0	0	0	0	0	0	2	0	1	16	26	17	273	271	264	4287	4111	3986
36	0	0	0	0	0	0	0	0	4	16	18	21	265	278	276	4107	4073	4011
37	0	0	0	0	0	0	1	0	1	16	24	27	247	288	269	4198	4122	4148
38	0	1	0	0	2	0	0	3	2	27	18	18	241	263	285	4071	4127	4208
39	0	1	0	0	2	0	0	3	0	13	19	17	257	258	258	4048	4020	4173
40	0	1	0	0	1	0	1	2	4	11	17	21	245	265	296	4231	4115	4151
41	0	0	0	0	1	0	1	1	1	17	12	14	241	262	279	4035	4164	4148
42	0	0	0	0	0	0	0	2	1	17	17	11	278	231	281	4161	4062	4149
43	0	0	0	0	0	0	0	0	1	11	15	15	215	268	274	4164	4175	4154
44	0	0	0	1	0	0	3	1	2	14	14	22	250	253	283	4129	4133	4140
45	0	0	0	0	0	0	4	0	0	22	14	15	273	247	297	4018	3968	4143
46	0	0	0	0	0	0	1	0	1	22	16	21	249	245	239	3987	4061	4089
47	0	0	0	0	0	0	0	0	1	19	13	16	284	268	258	4093	4102	4211
48	0	0	0	0	0	0	0	1	1	9	21	19	269	239	247	4160	4026	4132
49	0	0	0	0	0	0	1	0	0	16	15	22	274	251	268	4125	4138	4077
50	0	0	0	0	0	0	0	0	0	14	18	14	241	270	271	4138	4072	4120
51	0	0	0	0	0	0	1	1	1	18	14	15	251	260	231	4031	4115	4043
52	0	0	0	0	0	0	0	2	1	11	15	12	243	263	239	4158	4111	4147
53	0	0	0	0	0	0	1	0	1	15	17	14	246	227	231	3983	4095	4065
54	0	0	0	0	0	1	0	0	2	7	8	26	244	247	265	4078	4123	3997
55	0	0	0	1	0	0	3	0	2	14	21	12	255	257	278	4052	4129	4139
56	0	0	0	0	0	0	1	1	1	21	13	15	257	268	224	4143	4099	4014
57	0	0	0	0	0	0	3	1	1	16	16	15	283	251	242	4133	4153	4145
58	0	0	0	0	0	0	1	1	2	20	14	16	264	251	259	4116	4125	4070
59	0	0	0	0	0	0	0	0	1	19	18	24	257	256	233	4118	4086	4091
60	0	0	0	0	0	0	2	0	0	13	9	15	248	260	279	4072	4122	4160
61	0	0	0	0	0	0	1	0	1	24	13	18	237	248	261	4140	4009	4121
62	0	0	0	0	0	0	1	2	0	12	21	12	255	266	233	4065	4102	4005
63	0	0	0	1	0	0	2	0	0	10	14	19	230	263	266	4134	3988	4018
64	0	0	0	0	0	0	3	1	0	15	17	13	251	275	244	4040	4013	4151
65	0	0	0	0	0	0	0	1	2	18	24	12	249	264	244	4089	4095	4091
66	0	0	0	0	0	0	3	2	1	12	14	11	264	248	225	4112	4092	3954
67	0	0	0	0	0	0	1	1	1	17	23	16	256	276	255	4104	4103	4077
68	0	0	0	0	0	0	1	2	0	13	15	14	227	266	259	4177	4088	3963
69	0	0	0	0	0	0	2	1	2	21	15	17	246	245	257	4026	4112	4190
70	0	0	0	0	0	0	1	0	2	20	8	14	265	261	263	4172	4042	4068
71	0	0	0	0	0	0	1	2	2	22	24	15	266	263	243	4120	4111	4156
72	0	0	0	0	0	0	2	1	2	21	12	14	251	248	248	4083	4042	4114
73	0	0	0	0	2	1	1	2	2	15	18	17	264	261	253	4102	4058	4121
74	0	0	0	0	0	0	3	3	4	16	16	20	273	281	235	4056	4123	4269
75	0	0	0	0	0	0	3	1	1	21	23	18	271	271	258	4111	4156	4106
76	0	0	0	0	0	0	0	1	1	20	19	15	247	270	265	4101	4051	4113
77	0	0	0	0	0	0	1	2	1	18	19	14	244	266	266	4008	4079	4112
78	0	0	0	0	0	0	0	1	1	12	18	14	236	270	266	4022	4032	4148
79	0	0	0	0	1	0	0	2	2	14	15	15	277	259	259	4105	4129	4156
80	0	0	0	0	0	0	0	3	1	17	17	20	245	253	284	4153	4009	4083
81	0	0	0	0	0	0	0	0	3	15	26	16	278	279	256	4040	4146	4133
82	0	0	0	0	0	0	1	0	1	21	8	15	277	270	259	4179	4028	4073
83	0	0	0	0	0	0	2	3	2	15	17	14	256	238	252	4171	4117	4123
84	0	0	0	0	0	0	1	0	2	21	21	20	248	272	263	4026	4050	4080
85	0	0	0	0	1	0	0	1	0	17	16	24	234	279	293	4092	4078	4120
86	0	0	0	0	0	0	0	1	0	12	12	15	259	260	262	4214	4082	4102
87	0	0	0	0	0	1	1	0	1	18	14	12	260	264	273	4100	4132	4161
88	0	0	0	0	0	0	0	1	2	16	19	12	250	243	238	4069	4092	4080
89	0	0	0	0	0	1	0	1	2	13	18	23	273	261	251	4133	4063	4100
90	0	0	0	1	0	0	1	0	3	16	13	17	262	243	256	4119	4006	4062
91	0	0	0	0	0	1	1	0	2	13	14	15	242	249	259	4119	4034	4100
92	0	0	0	0	0	0	1	0	1	25	18	19	259	244	246	4195	3995	4029
93	0	0	0	1	0	0	2	1	0	13	12	24	240	241	281	4039	4084	4097
94	0	0	0	0	0	0	1	1	1	15	12	19	254	249	265	3898	4135	4062
95	0	0	0	0	0	0	2	2	0	9	19	12	219	273	265	4113	4087	4056
96	0	0	0	0	0	0	0	0	0	19	17	9	242	233	254	4145	4006	4042
97	0	0	0	0	0	0	1	0	0	18	14	15	259	263	242	4160	4043	4201
98	0	0	0	0	0	0	0	0	2	11	15	17	264	232	253	4139	4002	4155
99	0	0	0	0	0	0	0	1	2	13	20	20	259	268	252	4050	4180	4063
100	0	0	0	0	0	1	0	1	2	9	17	15	251	244	218	4094	4074	4022
101	0	0	0	0	0	0	0	0	3	13	17	17	239	248	251	4027	4127	4025
102	0	0	0	0	0	0	2	1	0	7	19	19	224	272	261	4084	4117	4049
103	0	0	0	0	0	0	0	0	0	12	18	15	218	282	263	4017	4184	4150
104	0	0	0	0	0	0	1	0	1	13	17	21	277	257	269	4030	4065	4114
105	0	0	0	0	0	0	0	2	2	11	17	12	264	267	247	4172	4090	4113
106	0	0	0	0	0	1	1	1	3	11	17	20	233	279	283	4044	4151	4104
107	0	0	0	0	0	0	0	2	3	16	16	18	243	225	257	4055	4034	4180
108	0	0	0	0	0	0	0	2	4	14	18	19	251	255	284	4028	4061	4109
109	0	0	0	0	0	0	2	2	1	19	18	22	279	275	285	4092	4062	4266
110	0	0	0	0	0	0	0	0	3	12	19	13	253	267	256	4171	4196	4118
111	0	0	0	0	0	0	0	1	1	13	14	25	261	279	230	4144	4067	4169
112	0	0	0	0	0	0	0	0	0	14	14	12	247	259	267	4101	4164	4108
113	0	0	0	0	0	0	0	1	4	11	12	18	233	248	275	4161	4138	4145
114	0	0	0	0	0	0	2	1	2	14	16	17	227	250	256	4053	4024	4127
115	0	0	0	0	0	0	0	1	1	17	15	16	256	250	246	4096	4052	4039
116	0	0	0	0	0	0	2	1	1	10	19	18	249	265	244	4058	4120	4077
117	0	0	0	0	0	0	0	1	0	15	23	7	262	274	251	4114	4078	4131
118	0	0	0	0	0	0	1	1	1	14	9	6	273	246	240	4054	4112	4124
119	0	0	0	0	0	0	1	1	1	14	12	17	233	230	260	4029	4058	4085
120	0	0	0	0	0	0	2	0	0	9	12	17	244	233	250	4095	4059	4020
121	0	0	1	0	0	2	1	1	5	24	18	21	276	282	259	4124	4141	4145
122	0	0	0	0	0	1	0	1	2	16	8	24	263	253	238	4030	4129	4070
123	0	3	1	1	0	0	3	2	1	18	13	18	264	241	274	4049	3956	4163
124	0	0	0	7	13	11	2	1	0	25	16	19	262	280	254	4081	4024	4026
125	0	0,02439	0,00813	0	0	0	120	117	153	18	16	16	254	237	284	4109	4044	4150
126				0,056452	0,104839	0,08871	0	1	1	1951	2042	2060	264	236	237	4093	4013	4125
127							0,96	0,936	1,224	14	18	15	32140	32869	32730	4033	4074	4087
										15,48413	16,20635	16,34921	264	248	266	524171	523454	525053
													253,0709	258,811	257,7165	4161	4095	4100
																4095,086	4089,484	4101,977

HEXADECIMAL f

Position	6 repetitions			5 repetitions			4 repetitions			3 repetitions			2 repetitions			1 repetition		
	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria	SHA2-512	SHA3-512	Viktoria
0	0	0	0	1	0	0	2	1	3	17	18	16	270	291	279	4180	4134	4274
1	0	0	0	0	0	0	1	1	2	16	14	18	250	255	279	4034	4101	4041
2	0	0	0	0	1	1	0	1	1	13	19	10	240	274	248	4117	4056	4123
3	0	0	0	0	0	0	0	1	2	18	13	11	246	262	263	3950	4072	4129
4	0	0	0	0	0	0	2	3	1	19	15	15	264	260	236	4183	4214	4103
5	0	0	0	0	1	0	2	3	0	15	16	19	285	269	279	4026	4118	4090
6	0	0	0	0	0	0	2	1	1	14	17	20	245	266	262	4102	4125	4123
7	0	0	0	0	0	0	0	0	1	12	15	15	241	225	255	4066	4091	4108
8	0	0	0	1	0	0	2	1	0	18	13	12	250	242	254	4076	3945	4128
9	0	0	0	0	0	0	2	2	1	22	16	22	263	241	277	4098	4159	4089
10	0	0	0	0	0	0	0	0	1	14	18	16	243	266	243	4020	4107	4146
11	0	0	0	0	0	0	0	1	0	14	11	14	235	229	256	3998	4115	4115
12	0	0	0	0	0	0	3	1	1	15	10	11	260	220	244	4151	4114	4079
13	0	0	0	0	0	0	0	0	1	14	10	12	260	262	253	4072	4073	4128
14	0	0	0	0	0	0	0	3	1	13	28	19	279	271	242	4090	4148	4102
15	0	0	0	0	0	0	3	1	0	18	16	15	257	279	250	4152	4026	4017
16	0	0	0	0	0	0	0	0	0	18	7	20	248	249	241	4044	4192	4129
17	0	0	0	0	0	0	2	0	2	18	14	19	255	221	285	4049	4070	4085
18	0	0	0	0	0	0	0	1	0	14	14	13	259	240	248	4097	4026	4025
19	0	0	0	1	0	0	1	1	0	14	24	13	245	256	277	4114	4117	4049
20	0	0	0	0	0	0	1	0	0	14	17	7	230	251	272	4043	4174	4173
21	0	0	0	0	0	0	0	1	2	22	18	13	270	288	254	4088	4098	4077
22	0	0	0	0	0	0	1	0	0	17	10	14	262	244	259	4105	4011	4051
23	0	0	0	0	0	0	0	0	1	19	13	12	257	280	251	3998	4204	4201
24	0	0	0	0	0	0	0	2	3	17	14	18	272	246	245	4216	4088	4076
25	0	0	0	0	0	0	0	0	1	14	14	20	245	259	293	4170	4073	4161
26	0	0	0	0	0	0	0	0	1	16	10	20	262	239	275	4194	3986	4180
27	0	0	0	0	0	0	1	4	1	19	19	19	268	256	280	4001	4127	4218
28	0	0	0	0	0	0	1	2	1	15	17	13	289	264	245	4120	4064	4099
29	0	0	0	0	0	0	0	1	1	22	14	14	270	271	245	4161	4134	4110
30	0	0	0	0	0	0	0	0	0	10	13	10	259	254	262	4020	4059	4038
31	0	0	0	0	0	0	0	0	1	14	21	15	232	249	255	4098	4053	4055
32	0	0	0	0	0	0	0	1	1	10	14	14	259	257	258	4022	4079	4102
33	0	0	0	0	0	0	1	0	0	17	12	18	249	230	279	4049	4058	4172
34	0	0	0	0	0	0	0	1	0	15	13	9	250	244	284	4144	4043	4125
35	0	0	0	0	0	0	0	1	0	16	27	11	241	294	251	4005	4161	4125
36	0	0	0	0	0	0	0	1	2	15	12	9	228	262	271	4050	4108	4108
37	0	0	0	0	0	0	2	0	1	14	22	13	267	245	257	4056	4104	4196
38	0	0	0	0	0	0	0	1	0	15	11	16	239	264	246	4098	4042	4070
39	0	0	0	0	0	0	1	1	2	14	15	14	246	218	274	4106	4057	4032
40	0	0	0	0	0	0	4	2	1	23	11	19	249	252	262	4045	4111	4199
41	0	0	0	0	0	0	0	1	4	26	23	15	265	252	251	4142	4093	3989
42	0	0	0	0	1	0	1	1	1	17	17	17	274	266	280	4080	4146	4073
43	0	0	0	0	0	0	0	3	4	11	13	24	253	241	275	4011	4026	4097
44	0	0	0	0	0	0	0	2	2	22	24	20	258	268	278	4038	4116	4179
45	0	0	0	0	0	0	1	1	1	20	13	27	283	292	277	4031	4056	4103
46	0	0	0	1	0	0	1	0	1	15	14	14	254	228	254	4191	4069	4011
47	0	0	0	0	0	0	3	1	2	23	16	19	253	256	271	4084	4022	4096
48	0	0	0	0	0	0	0	0	0	16	13	28	281	249	263	4160	4106	4090
49	0	0	0	0	0	0	0	2	1	15	15	16	260	244	263	4158	4008	4072
50	0	0	0	0	0	0	0	1	0	10	22	17	254	265	264	4063	4024	4094
51	0	0	0	0	0	0	1	1	1	17	18	23	240	269	267	3966	4117	4192
52	0	0	0	0	0	0	2	0	1	21	14	19	255	262	289	4097	4093	4195
53	0	0	0	0	0	0	0	4	0	16	24	12	242	262	246	4002	4016	4117
54	0	0	0	0	0	0	0	1	1	18	15	13	262	266	263	4123	4173	4123
55	0	0	0	0	0	0	1	0	1	9	10	16	263	242	257	4144	4075	4039
56	0	0	0	0	0	0	2	0	1	19	14	17	243	266	252	4100	4079	4150
57	0	0	0	0	0	0	1	0	1	13	18	17	246	266	240	4126	4113	4214
58	0	0	0	0	0	0	1	1	1	12	17	11	255	283	260	4067	4071	4086
59	0	0	0	0	0	0	1	1	0	22	12	21	241	240	242	4093	4056	4062
60	0	0	0	0	0	0	1	0	0	11	9	15	257	227	245	4172	4077	3987
61	0	0	0	0	0	0	0	0	0	18	19	15	235	256	245	4121	4129	4115
62	0	0	0	0	0	0	0	1	2	10	18	11	261	260	250	3994	4056	4178
63	0	0	0	0	0	0	0	0	1	22	18	12	256	256	267	4070	4033	4108
64	0	0	0	0	0	0	1	1	1	13	26	20	253	256	250	4117	4105	4153
65	0	0	0	1	0	0	3	0	0	26	24	17	261	285	271	4140	4109	4183
66	0	0	0	0	0	0	3	0	1	16	17	10	254	244	232	4053	4146	4071
67	0	0	0	1	0	0	1	0	1	15	15	13	249	249	244	4053	4019	4140
68	0	0	0	0	0	0	2	0	0	16	9	16	267	237	228	4217	4209	3985
69	0	0	0	0	0	0	2	0	1	16	14	16	261	239	269	4074	4101	4063
70	0	0	0	0	0	0	1	1	3	16	10	15	245	243	240	4174	4138	4082
71	0	0	0	0	0	0	2	0	2	14	15	20	249	269	255	3951	4004	4117
72	0	0	0	0	0	0	1	1	2	11	20	16	273	246	254	4089	4083	4149
73	0	0	0	0	0	0	1	1	1	16	15	10	257	272	249	4141	4068	4017
74	0	0	0	0	0	0	1	0	2	19	17	13	265	248	238	4143	4081	4025
75	0	0	0	0	0	0	3	0	0	15	9	15	229	254	255	3979	4160	4067
76	0	0	0	0	1	0	0	1	0	20	17	8	237	268	267	3993	4079	4128
77	0	0	0	0	0	0	1	1	0	15	18	11	247	235	248	4047	4107	4078
78	0	0	0	0	0	0	0	2	0	19	22	20	287	271	271	4132	4004	4039
79	0	0	0	0	0	0	1	1	1	18	16	13	258	248	266	4066	4185	4132
80	0	0	0	0	0	0	1	1	0	24	16	12	263	257	228	4125	4108	4044
81	0	0	0	0	0	0	0	0	1	20	16	21	243	239	273	4154	4041	4162
82	0	0	0	0	0	0	0	1	1	14	19	12	271	293	240	4044	4067	4116
83	0	0	0	0	0	0	1	0	0	11	16	18	248	270	240	4068	4179	3966
84	0	0	0	0	0	0	2	1	1	16	15	14	276	253	258	4280	4102	4088
85	0	0	0	0	1	0	0	2	2	10	18	14	238	246	249	4138	4070	4101
86	0	0	0	0	0	0	0	2	1	17	17	16	249	296	274	4159	4170	4183
87	0	0	0	1	0	0	1	0	1	12	19	11	244	249	257	4083	4245	4044
88	0	0	0	0	0	0	1	1	1	15	14	13	231	256	235	4235	3975	4115
89	0	0	0	0	0	0	1	1	1	10	13	18	224	253	234	4121	4156	4133
90	0	0	0	0	0	0	0	1	2	11	18	25	253	238	292	3965	4002	4141
91	0	0	0	0	0	0	0	0	0	21	12	10	235	258	218	3974	4191	4023
92	0	0	0	0	0	0	4	0	2	17	13	14	273	224	247	4085	4117	4120
93	0	0	0	0	0	0	1	0	1	19	18	15	256	268	246	4215	4031	3974
94	0	0	0	0	0	0	1	0	2	20	12	12	252	253	259	4235	4074	4162
95	0	0	0	0	0	0	0	0	1	17	23	16	269	250	259	4169	4084	4081
96	0	0	0	0	0	0	3	0	0	21	19	14	267	263	255	4151	4143	4051
97	0	0	0	0	0	0	0	0	1	13	18	11	262	262	275	4127	3985	4029
98	0	0	0	0	0	0	0	1	1	11	19	18	246	263	241	4127	4103	4150
99	0	0	0	0	0	0	1	1	0	18	16	18	248	255	256	4092	4097	4135
100	0	0	0	0	0	0	0	0	2	14	15	11	278	243	266	4063	4119	4003
101	0	0	0	0	0	0	1	1	0	9	19	21	245	236	258	4008	4155	4162
102	0	0	0	0	1	0	1	2	1	14	18	17	244	254	264	4084	3954	4126
103	0	0	0	0	0	0	1	1	1	18	8	11	261	245	245	4111	4064	4140
104	0	0	0	0	0	0	1	1	1	17	16	14	276	225	266	4208	4081	4041
105	0	0	0	1	0	0	1	1	1	15	21	17	265	265	250	4074	4084	4041
106	0	0	0	0	0	0	4	2	1	22	19	11	246	289	260	4132	4137	4107
107	0	0	0	0	0	0	0	2	1	23	14	16	277	259	287	4103	4142	4248
108	0	0	0	0	1	0	2	2	2	14	19	16	271	259	258	4161	4046	4150
109	0	0	0	0	0	0	1	2	0	12	12	25	258	252	279	4093	4150	4244
110	0	0	0	1	0	0	1	1	0	21	18	16	257	265	266	4037	4027	4069
111	0	0	0	0	0	1	1	2	1	23	23	9	272	268	241	4133	4073	4075
112	0	0	0	0	2	0	2	3	1	15	30	14	269	268	228	4138	4011	4101
113	0	0	0	0	0	0	1	4	1	23	16	9	273	247	247	4128	3988	4072
114	0	0	0	0	0	0	1	1	1	14	24	13	268	253	254	4184	4017	4098
115	0	0	0	0	0	0	0	2	0	15	21	18	246	277	257	4143	4019	4171
116	0	0	0	0	0	0	1	0	1	16	16	18	242	269	262	4030	4237	4063
117	0	0	0	0	0	0	0	0	1	11	12	20	256	257	265	4058	4080	4034
118	0	0	0	0	0	0	1	2	0	20	18	16	281	276	256	4150	4121	4002
119	0	0	0	0	0	0	0	1	0	12	20	15	234	270	256	4078	4056	4105
120	0	0	0	0	0	0	0	0	1	10	19	15	244	259	251	4076	4153	4146
121	0	0	0	0	0	0	0	0	0	11	14	15	249	254	254	4144	4133	4040
122	0	0	0	0	0	0	0	1	2	11	16	14	237	234	246	4110	4034	4062
123	**0**	**0**	**1**	0	0	1	1	1	1	12	17	19	243	237	255	4121	4048	4005
124	**0**	**0**	**0**	**9**	**9**	**13**	0	0	2	19	20	13	254	246	254	4126	4042	4213
125	**0**	**0**	**0,00813**	**0**	**0**	**0**	**111**	**114**	**129**	14	5	19	247	257	252	4080	4063	3992
126				**0,072581**	**0,072581**	**0,104839**	**0**	**1**	**0**	**2024**	**2046**	**1971**	231	269	243	4050	4103	3994
127							**0,888**	**0,912**	**1,032**	**14**	**18**	**15**	**32394**	**32504**	**32407**	4096	4119	4087
										16,06349	**16,2381**	**15,64286**	**246**	**256**	**250**	**524276**	**523385**	**523638**
													255,0709	**255,937**	**255,1732**	**4098**	**4056**	**4021**
																4095,906	**4088,945**	**4090,922**

ANNEX XVI - DIFFERENTIAL TEST SHA2-512

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                    arqsha512.bin|  1.86e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.83448560|  PASSED
      diehard_operm5|   0|   1000000|     100|0.00000000|  FAILED
  diehard_rank_32x32|   0|     40000|     100|0.00000000|  FAILED
    diehard_rank_6x8|   0|    100000|     100|0.01278267|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.00000000|  FAILED
        diehard_opso|   0|   2097152|     100|0.00000000|  FAILED
        diehard_oqso|   0|   2097152|     100|0.00000000|  FAILED
         diehard_dna|   0|   2097152|     100|0.00000000|  FAILED
diehard_count_1s_str|   0|    256000|     100|0.00000000|  FAILED
diehard_count_1s_byt|   0|    256000|     100|0.00000000|  FAILED
 diehard_parking_lot|   0|     12000|     100|0.02499223|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.03608086|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.01386186|  PASSED
     diehard_squeeze|   0|    100000|     100|0.00000000|  FAILED
        diehard_sums|   0|       100|     100|0.02858330|  PASSED
        diehard_runs|   0|    100000|     100|0.67993300|  PASSED
        diehard_runs|   0|    100000|     100|0.74187019|  PASSED
       diehard_craps|   0|    200000|     100|0.98702385|  PASSED
       diehard_craps|   0|    200000|     100|0.05449752|  PASSED
# The file file_input_raw was rewound 1 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000005|  FAILED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 1 times
         sts_monobit|   1|    100000|     100|0.72680883|  PASSED
# The file file_input_raw was rewound 1 times
            sts_runs|   2|    100000|     100|0.47519137|  PASSED
# The file file_input_raw was rewound 1 times
          sts_serial|   1|    100000|     100|0.86221513|  PASSED
          sts_serial|   2|    100000|     100|0.73870943|  PASSED
          sts_serial|   3|    100000|     100|0.23767587|  PASSED
          sts_serial|   3|    100000|     100|0.30569328|  PASSED
          sts_serial|   4|    100000|     100|0.40966424|  PASSED
          sts_serial|   4|    100000|     100|0.63351788|  PASSED
          sts_serial|   5|    100000|     100|0.99439456|  PASSED
          sts_serial|   5|    100000|     100|0.77496269|  PASSED
          sts_serial|   6|    100000|     100|0.92133647|  PASSED
          sts_serial|   6|    100000|     100|0.94733462|  PASSED
          sts_serial|   7|    100000|     100|0.05681285|  PASSED
          sts_serial|   7|    100000|     100|0.60674224|  PASSED
          sts_serial|   8|    100000|     100|0.38066091|  PASSED
          sts_serial|   8|    100000|     100|0.76398194|  PASSED
          sts_serial|   9|    100000|     100|0.77530653|  PASSED
          sts_serial|   9|    100000|     100|0.49748204|  PASSED
          sts_serial|  10|    100000|     100|0.97044817|  PASSED
          sts_serial|  10|    100000|     100|0.82677034|  PASSED
          sts_serial|  11|    100000|     100|0.21680438|  PASSED
          sts_serial|  11|    100000|     100|0.44704932|  PASSED
          sts_serial|  12|    100000|     100|0.05125624|  PASSED
          sts_serial|  12|    100000|     100|0.01036208|  PASSED
          sts_serial|  13|    100000|     100|0.23719047|  PASSED
          sts_serial|  13|    100000|     100|0.98417666|  PASSED
          sts_serial|  14|    100000|     100|0.44217216|  PASSED
          sts_serial|  14|    100000|     100|0.99259309|  PASSED
          sts_serial|  15|    100000|     100|0.30071199|  PASSED
          sts_serial|  15|    100000|     100|0.95603587|  PASSED
          sts_serial|  16|    100000|     100|0.56234633|  PASSED
          sts_serial|  16|    100000|     100|0.97638119|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   1|    100000|     100|0.94697962|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   2|    100000|     100|0.71201834|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   3|    100000|     100|0.41452534|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   4|    100000|     100|0.37910137|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   5|    100000|     100|0.07121991|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   6|    100000|     100|0.84719197|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   7|    100000|     100|0.56924067|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   8|    100000|     100|0.49002629|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   9|    100000|     100|0.42975163|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  10|    100000|     100|0.15682101|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  11|    100000|     100|0.45328737|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  12|    100000|     100|0.00299095|   WEAK
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   2|     10000|    1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   3|     10000|    1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   4|     10000|    1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   5|     10000|    1000|0.00000000|  FAILED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
# The file file_input_raw was rewound 2 times
    rgb_permutations|   2|    100000|     100|0.47947119|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   3|    100000|     100|0.38748372|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   4|    100000|     100|0.00000001|  FAILED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   5|    100000|     100|0.00003448|   WEAK
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   0|   1000000|     100|0.15561073|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   1|   1000000|     100|0.02004180|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   2|   1000000|     100|0.73701086|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   3|   1000000|     100|0.16053374|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   4|   1000000|     100|0.89435528|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   5|   1000000|     100|0.86439810|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   6|   1000000|     100|0.14518173|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   7|   1000000|     100|0.56126854|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   8|   1000000|     100|0.03441880|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   9|   1000000|     100|0.48995703|  PASSED
# The file file_input_raw was rewound 5 times
      rgb_lagged_sum|  10|   1000000|     100|0.69863826|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  11|   1000000|     100|0.08762853|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  12|   1000000|     100|0.56493254|  PASSED
# The file file_input_raw was rewound 7 times
      rgb_lagged_sum|  13|   1000000|     100|0.93586340|  PASSED
# The file file_input_raw was rewound 7 times
      rgb_lagged_sum|  14|   1000000|     100|0.23963436|  PASSED
# The file file_input_raw was rewound 8 times
      rgb_lagged_sum|  15|   1000000|     100|0.01521374|  PASSED
# The file file_input_raw was rewound 9 times
      rgb_lagged_sum|  16|   1000000|     100|0.81491356|  PASSED
# The file file_input_raw was rewound 10 times
      rgb_lagged_sum|  17|   1000000|     100|0.27392971|  PASSED
# The file file_input_raw was rewound 11 times
      rgb_lagged_sum|  18|   1000000|     100|0.92324229|  PASSED
# The file file_input_raw was rewound 12 times
      rgb_lagged_sum|  19|   1000000|     100|0.94434495|  PASSED
# The file file_input_raw was rewound 13 times
      rgb_lagged_sum|  20|   1000000|     100|0.84405283|  PASSED
# The file file_input_raw was rewound 14 times
      rgb_lagged_sum|  21|   1000000|     100|0.71485832|  PASSED
# The file file_input_raw was rewound 15 times
      rgb_lagged_sum|  22|   1000000|     100|0.97576170|  PASSED
# The file file_input_raw was rewound 16 times
      rgb_lagged_sum|  23|   1000000|     100|0.89027432|  PASSED
# The file file_input_raw was rewound 17 times
      rgb_lagged_sum|  24|   1000000|     100|0.96400793|  PASSED
# The file file_input_raw was rewound 18 times
      rgb_lagged_sum|  25|   1000000|     100|0.01657126|  PASSED
# The file file_input_raw was rewound 19 times
      rgb_lagged_sum|  26|   1000000|     100|0.86189687|  PASSED
# The file file_input_raw was rewound 21 times
      rgb_lagged_sum|  27|   1000000|     100|0.29543326|  PASSED
# The file file_input_raw was rewound 22 times
      rgb_lagged_sum|  28|   1000000|     100|0.23139324|  PASSED
# The file file_input_raw was rewound 24 times
      rgb_lagged_sum|  29|   1000000|     100|0.70494232|  PASSED
# The file file_input_raw was rewound 25 times
      rgb_lagged_sum|  30|   1000000|     100|0.62063235|  PASSED
# The file file_input_raw was rewound 26 times
      rgb_lagged_sum|  31|   1000000|     100|0.06966279|  PASSED
# The file file_input_raw was rewound 28 times
      rgb_lagged_sum|  32|   1000000|     100|0.65344817|  PASSED
# The file file_input_raw was rewound 28 times
     rgb_kstest_test|   0|     10000|    1000|0.09516346|  PASSED
# The file file_input_raw was rewound 28 times
     dab_bytedistrib|   0|  51200000|       1|0.13019999|  PASSED
# The file file_input_raw was rewound 28 times
             dab_dct| 256|     50000|       1|0.61967499|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 28 times
        dab_filltree|  32|  15000000|       1|0.55898098|  PASSED
        dab_filltree|  32|  15000000|       1|0.15182873|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 28 times
       dab_filltree2|   0|   5000000|       1|0.68270035|  PASSED
       dab_filltree2|   1|   5000000|       1|0.52125911|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 28 times
        dab_monobit2|  12|  65000000|       1|0.99999908|  FAILED
```

ANNEX XVII - VIKTORIA DIFFERENTIAL TEST

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                      arqvik.bin|  1.78e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|       100|0.80980066|  PASSED
      diehard_operm5|   0|   1000000|       100|0.00000000|  FAILED
  diehard_rank_32x32|   0|     40000|       100|0.00000000|  FAILED
    diehard_rank_6x8|   0|    100000|       100|0.00363075|   WEAK
   diehard_bitstream|   0|   2097152|       100|0.00000000|  FAILED
        diehard_opso|   0|   2097152|       100|0.00000000|  FAILED
        diehard_oqso|   0|   2097152|       100|0.00000000|  FAILED
         diehard_dna|   0|   2097152|       100|0.00000000|  FAILED
diehard_count_1s_str|   0|    256000|       100|0.00000000|  FAILED
diehard_count_1s_byt|   0|    256000|       100|0.00000000|  FAILED
 diehard_parking_lot|   0|     12000|       100|0.17780825|  PASSED
    diehard_2dsphere|   2|      8000|       100|0.03508219|  PASSED
    diehard_3dsphere|   3|      4000|       100|0.01840861|  PASSED
     diehard_squeeze|   0|    100000|       100|0.00000000|  FAILED
        diehard_sums|   0|       100|       100|0.00050691|   WEAK
        diehard_runs|   0|    100000|       100|0.72243425|  PASSED
        diehard_runs|   0|    100000|       100|0.17611909|  PASSED
       diehard_craps|   0|    200000|       100|0.46840702|  PASSED
       diehard_craps|   0|    200000|       100|0.93428956|  PASSED
# The file file_input_raw was rewound 1 times
 marsaglia_tsang_gcd|   0|  10000000|       100|0.00000000|  FAILED
 marsaglia_tsang_gcd|   0|  10000000|       100|0.00000000|  FAILED
# The file file_input_raw was rewound 1 times
         sts_monobit|   1|    100000|       100|0.77279498|  PASSED
# The file file_input_raw was rewound 1 times
            sts_runs|   2|    100000|       100|0.29323615|  PASSED
# The file file_input_raw was rewound 1 times
          sts_serial|   1|    100000|       100|0.27586996|  PASSED
          sts_serial|   2|    100000|       100|0.43267838|  PASSED
          sts_serial|   3|    100000|       100|0.12802595|  PASSED
          sts_serial|   3|    100000|       100|0.30441256|  PASSED
          sts_serial|   4|    100000|       100|0.18626376|  PASSED
          sts_serial|   4|    100000|       100|0.66530881|  PASSED
          sts_serial|   5|    100000|       100|0.04364729|  PASSED
          sts_serial|   5|    100000|       100|0.36235534|  PASSED
          sts_serial|   6|    100000|       100|0.39797676|  PASSED
          sts_serial|   6|    100000|       100|0.37756534|  PASSED
          sts_serial|   7|    100000|       100|0.66364463|  PASSED
          sts_serial|   7|    100000|       100|0.81376448|  PASSED
          sts_serial|   8|    100000|       100|0.74499546|  PASSED
          sts_serial|   8|    100000|       100|0.98587695|  PASSED
          sts_serial|   9|    100000|       100|0.57520828|  PASSED
          sts_serial|   9|    100000|       100|0.56442559|  PASSED
          sts_serial|  10|    100000|       100|0.85289409|  PASSED
          sts_serial|  10|    100000|       100|0.99743545|   WEAK
          sts_serial|  11|    100000|       100|0.42226109|  PASSED
          sts_serial|  11|    100000|       100|0.29877650|  PASSED
          sts_serial|  12|    100000|       100|0.24859530|  PASSED
          sts_serial|  12|    100000|       100|0.81034323|  PASSED
          sts_serial|  13|    100000|       100|0.99155406|  PASSED
          sts_serial|  13|    100000|       100|0.55976197|  PASSED
          sts_serial|  14|    100000|       100|0.16865694|  PASSED
          sts_serial|  14|    100000|       100|0.96660456|  PASSED
          sts_serial|  15|    100000|       100|0.33516389|  PASSED
          sts_serial|  15|    100000|       100|0.43480960|  PASSED
          sts_serial|  16|    100000|       100|0.51657567|  PASSED
          sts_serial|  16|    100000|       100|0.22794651|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   1|    100000|       100|0.48356407|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   2|    100000|       100|0.45336065|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   3|    100000|       100|0.21549248|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   4|    100000|       100|0.56386129|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   5|    100000|       100|0.78347674|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   6|    100000|       100|0.72171089|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   7|    100000|       100|0.28666364|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   8|    100000|       100|0.69816204|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   9|    100000|       100|0.94185004|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  10|    100000|       100|0.60541589|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  11|    100000|       100|0.81381574|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  12|    100000|       100|0.53311366|  PASSED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   2|     10000|      1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   3|     10000|      1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   4|     10000|      1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   5|     10000|      1000|0.00000000|  FAILED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
# The file file_input_raw was rewound 2 times
    rgb_permutations|   2|    100000|       100|0.73336523|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   3|    100000|       100|0.82606740|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   4|    100000|       100|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   5|    100000|       100|0.00337297|   WEAK
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   0|   1000000|       100|0.64322569|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   1|   1000000|       100|0.02826583|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   2|   1000000|       100|0.90219007|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   3|   1000000|       100|0.25457285|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   4|   1000000|       100|0.18789036|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   5|   1000000|       100|0.00082239|   WEAK
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   6|   1000000|       100|0.44079496|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   7|   1000000|       100|0.45587344|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   8|   1000000|       100|0.00005248|   WEAK
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   9|   1000000|       100|0.98194500|  PASSED
# The file file_input_raw was rewound 5 times
      rgb_lagged_sum|  10|   1000000|       100|0.11951291|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  11|   1000000|       100|0.47324090|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  12|   1000000|       100|0.44851596|  PASSED
# The file file_input_raw was rewound 7 times
      rgb_lagged_sum|  13|   1000000|       100|0.32777018|  PASSED
# The file file_input_raw was rewound 7 times
      rgb_lagged_sum|  14|   1000000|       100|0.19408335|  PASSED
# The file file_input_raw was rewound 8 times
      rgb_lagged_sum|  15|   1000000|       100|0.75708162|  PASSED
# The file file_input_raw was rewound 9 times
      rgb_lagged_sum|  16|   1000000|       100|0.12558499|  PASSED
# The file file_input_raw was rewound 10 times
      rgb_lagged_sum|  17|   1000000|       100|0.55439223|  PASSED
# The file file_input_raw was rewound 11 times
      rgb_lagged_sum|  18|   1000000|       100|0.60240772|  PASSED
# The file file_input_raw was rewound 12 times
      rgb_lagged_sum|  19|   1000000|       100|0.38380537|  PASSED
# The file file_input_raw was rewound 13 times
      rgb_lagged_sum|  20|   1000000|       100|0.27915302|  PASSED
# The file file_input_raw was rewound 14 times
      rgb_lagged_sum|  21|   1000000|       100|0.22112737|  PASSED
# The file file_input_raw was rewound 15 times
      rgb_lagged_sum|  22|   1000000|       100|0.04745693|  PASSED
# The file file_input_raw was rewound 16 times
      rgb_lagged_sum|  23|   1000000|       100|0.27223974|  PASSED
# The file file_input_raw was rewound 17 times
      rgb_lagged_sum|  24|   1000000|       100|0.11919993|  PASSED
# The file file_input_raw was rewound 18 times
      rgb_lagged_sum|  25|   1000000|       100|0.22019615|  PASSED
# The file file_input_raw was rewound 19 times
      rgb_lagged_sum|  26|   1000000|       100|0.26953135|  PASSED
# The file file_input_raw was rewound 21 times
      rgb_lagged_sum|  27|   1000000|       100|0.19221811|  PASSED
# The file file_input_raw was rewound 22 times
      rgb_lagged_sum|  28|   1000000|       100|0.01594772|  PASSED
# The file file_input_raw was rewound 24 times
      rgb_lagged_sum|  29|   1000000|       100|0.16044572|  PASSED
# The file file_input_raw was rewound 25 times
      rgb_lagged_sum|  30|   1000000|       100|0.06566754|  PASSED
# The file file_input_raw was rewound 26 times
      rgb_lagged_sum|  31|   1000000|       100|0.77148753|  PASSED
# The file file_input_raw was rewound 28 times
      rgb_lagged_sum|  32|   1000000|       100|0.01000619|  PASSED
# The file file_input_raw was rewound 28 times
     rgb_kstest_test|   0|     10000|      1000|0.14503882|  PASSED
# The file file_input_raw was rewound 28 times
     dab_bytedistrib|   0|  51200000|         1|0.30909317|  PASSED
# The file file_input_raw was rewound 28 times
             dab_dct| 256|     50000|         1|0.27828998|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 28 times
        dab_filltree|  32|  15000000|         1|0.92962929|  PASSED
        dab_filltree|  32|  15000000|         1|0.74063832|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 28 times
       dab_filltree2|   0|   5000000|         1|0.89512611|  PASSED
       dab_filltree2|   1|   5000000|         1|0.90512783|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 28 times
        dab_monobit2|  12|  65000000|         1|1.00000000|  FAILED
```

ANNEX XVIII - DIFFERENTIAL TEST SHA3-512

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                     arqsha3.bin|  1.87e+07  |
#=============================================================================#
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
   diehard_birthdays|   0|       100|     100|0.85746119|  PASSED
      diehard_operm5|   0|   1000000|     100|0.00000000|  FAILED
  diehard_rank_32x32|   0|     40000|     100|0.00000000|  FAILED
    diehard_rank_6x8|   0|    100000|     100|0.02732275|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.00000000|  FAILED
        diehard_opso|   0|   2097152|     100|0.00000000|  FAILED
        diehard_oqso|   0|   2097152|     100|0.00000000|  FAILED
         diehard_dna|   0|   2097152|     100|0.00000000|  FAILED
diehard_count_1s_str|   0|    256000|     100|0.00000000|  FAILED
diehard_count_1s_byt|   0|    256000|     100|0.00000000|  FAILED
 diehard_parking_lot|   0|     12000|     100|0.02074842|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.00010112|   WEAK
    diehard_3dsphere|   3|      4000|     100|0.00168023|   WEAK
     diehard_squeeze|   0|    100000|     100|0.00000000|  FAILED
        diehard_sums|   0|       100|     100|0.09233500|  PASSED
        diehard_runs|   0|    100000|     100|0.83190568|  PASSED
        diehard_runs|   0|    100000|     100|0.08879769|  PASSED
       diehard_craps|   0|    200000|     100|0.47109393|  PASSED
       diehard_craps|   0|    200000|     100|0.67282628|  PASSED
# The file file_input_raw was rewound 1 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000000|  FAILED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 1 times
         sts_monobit|   1|    100000|     100|0.47947866|  PASSED
# The file file_input_raw was rewound 1 times
            sts_runs|   2|    100000|     100|0.59834489|  PASSED
# The file file_input_raw was rewound 1 times
          sts_serial|   1|    100000|     100|0.99600329|   WEAK
          sts_serial|   2|    100000|     100|0.99626941|   WEAK
          sts_serial|   3|    100000|     100|0.08844238|  PASSED
          sts_serial|   3|    100000|     100|0.06912171|  PASSED
          sts_serial|   4|    100000|     100|0.25111779|  PASSED
          sts_serial|   4|    100000|     100|0.66934447|  PASSED
          sts_serial|   5|    100000|     100|0.95435618|  PASSED
          sts_serial|   5|    100000|     100|0.44517987|  PASSED
          sts_serial|   6|    100000|     100|0.99770952|   WEAK
          sts_serial|   6|    100000|     100|0.60021332|  PASSED
          sts_serial|   7|    100000|     100|0.93788950|  PASSED
          sts_serial|   7|    100000|     100|0.91515956|  PASSED
          sts_serial|   8|    100000|     100|0.48873432|  PASSED
          sts_serial|   8|    100000|     100|0.38278118|  PASSED
          sts_serial|   9|    100000|     100|0.80329833|  PASSED
          sts_serial|   9|    100000|     100|0.09724849|  PASSED
          sts_serial|  10|    100000|     100|0.60477920|  PASSED
          sts_serial|  10|    100000|     100|0.52611455|  PASSED
          sts_serial|  11|    100000|     100|0.48819362|  PASSED
          sts_serial|  11|    100000|     100|0.54994263|  PASSED
          sts_serial|  12|    100000|     100|0.39185982|  PASSED
          sts_serial|  12|    100000|     100|0.07340076|  PASSED
          sts_serial|  13|    100000|     100|0.28369542|  PASSED
          sts_serial|  13|    100000|     100|0.31511192|  PASSED
          sts_serial|  14|    100000|     100|0.52998426|  PASSED
          sts_serial|  14|    100000|     100|0.99998190|   WEAK
          sts_serial|  15|    100000|     100|0.66152327|  PASSED
          sts_serial|  15|    100000|     100|0.72253637|  PASSED
          sts_serial|  16|    100000|     100|0.13170012|  PASSED
          sts_serial|  16|    100000|     100|0.36791337|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   1|    100000|     100|0.13857457|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   2|    100000|     100|0.04267345|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   3|    100000|     100|0.39583047|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   4|    100000|     100|0.89814118|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   5|    100000|     100|0.74399166|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   6|    100000|     100|0.71322623|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   7|    100000|     100|0.42302340|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   8|    100000|     100|0.67987675|  PASSED
# The file file_input_raw was rewound 1 times
         rgb_bitdist|   9|    100000|     100|0.29154581|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  10|    100000|     100|0.96820965|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  11|    100000|     100|0.22675024|  PASSED
# The file file_input_raw was rewound 2 times
         rgb_bitdist|  12|    100000|     100|0.99142450|  PASSED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   2|     10000|    1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   3|     10000|    1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   4|     10000|    1000|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
rgb_minimum_distance|   5|     10000|    1000|0.00000000|  FAILED
```

```
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
# The file file_input_raw was rewound 2 times
    rgb_permutations|   2|    100000|     100|0.14353727|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   3|    100000|     100|0.65755030|  PASSED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   4|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 2 times
    rgb_permutations|   5|    100000|     100|0.04521471|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   0|   1000000|     100|0.94743507|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   1|   1000000|     100|0.25078163|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   2|   1000000|     100|0.77983604|  PASSED
# The file file_input_raw was rewound 2 times
      rgb_lagged_sum|   3|   1000000|     100|0.31210749|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   4|   1000000|     100|0.50404799|  PASSED
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   5|   1000000|     100|0.00100928|   WEAK
# The file file_input_raw was rewound 3 times
      rgb_lagged_sum|   6|   1000000|     100|0.49202705|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   7|   1000000|     100|0.40097935|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   8|   1000000|     100|0.18850544|  PASSED
# The file file_input_raw was rewound 4 times
      rgb_lagged_sum|   9|   1000000|     100|0.24470035|  PASSED
# The file file_input_raw was rewound 5 times
      rgb_lagged_sum|  10|   1000000|     100|0.80766166|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  11|   1000000|     100|0.70218962|  PASSED
# The file file_input_raw was rewound 6 times
      rgb_lagged_sum|  12|   1000000|     100|0.62802213|  PASSED
# The file file_input_raw was rewound 7 times
      rgb_lagged_sum|  13|   1000000|     100|0.11618923|  PASSED
# The file file_input_raw was rewound 7 times
      rgb_lagged_sum|  14|   1000000|     100|0.29009521|  PASSED
# The file file_input_raw was rewound 8 times
      rgb_lagged_sum|  15|   1000000|     100|0.15187554|  PASSED
# The file file_input_raw was rewound 9 times
      rgb_lagged_sum|  16|   1000000|     100|0.65636309|  PASSED
# The file file_input_raw was rewound 10 times
      rgb_lagged_sum|  17|   1000000|     100|0.73990692|  PASSED
# The file file_input_raw was rewound 11 times
      rgb_lagged_sum|  18|   1000000|     100|0.92891924|  PASSED
# The file file_input_raw was rewound 12 times
      rgb_lagged_sum|  19|   1000000|     100|0.34723467|  PASSED
# The file file_input_raw was rewound 13 times
      rgb_lagged_sum|  20|   1000000|     100|0.19312594|  PASSED
# The file file_input_raw was rewound 14 times
      rgb_lagged_sum|  21|   1000000|     100|0.46253647|  PASSED
# The file file_input_raw was rewound 15 times
      rgb_lagged_sum|  22|   1000000|     100|0.81470019|  PASSED
# The file file_input_raw was rewound 16 times
      rgb_lagged_sum|  23|   1000000|     100|0.62181440|  PASSED
# The file file_input_raw was rewound 17 times
      rgb_lagged_sum|  24|   1000000|     100|0.22702952|  PASSED
# The file file_input_raw was rewound 18 times
      rgb_lagged_sum|  25|   1000000|     100|0.90461570|  PASSED
# The file file_input_raw was rewound 19 times
      rgb_lagged_sum|  26|   1000000|     100|0.89642948|  PASSED
# The file file_input_raw was rewound 21 times
      rgb_lagged_sum|  27|   1000000|     100|0.20100598|  PASSED
# The file file_input_raw was rewound 22 times
      rgb_lagged_sum|  28|   1000000|     100|0.08745666|  PASSED
# The file file_input_raw was rewound 24 times
      rgb_lagged_sum|  29|   1000000|     100|0.63090483|  PASSED
# The file file_input_raw was rewound 25 times
      rgb_lagged_sum|  30|   1000000|     100|0.10863050|  PASSED
# The file file_input_raw was rewound 26 times
      rgb_lagged_sum|  31|   1000000|     100|0.05091289|  PASSED
# The file file_input_raw was rewound 28 times
      rgb_lagged_sum|  32|   1000000|     100|0.53175963|  PASSED
# The file file_input_raw was rewound 28 times
     rgb_kstest_test|   0|     10000|    1000|0.20340918|  PASSED
# The file file_input_raw was rewound 28 times
     dab_bytedistrib|   0|  51200000|       1|0.05403366|  PASSED
# The file file_input_raw was rewound 28 times
             dab_dct| 256|     50000|       1|0.54004870|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 28 times
        dab_filltree|  32|  15000000|       1|0.39003339|  PASSED
        dab_filltree|  32|  15000000|       1|0.24025124|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 28 times
       dab_filltree2|   0|   5000000|       1|0.69927836|  PASSED
       dab_filltree2|   1|   5000000|       1|0.44656637|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 28 times
        dab_monobit2|  12|  65000000|       1|0.97677955|  PASSED
```

ANNEX XIX - SOURCE-COMPLETE CODE OF HASH VIKTORIA FUNCTION

```
/*-------------------------------------------------------------------------------------------------
                                        VIKTORIA++ HASH
---------------------------------------------------------------------------------------------------
Designer and developer..: Edimar Veríssimo
Last modified...........: 22/02/2020
---------------------------------------------------------------------------------------------------
SOURCE CODE COMPILED WITH:
    gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
    Copyright (C) 2017 Free Software Foundation, Inc.
    This is free software; see the source for copying conditions.  There is NO
    warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
---------------------------------------------------------------------------------------------------
This work is dedicated exclusively in memory of Viktoria Tkotz.
---------------------------------------------------------------------------------------------------*/

#include <ctype.h>
#include <sys/time.h>
#include <fcntl.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

// DECLARATION OF GLOBAL VARIABLES:
FILE *p1;
unsigned char BLOCK[64], BLOCK_TMP[64];  // Processing blocks
unsigned long long int tamanho;             // 64-bit unsigned variable representing file size
unsigned char PERMUTACAO[7920][4];       // Switching prime numbers {2, 3, ..., 31} to rotation
unsigned int BINARIO[32];                // Binary vector
unsigned char T1[256] =
{204,  193,   96,   10,  100,  208,  104,  212,  109,   52,   70,   95,  108,   99,  103,   11,
 107,   98,  102,  106,  118,   22,  122,  111,  130,    1,  154,  162,  166,  115,  186,  198,
 119,  238,  250,  123,   30,  127,  216,  131,   61,  135,   14,  139,  143,  147,  151,  112,
 220,   54,  155,   57,  159,   60,  163,  167,   63,   17,  171,   69,  205,  175,    7,  179,
  75,  183,  187,  116,  191,   97,  165,    2,  195,   20,   90,  199,   88,  203,  207,  211,
 133,  215,  225,  224,   43,  219,   79,  223,  120,  227,   23,  231,  235,  153,  185,  239,
   0,  243,  247,  251,  228,   26,  255,  124,   29,  232,  128,  121,  141,   32,   13,  114,
 217,   12,   34,  142,   18,  190,  132,   33,  236,   48,   35,  240,  249,  136,   38,   72,
  84,  244,  237,   25,   41,  177,    4,  248,  140,   44,   64,   73,  144,   47,  252,  148,
 101,   50,  197,   46,  152,   53,  113,  145,   82,   91,  156,   56,   16,   55,  160,  157,
  37,   59,  221,  164,    6,   62,  169,  209,   65,  168,  229,   68,   28,  172,    9,  110,
 189,  241,  134,  158,  170,  178,  182,  194,   21,  202,  206,  218,  226,  234,  242,  254,
  27,   71,  253,  176,   67,   76,    5,   74,  125,  180,   87,   49,   77,   93,  184,  137,
  19,   85,   80,  181,    8,   83,  188,  105,  149,   58,   86,  192,  213,   40,  126,  138,
 146,  150,   15,  174,   94,  210,  214,  222,  230,   24,  246,  117,    3,  201,  233,  245,
  31,  196,   36,   89,   39,   42,   45,   51,   66,  129,  161,   92,   78,   81,  200,  173 };
unsigned char T2[256] =
{240,   49,  145,  148,   52,  244,  152,  193,   56,  248,   41,  229,  241,  156,  137,  157,
 165,  252,   60,    6,   14,   50,   66,   21,   74,   77,  114,  118,  160,  222,  226,  234,
 242,  250,   97,  109,   64,  164,    9,   69,  149,  185,  213,   68,  168,  129,    3,   72,
   7,  172,   11,   15,   19,   23,  177,   27,   31,   35,   39,   43,   47,  176,   51,   55,
  76,   59,  141,   63,   67,   71,    2,   10,   18,   26,   46,   75,   80,   62,   33,   79,
  89,  121,  106,   83,  110,  126,  130,  142,  146,   87,  170,  174,  182,  186,  190,  194,
  91,  206,  210,  214,  218,   95,  254,   99,  103,  180,  107,  111,  115,   84,  119,   61,
 123,  127,  131,  135,  139,    1,  205,  143,  147,  151,  155,  184,   53,   88,  159,  101,
 169,  163,  167,  171,  175,   81,  179,  233,  183,  187,  191,  188,  195,  199,  203,   92,
 207,  211,  215,  197,  253,  219,  223,  227,   25,  231,  192,  235,  239,  243,  247,   96,
 251,  245,  255,  196,    0,  161,  100,    4,  104,   45,  189,  200,   73,  113,  217,   37,
 108,  225,    8,   13,  133,  181,  209,  204,  112,  208,   12,   93,   16,  116,  212,   20,
 173,  120,   30,   34,   42,   85,   82,  153,  237,   98,  102,  134,  138,  150,   24,  158,
 162,  166,  178,  202,  238,  216,  124,   28,    5,  220,  125,  105,   32,  128,  224,   57,
 132,   36,   65,   17,   40,  228,  136,   29,  201,   22,   38,  140,   54,   58,   70,   78,
  86,   90,   94,  232,  122,  154,  198,  230,  246,   44,  236,  117,  144,  221,  249,   48 };

// DECLARATION OF FUNCTIONS:
unsigned long long int verify_size();                                      // See the size of a file
void read_block();                                                             // Reads 512 bits of the file
void rotate_block(unsigned int palavras[]);                           // Rotates a 128-bit set of a complete block.
void permutation_block(register unsigned char tipo);                                // Switches in a 64-byte group
void mixword();                                                            // 512-bit block mixing function
// Main Body of Code:
void processing(char nome_arquivo[], char numbits_string[], char valuebits_string[]);
void start_maps(char nome_arquivo[]);                                      // Initiates the map tables according to the file
// Creates the header containing information about the size of the file:
void header_archive(unsigned long long int tamanho_arquivo, unsigned char numbits, unsigned char valuebits);
void control_bytes_null(unsigned long long int tamanho_arquivo);      // Control for file size that is not a multiple of 64 bytes
void reset_maps();                                                             // Initializes swap tables T1 and T2
void finalizes();                                                          // Latest Processing Routine
void mixword_final();                                                      // mixword() function with more security features
void calculate_permutations();                                             // Calculates 7920 number combinations
unsigned int* permutation_binary_128(unsigned int palavras[], unsigned int* wpalavra ); // 128-bit binary exchange
void permutation_binary_512();                                         // 512-bit binary exchange
// Rotates a 128-bit set of a complete block.
void rotate_block2(register unsigned int p0, register unsigned int p1, register unsigned int p2, register unsigned int p3);

// PSEUDO-FUNCTION DEFINITIONS:
#define TRAN32(x,y,w,z)    ( (x << 24) ^ (y << 16) ^ (w << 8) ^ z  )
#define TRAN32B(x1,x2,y1,y2,w1,w2,z1,z2)    ( ((x1^x2) << 24) ^ ((y1^y2) << 16) ^ ((w1^w2) << 8) ^ z1  ^ z2 )
#define TRAN32M1(x,y,w,z)  ( (T1[x] << 24)  ^ (T1[y] << 16) ^ (T1[w] << 8) ^ T1[z]  )
#define TRAN32M2(x,y,w,z)  ( (T2[x] << 24)  ^ (T2[y] << 16) ^ (T2[w] << 8) ^ T2[z]  )
#define ROTL32(x, y) (((x) << (y)) | ((x) >> (32 - (y))))    // Optimized rotation routine

/*-------------------------------------------------------------------------------------------------
MAIN BODY
---------------------------------------------------------------------------------------------------*/
void main(int argc, char *argv[ ] ) {
    register unsigned int ct, size_hash,ct2,control_perm=0;

    // Call for file processing:
    processing(argv[1], argv[3], argv[4]);  // The argument is the file name  and then non-binary message controls numbits + valuebits!!!

    // hash control greater than 512 bits -> 1 = 512, 2 = 1024, 3 = 1536, 4 = 2048, etc..
    if (argv[2] == NULL){
        size_hash = 1;
    } else {
        size_hash = atoi(argv[2]);
    }

    // Routine to present hash on file
    printf("\n");
    for(ct=0;ct<64;ct++){
         if(BLOCK[ct]>=16){
             printf("%x",BLOCK[ct]);
         } else {
             printf("0%x",BLOCK[ct]);
         }
    }

    // Control for hash greater than 512 bits, generating larger sized hashes
        for (ct=1;ct<size_hash;ct++){
            permutation_binary_512();
            mixword_final();
             permutation_block(control_perm);
            mixword_final();
            finalizes();

             control_perm++;
             if(control_perm > 7){
                 control_perm = 0;
             }

            for(ct2=0;ct2<64;ct2++){
                 if(BLOCK[ct2]>=16){
                     printf("%x",BLOCK[ct2]);
                 } else {
                     printf("0%x",BLOCK[ct2]);
                 }
             }

        }

    printf("\n");
}

/*--------------------------------------------------------------------------------------------------
FILE PROCESSING FUNCTION
---------------------------------------------------------------------------------------------------*/
void processing(char nome_arquivo[], char numbits_string[], char valuebits_string[]){
    register unsigned long long int ct;
    register unsigned char control_perm = 0;
    register unsigned int ct2;
    unsigned int size_hash = 1;
    unsigned char numbits, valuebits;

    // Control to check non-binary messages (parameters 3 and 4)
    // Parameter 3: quantities of bits to include in the file:
```

```
if (numbits_string == NULL){
    numbits = 0;
} else {
    numbits = atoi(numbits_string);
}

// Parameter 4: Byte that indicates the value of the included bits:
if (valuebits_string == NULL){
    valuebits = 0;
} else {
    valuebits = atoi(valuebits_string);
}

// Starting the file reading block and the temporary block
for(ct=0;ct<64;ct++){
     BLOCK[ct]=0;
     BLOCK_TMP[ct]=0;
}

// Calculating auxiliary functions
calculate_permutations();
reset_maps();
start_maps(nome_arquivo);

// Calculating the required powers of 2
BINARIO[0] = 1;
for (ct=1;ct<32;ct++){
     BINARIO[ct] = BINARIO[ct-1] * 2;
}

// Opening the file
if( (p1=fopen(nome_arquivo,"rb"))==NULL ) {  // always use "rb" to open file
     printf("\nThe file cannot be opened!\n");
     exit(1);
}

// Reading the file size:
tamanho = verify_size();

// Processing the header_archive with file size information.
header_archive(tamanho, numbits, valuebits);

// Null byte control for a non-multiple size file of 64
control_bytes_null(tamanho);

control_perm = 0; // Auxiliary variable to control byte exchange

for(ct=0;ct<tamanho;ct=ct+64){

     read_block();
     mixword();
     permutation_block(control_perm);
```

```
        control_perm++;
        if(control_perm > 7){
            control_perm = 0;
        }

    }

    mixword_final();
    finalizes();

    fflush(p1);
    fclose(p1);

}

/*-------------------------------------------------------------------------------------------------
NULL BYTE CONTROL TO COMPLETE FILE SIZE THAT IS NOT A MULTIPLE OF 64
---------------------------------------------------------------------------------------------------*/
void control_bytes_null(unsigned long long int tamanho_arquivo){
    register unsigned int ct;
    register unsigned char quant_bytes;
    unsigned char read_block;

    for(ct=0;ct<64;ct++){
        BLOCK_TMP[ct] = 0;
    }

    if (tamanho_arquivo % 64 != 0){
        quant_bytes = (tamanho_arquivo % 64);
        tamanho = tamanho - quant_bytes;   // Recalculating file size

        for(ct=0;ct<quant_bytes;ct++){
            fread(&read_block,sizeof(read_block),1,p1);
            BLOCK_TMP[ct]=T1[read_block];
        }
    }

    BLOCK_TMP[63] = (64 - (tamanho_arquivo % 64)) % 64;  // Number of null bytes considered

    for(ct=0;ct<64;ct++){
        BLOCK[ct] = BLOCK[ct] ^ BLOCK_TMP[ct];  // XOR with data from previous BLOCK
    }

    // Doing the block processing (processing in 16 times)
    for(ct=0;ct<16;ct++){
    mixword();
        permutation_binary_512();
    }

}

/*-------------------------------------------------------------------------------------------------
FUNCTION TO CREATE THE INITIAL CONTROL BLOCK WITH INFORMATION ABOUT THE FILE SIZE
```

```
----------------------------------------------------------------------------------------------------*/
void header_archive(unsigned long long int tamanho_arquivo, unsigned char numbits, unsigned char valuebits){
    register unsigned int ct;
    unsigned long long int potencia[8],resultado[8];
    register unsigned int posic;

        for(ct=0;ct<8;ct++){
            resultado[ct]=0;
        }

        // Powers to manage file size:
        potencia[0] = 1;
        potencia[1] = 256;
        potencia[2] = 65536;
        potencia[3] = 16777216;
        potencia[4] = 4294967296;
        potencia[5] = pow(256,5);
        potencia[6] = pow(256,6);
        potencia[7] = pow(256,7);

        // File header management:
        BLOCK[0] = 255;                  // Fixed byte
        BLOCK[1] = tamanho_arquivo % 64;  // File size in MOD 64 bytes
        BLOCK[2] = numbits;               // Amount of surplus bits (0 to 7)
        BLOCK[3] = valuebits;             // Byte (7 bits) representing the surplus bits (final bits are reset to the left)

        // Area for file size (Maximum 2^480 bytes)
        for(ct=4;ct<56;ct++){
            BLOCK[ct] = 0;
        }

        // Turning the size into a grade 7 polymer (in a real implementation should take into account larger file sizes):
        posic=7;
        for(;;){
            if (tamanho_arquivo >= potencia[posic]){;
                resultado[posic]++;
                tamanho_arquivo = tamanho_arquivo - potencia[posic];
            } else {
                --posic;
            }
            if (tamanho_arquivo==0){
                break;
            }
        }

        // Placing the information in the block
        posic=7;
        for(ct=56;ct<64;ct++){
            BLOCK[ct]=resultado[posic];
            --posic;
        }

        // Doing the processing of the header block (Switches in 16 times)
        for(ct=0;ct<16;ct++){
        mixword();
            permutation_binary_512();
        }
}

/*--------------------------------------------------------------------------------------------------
512-BIT BLOCK MIXING FUNCTION (main processing function)
----------------------------------------------------------------------------------------------------*/
void mixword(){
    register unsigned char round=0, ct;
    register unsigned int tmp;
    register unsigned int p0, p1, p2, p3;

    for(round=0;round<16;round++){

        if (round % 4 == 0) {
            p0 = (TRAN32B(BLOCK[0],BLOCK[16], BLOCK[4],BLOCK[21], BLOCK[8],BLOCK[26] , BLOCK[12],BLOCK[31]) + TRAN32M1(BLOCK[32], BLOCK[38],
BLOCK[40],BLOCK[46]))^ TRAN32M2(BLOCK[48], BLOCK[55], BLOCK[58], BLOCK[61]);
            p1 = (TRAN32B(BLOCK[1],BLOCK[17], BLOCK[5],BLOCK[22], BLOCK[9],BLOCK[27] , BLOCK[13],BLOCK[28]) + TRAN32M1(BLOCK[33], BLOCK[39],
BLOCK[41],BLOCK[47]))^ TRAN32M2(BLOCK[49], BLOCK[52], BLOCK[59], BLOCK[62]);
            p2 = (TRAN32B(BLOCK[2],BLOCK[18], BLOCK[6],BLOCK[23], BLOCK[10],BLOCK[24], BLOCK[14],BLOCK[29]) + TRAN32M1(BLOCK[34], BLOCK[36],
BLOCK[42],BLOCK[44]))^ TRAN32M2(BLOCK[50], BLOCK[53], BLOCK[56], BLOCK[63]);
            p3 = (TRAN32B(BLOCK[3],BLOCK[19], BLOCK[7],BLOCK[20], BLOCK[11],BLOCK[25], BLOCK[15],BLOCK[30]) + TRAN32M1(BLOCK[35], BLOCK[37],
BLOCK[43],BLOCK[45]))^ TRAN32M2(BLOCK[51], BLOCK[54], BLOCK[57], BLOCK[60]);
        } else if (round % 4 == 1) {
            p0 = (TRAN32B(BLOCK[4],BLOCK[16], BLOCK[8],BLOCK[21], BLOCK[12],BLOCK[26] , BLOCK[0],BLOCK[31]) + TRAN32M1(BLOCK[32], BLOCK[38],
BLOCK[40],BLOCK[46]))^ TRAN32M2(BLOCK[48], BLOCK[55], BLOCK[58], BLOCK[61]);
            p1 = (TRAN32B(BLOCK[5],BLOCK[17], BLOCK[9],BLOCK[22], BLOCK[13],BLOCK[27] , BLOCK[1],BLOCK[28])+ TRAN32M1(BLOCK[33], BLOCK[39],
BLOCK[41],BLOCK[47])) ^ TRAN32M2(BLOCK[49], BLOCK[52], BLOCK[59], BLOCK[62]);
            p2 = (TRAN32B(BLOCK[6],BLOCK[18], BLOCK[10],BLOCK[23], BLOCK[14],BLOCK[24], BLOCK[2],BLOCK[29]) + TRAN32M1(BLOCK[34], BLOCK[36],
BLOCK[42],BLOCK[44]))^ TRAN32M2(BLOCK[50], BLOCK[53], BLOCK[56], BLOCK[63]);
            p3 = (TRAN32B(BLOCK[7],BLOCK[19], BLOCK[11],BLOCK[20], BLOCK[15],BLOCK[25], BLOCK[3],BLOCK[30]) + TRAN32M1(BLOCK[35], BLOCK[37],
BLOCK[43],BLOCK[45]))^ TRAN32M2(BLOCK[51], BLOCK[54], BLOCK[57], BLOCK[60]);
        } else if (round % 4 == 2) {
            p0 = (TRAN32B(BLOCK[8],BLOCK[16], BLOCK[12],BLOCK[21] , BLOCK[0],BLOCK[26], BLOCK[4],BLOCK[31]) + TRAN32M1(BLOCK[32], BLOCK[38],
BLOCK[40],BLOCK[46]))^ TRAN32M2(BLOCK[48], BLOCK[55], BLOCK[58], BLOCK[61]);
            p1 = (TRAN32B(BLOCK[9],BLOCK[17], BLOCK[13],BLOCK[22] , BLOCK[1],BLOCK[27], BLOCK[5],BLOCK[28])+ TRAN32M1(BLOCK[33], BLOCK[39],
BLOCK[41],BLOCK[47])) ^ TRAN32M2(BLOCK[49], BLOCK[52], BLOCK[59], BLOCK[62]);
            p2 = (TRAN32B(BLOCK[10],BLOCK[18], BLOCK[14],BLOCK[23], BLOCK[2],BLOCK[24], BLOCK[6],BLOCK[29]) + TRAN32M1(BLOCK[34], BLOCK[36],
BLOCK[42],BLOCK[44]))^ TRAN32M2(BLOCK[50], BLOCK[53], BLOCK[56], BLOCK[63]);
            p3 = (TRAN32B(BLOCK[11],BLOCK[19], BLOCK[15],BLOCK[20], BLOCK[3],BLOCK[25], BLOCK[7],BLOCK[30])+ TRAN32M1(BLOCK[35], BLOCK[37],
BLOCK[43],BLOCK[45])) ^ TRAN32M2(BLOCK[51], BLOCK[54], BLOCK[57], BLOCK[60]);
        } else if (round % 4 == 3) {
            p0 = (TRAN32B(BLOCK[12],BLOCK[16] , BLOCK[0],BLOCK[21], BLOCK[4],BLOCK[26], BLOCK[8],BLOCK[31]) + TRAN32M1(BLOCK[32], BLOCK[38],
BLOCK[40],BLOCK[46]))^ TRAN32M2(BLOCK[48], BLOCK[55], BLOCK[58], BLOCK[61]);
            p1 = (TRAN32B(BLOCK[13],BLOCK[17] , BLOCK[1],BLOCK[22], BLOCK[5],BLOCK[27], BLOCK[9],BLOCK[28])+ TRAN32M1(BLOCK[33], BLOCK[39],
BLOCK[41],BLOCK[47])) ^ TRAN32M2(BLOCK[49], BLOCK[52], BLOCK[59], BLOCK[62]);
            p2 = (TRAN32B(BLOCK[14],BLOCK[18], BLOCK[2],BLOCK[23], BLOCK[6],BLOCK[24], BLOCK[10],BLOCK[29])+TRAN32M1(BLOCK[34], BLOCK[36],
BLOCK[42],BLOCK[44]))  ^ TRAN32M2(BLOCK[50], BLOCK[53], BLOCK[56], BLOCK[63]);
            p3 = (TRAN32B(BLOCK[15],BLOCK[19], BLOCK[3],BLOCK[20], BLOCK[7],BLOCK[25], BLOCK[11],BLOCK[30])+ TRAN32M1(BLOCK[35], BLOCK[37],
BLOCK[43],BLOCK[45])) ^ TRAN32M2(BLOCK[51], BLOCK[54], BLOCK[57], BLOCK[60]);
        }
```

```
// Modifying the T1 swap table
tmp = T1[ T2[ BLOCK[48] ] ];
T1[ T2[ BLOCK[48] ] ] = T1[ T2[ BLOCK[55] ] ];
T1[ T2[ BLOCK[55] ] ] = T1[ T2[ BLOCK[58] ] ];
T1[ T2[ BLOCK[58] ] ] = T1[ T2[ BLOCK[61] ] ];
T1[ T2[ BLOCK[61] ] ] = T1[ T2[ BLOCK[49] ] ];

T1[ T2[ BLOCK[49] ] ] = T1[ T2[ BLOCK[52] ] ];
T1[ T2[ BLOCK[52] ] ] = T1[ T2[ BLOCK[59] ] ];
T1[ T2[ BLOCK[59] ] ] = T1[ T2[ BLOCK[62] ] ];
T1[ T2[ BLOCK[62] ] ] = T1[ T2[ BLOCK[50] ] ];

T1[ T2[ BLOCK[50] ] ] = T1[ T2[ BLOCK[53] ] ];
T1[ T2[ BLOCK[53] ] ] = T1[ T2[ BLOCK[56] ] ];
T1[ T2[ BLOCK[56] ] ] = T1[ T2[ BLOCK[63] ] ];
T1[ T2[ BLOCK[63] ] ] = T1[ T2[ BLOCK[51] ] ];

T1[ T2[ BLOCK[51] ] ] = T1[ T2[ BLOCK[54] ] ];
T1[ T2[ BLOCK[54] ] ] = T1[ T2[ BLOCK[57] ] ];
T1[ T2[ BLOCK[57] ] ] = T1[ T2[ BLOCK[60] ] ];
T1[ T2[ BLOCK[60] ] ] = tmp;

// Modifying the T2 swap table
tmp = T2[ T1[ BLOCK[32] ] ];
T2[ T1[ BLOCK[32] ] ] = T2[ T1[ BLOCK[38] ] ];
T2[ T1[ BLOCK[38] ] ] = T2[ T1[ BLOCK[40] ] ];
T2[ T1[ BLOCK[40] ] ] = T2[ T1[ BLOCK[46] ] ];
T2[ T1[ BLOCK[46] ] ] = T2[ T1[ BLOCK[33] ] ];

T2[ T1[ BLOCK[33] ] ] = T2[ T1[ BLOCK[39] ] ];
T2[ T1[ BLOCK[39] ] ] = T2[ T1[ BLOCK[41] ] ];
T2[ T1[ BLOCK[41] ] ] = T2[ T1[ BLOCK[47] ] ];
T2[ T1[ BLOCK[47] ] ] = T2[ T1[ BLOCK[34] ] ];

T2[ T1[ BLOCK[34] ] ] = T2[ T1[ BLOCK[36] ] ];
T2[ T1[ BLOCK[36] ] ] = T2[ T1[ BLOCK[42] ] ];
T2[ T1[ BLOCK[42] ] ] = T2[ T1[ BLOCK[44] ] ];
T2[ T1[ BLOCK[44] ] ] = T2[ T1[ BLOCK[35] ] ];

T2[ T1[ BLOCK[35] ] ] = T2[ T1[ BLOCK[37] ] ];
T2[ T1[ BLOCK[37] ] ] = T2[ T1[ BLOCK[43] ] ];
T2[ T1[ BLOCK[43] ] ] = T2[ T1[ BLOCK[45] ] ];
T2[ T1[ BLOCK[45] ] ] = tmp;

// Diffusion of words
p0 ^= (ROTL32(~p1,13) ^ ROTL32(p2,3)) + ROTL32(~p3,27);
p1 += (ROTL32(p0,14) ^ ROTL32(~p2,11)) + ROTL32(p3,26);
p2 ^= (ROTL32(~p0,9) ^ ROTL32(p1,20)) + ROTL32(~p3,28);
p3 += (ROTL32(p0,17) ^ ROTL32(~p1,2)) + ROTL32(p2,1);

p0 ^= (ROTL32(~p1,25) ^ ROTL32(p2,7)) + ROTL32(~p3,18);
p1 += (ROTL32(p0,10) ^ ROTL32(~p2,8)) + ROTL32(p3,23);
```

```
        p2 ^= (ROTL32(~p0,15) ^ ROTL32(p1,31)) + ROTL32(~p3,29);
        p3 += (ROTL32(p0,30) ^ ROTL32(~p1,16)) + ROTL32(p2,21);

        p0 ^= (ROTL32(~p1,19) ^ ROTL32(p2,24)) + ROTL32(~p3,12);
        p1 += (ROTL32(p0,22) ^ ROTL32(~p2,4)) + ROTL32(p3,6);
        p2 ^= (ROTL32(~p0,5) ^ ROTL32(p1,8)) + ROTL32(~p3,13);
        p3 += (ROTL32(p0,14) ^ ROTL32(~p1,24)) + ROTL32(p2,20);

        // Rotating the subblocks
        rotate_block2(p0,p1,p2,p3);
    }
}

/*-------------------------------------------------------------------------------------------------
 READS A 512-BIT BLOCK FROM THE FILE BEING PROCESSED.
--------------------------------------------------------------------------------------------------*/
void read_block() {
    unsigned char read_block[64];
    register unsigned char ct;

    // Reading 64 bytes of the file
    fread(&read_block,sizeof(read_block),1,p1);

    // XOR with data from the previous block
    // We eliminate the FOR to gain processing speed
        BLOCK[ 0] ^= T1[read_block[ 0]];
        BLOCK[ 1] ^= T1[read_block[ 1]];
        BLOCK[ 2] ^= T1[read_block[ 2]];
        BLOCK[ 3] ^= T1[read_block[ 3]];
        BLOCK[ 4] ^= T1[read_block[ 4]];
        BLOCK[ 5] ^= T1[read_block[ 5]];
        BLOCK[ 6] ^= T1[read_block[ 6]];
        BLOCK[ 7] ^= T1[read_block[ 7]];
        BLOCK[ 8] ^= T1[read_block[ 8]];
        BLOCK[ 9] ^= T1[read_block[ 9]];
        BLOCK[10] ^= T1[read_block[10]];
        BLOCK[11] ^= T1[read_block[11]];
        BLOCK[12] ^= T1[read_block[12]];
        BLOCK[13] ^= T1[read_block[13]];
        BLOCK[14] ^= T1[read_block[14]];
        BLOCK[15] ^= T1[read_block[15]];

        BLOCK[16] ^= T1[read_block[16]];
        BLOCK[17] ^= T1[read_block[17]];
        BLOCK[18] ^= T1[read_block[18]];
        BLOCK[19] ^= T1[read_block[19]];
        BLOCK[20] ^= T1[read_block[20]];
        BLOCK[21] ^= T1[read_block[21]];
        BLOCK[22] ^= T1[read_block[22]];
        BLOCK[23] ^= T1[read_block[23]];
        BLOCK[24] ^= T1[read_block[24]];
        BLOCK[25] ^= T1[read_block[25]];
        BLOCK[26] ^= T1[read_block[26]];
```

```
         BLOCK[27] ^= T1[read_block[27]];
         BLOCK[28] ^= T1[read_block[28]];
         BLOCK[29] ^= T1[read_block[29]];
         BLOCK[30] ^= T1[read_block[30]];
         BLOCK[31] ^= T1[read_block[31]];

         BLOCK[32] ^= T1[read_block[32]];
         BLOCK[33] ^= T1[read_block[33]];
         BLOCK[34] ^= T1[read_block[34]];
         BLOCK[35] ^= T1[read_block[35]];
         BLOCK[36] ^= T1[read_block[36]];
         BLOCK[37] ^= T1[read_block[37]];
         BLOCK[38] ^= T1[read_block[38]];
         BLOCK[39] ^= T1[read_block[39]];
         BLOCK[40] ^= T1[read_block[40]];
         BLOCK[41] ^= T1[read_block[41]];
         BLOCK[42] ^= T1[read_block[42]];
         BLOCK[43] ^= T1[read_block[43]];
         BLOCK[44] ^= T1[read_block[44]];
         BLOCK[45] ^= T1[read_block[45]];
         BLOCK[46] ^= T1[read_block[46]];
         BLOCK[47] ^= T1[read_block[47]];

         BLOCK[48] ^= T1[read_block[48]];
         BLOCK[49] ^= T1[read_block[49]];
         BLOCK[50] ^= T1[read_block[50]];
         BLOCK[51] ^= T1[read_block[51]];
         BLOCK[52] ^= T1[read_block[52]];
         BLOCK[53] ^= T1[read_block[53]];
         BLOCK[54] ^= T1[read_block[54]];
         BLOCK[55] ^= T1[read_block[55]];
         BLOCK[56] ^= T1[read_block[56]];
         BLOCK[57] ^= T1[read_block[57]];
         BLOCK[58] ^= T1[read_block[58]];
         BLOCK[59] ^= T1[read_block[59]];
         BLOCK[60] ^= T1[read_block[60]];
         BLOCK[61] ^= T1[read_block[61]];
         BLOCK[62] ^= T1[read_block[62]];
         BLOCK[63] ^= T1[read_block[63]];
}

/*----------------------------------------------------------------------------------------------------
PERMUTATION OF THE 64 BYTES OF BLOCK
------------------------------------------------------------------------------------------------------*/
void permutation_block(register unsigned char tipo){
    register unsigned int ct;
    unsigned int posic;

    // Reordering the 64 bytes of block
    posic=0;

    switch(tipo) {
        case 0:
```

```
    for(ct=0;ct<256;ct++){
        if (T2[ct] < 64){
             BLOCK_TMP[posic] = BLOCK[T2[ct]];
             posic++;
            if (posic > 63){
                break;
            }
        }
    }
    break;

case 1:
    for(ct=0;ct<256;ct++){
        if (T2[ct] >= 64 & T2[ct] < 128){
             BLOCK_TMP[posic] = BLOCK[T2[ct]%64];
             posic++;
            if (posic > 63){
                break;
            }
        }
    }
    break;

case 2:
    for(ct=0;ct<256;ct++){
        if (T2[ct] >= 128 & T2[ct] < 192){
             BLOCK_TMP[posic] = BLOCK[T2[ct]%64];
             posic++;
            if (posic > 63){
                break;
            }
        }
    }
    break;

case 3:
    for(ct=0;ct<256;ct++){
        if (T2[ct] >= 192){
             BLOCK_TMP[posic] = BLOCK[T2[ct]%64];
             posic++;
            if (posic > 63){
                break;
            }
        }
    }
    break;


case 4:
    for(ct=0;ct<256;ct++){
        if (T1[ct] < 64){
             BLOCK_TMP[posic] = BLOCK[T1[ct]];
             posic++;
```

```
                if (posic > 63){
                    break;
                }
            }
        }
        break;

    case 5:
        for(ct=0;ct<256;ct++){
            if (T1[ct] >= 64 & T1[ct] < 128){
                 BLOCK_TMP[posic] = BLOCK[T1[ct]%64];
                 posic++;
                if (posic > 63){
                    break;
                }
            }
        }
        break;

    case 6:
        for(ct=0;ct<256;ct++){
            if (T1[ct] >= 128 & T1[ct] < 192){
                 BLOCK_TMP[posic] = BLOCK[T1[ct]%64];
                 posic++;
                if (posic > 63){
                    break;
                }
            }
        }
        break;

    case 7:
        for(ct=0;ct<256;ct++){
            if (T1[ct] >= 192){
                 BLOCK_TMP[posic] = BLOCK[T1[ct]%64];
                 posic++;
                if (posic > 63){
                    break;
                }
            }
        }
        break;
}

// Forming the new block
// We eliminate FOR to gain speed
     BLOCK[ 0] = BLOCK_TMP[ 0];
     BLOCK[ 1] = BLOCK_TMP[ 1];
     BLOCK[ 2] = BLOCK_TMP[ 2];
     BLOCK[ 3] = BLOCK_TMP[ 3];
     BLOCK[ 4] = BLOCK_TMP[ 4];
     BLOCK[ 5] = BLOCK_TMP[ 5];
     BLOCK[ 6] = BLOCK_TMP[ 6];
```

```
BLOCK[ 7] = BLOCK_TMP[ 7];
BLOCK[ 8] = BLOCK_TMP[ 8];
BLOCK[ 9] = BLOCK_TMP[ 9];
BLOCK[10] = BLOCK_TMP[10];
BLOCK[11] = BLOCK_TMP[11];
BLOCK[12] = BLOCK_TMP[12];
BLOCK[13] = BLOCK_TMP[13];
BLOCK[14] = BLOCK_TMP[14];
BLOCK[15] = BLOCK_TMP[15];
BLOCK[16] = BLOCK_TMP[16];
BLOCK[17] = BLOCK_TMP[17];
BLOCK[18] = BLOCK_TMP[18];
BLOCK[19] = BLOCK_TMP[19];
BLOCK[20] = BLOCK_TMP[20];
BLOCK[21] = BLOCK_TMP[21];
BLOCK[22] = BLOCK_TMP[22];
BLOCK[23] = BLOCK_TMP[23];
BLOCK[24] = BLOCK_TMP[24];
BLOCK[25] = BLOCK_TMP[25];
BLOCK[26] = BLOCK_TMP[26];
BLOCK[27] = BLOCK_TMP[27];
BLOCK[28] = BLOCK_TMP[28];
BLOCK[29] = BLOCK_TMP[29];
BLOCK[30] = BLOCK_TMP[30];
BLOCK[31] = BLOCK_TMP[31];
BLOCK[32] = BLOCK_TMP[32];
BLOCK[33] = BLOCK_TMP[33];
BLOCK[34] = BLOCK_TMP[34];
BLOCK[35] = BLOCK_TMP[35];
BLOCK[36] = BLOCK_TMP[36];
BLOCK[37] = BLOCK_TMP[37];
BLOCK[38] = BLOCK_TMP[38];
BLOCK[39] = BLOCK_TMP[39];
BLOCK[40] = BLOCK_TMP[40];
BLOCK[41] = BLOCK_TMP[41];
BLOCK[42] = BLOCK_TMP[42];
BLOCK[43] = BLOCK_TMP[43];
BLOCK[44] = BLOCK_TMP[44];
BLOCK[45] = BLOCK_TMP[45];
BLOCK[46] = BLOCK_TMP[46];
BLOCK[47] = BLOCK_TMP[47];
BLOCK[48] = BLOCK_TMP[48];
BLOCK[49] = BLOCK_TMP[49];
BLOCK[50] = BLOCK_TMP[50];
BLOCK[51] = BLOCK_TMP[51];
BLOCK[52] = BLOCK_TMP[52];
BLOCK[53] = BLOCK_TMP[53];
BLOCK[54] = BLOCK_TMP[54];
BLOCK[55] = BLOCK_TMP[55];
BLOCK[56] = BLOCK_TMP[56];
BLOCK[57] = BLOCK_TMP[57];
BLOCK[58] = BLOCK_TMP[58];
BLOCK[59] = BLOCK_TMP[59];
```

```
        BLOCK[60] = BLOCK_TMP[60];
        BLOCK[61] = BLOCK_TMP[61];
        BLOCK[62] = BLOCK_TMP[62];
        BLOCK[63] = BLOCK_TMP[63];
}

/*----------------------------------------------------------------------------------------------------
ROTATES 512 BLOCK IN 128-BIT SUBBLOCKS (mixword_FINAL)
----------------------------------------------------------------------------------------------------*/
void rotate_block(unsigned int palavras[]) {
    register unsigned int tmp;

      // We eliminate FOR to gain speed
        BLOCK[ 0] = BLOCK[16];
        BLOCK[ 1] = BLOCK[17];
        BLOCK[ 2] = BLOCK[18];
        BLOCK[ 3] = BLOCK[19];
        BLOCK[ 4] = BLOCK[20];
        BLOCK[ 5] = BLOCK[21];
        BLOCK[ 6] = BLOCK[22];
        BLOCK[ 7] = BLOCK[23];
        BLOCK[ 8] = BLOCK[24];
        BLOCK[ 9] = BLOCK[25];
        BLOCK[10] = BLOCK[26];
        BLOCK[11] = BLOCK[27];
        BLOCK[12] = BLOCK[28];
        BLOCK[13] = BLOCK[29];
        BLOCK[14] = BLOCK[30];
        BLOCK[15] = BLOCK[31];

        BLOCK[16] = BLOCK[32];
        BLOCK[17] = BLOCK[33];
        BLOCK[18] = BLOCK[34];
        BLOCK[19] = BLOCK[35];
        BLOCK[20] = BLOCK[36];
        BLOCK[21] = BLOCK[37];
        BLOCK[22] = BLOCK[38];
        BLOCK[23] = BLOCK[39];
        BLOCK[24] = BLOCK[40];
        BLOCK[25] = BLOCK[41];
        BLOCK[26] = BLOCK[42];
        BLOCK[27] = BLOCK[43];
        BLOCK[28] = BLOCK[44];
        BLOCK[29] = BLOCK[45];
        BLOCK[30] = BLOCK[46];
        BLOCK[31] = BLOCK[47];

        BLOCK[32] = BLOCK[48];
        BLOCK[33] = BLOCK[49];
        BLOCK[34] = BLOCK[50];
        BLOCK[35] = BLOCK[51];
        BLOCK[36] = BLOCK[52];
        BLOCK[37] = BLOCK[53];
```

```
         BLOCK[38] = BLOCK[54];
         BLOCK[39] = BLOCK[55];
         BLOCK[40] = BLOCK[56];
         BLOCK[41] = BLOCK[57];
         BLOCK[42] = BLOCK[58];
         BLOCK[43] = BLOCK[59];
         BLOCK[44] = BLOCK[60];
         BLOCK[45] = BLOCK[61];
         BLOCK[46] = BLOCK[62];
         BLOCK[47] = BLOCK[63];

        tmp = palavras[0];
         BLOCK[48] = T1[(unsigned char)(tmp >> 24)];
         BLOCK[49] = T1[(unsigned char)(((tmp >> 16) & 255) +1)];
         BLOCK[50] = T1[(unsigned char)(((tmp >>  8) & 255)+2)];
         BLOCK[51] = T1[(unsigned char)((tmp & 255)+3)];

        tmp = palavras[1];
         BLOCK[52] = T2[(unsigned char)((tmp >> 24)+4)];
         BLOCK[53] = T2[(unsigned char)(((tmp >> 16) & 255) +5)];
         BLOCK[54] = T2[(unsigned char)(((tmp >>  8) & 255)+6)];
         BLOCK[55] = T2[(unsigned char)((tmp & 255)+7)];

        tmp = palavras[2];
         BLOCK[56] = T1[(unsigned char)((tmp >> 24)+8)];
         BLOCK[57] = T1[(unsigned char)(((tmp >> 16) & 255) +9)];
         BLOCK[58] = T1[(unsigned char)(((tmp >>  8) & 255)+10)];
         BLOCK[59] = T1[(unsigned char)((tmp & 255)+11)];

        tmp = palavras[3];
         BLOCK[60] = T2[(unsigned char)((tmp >> 24)+12)];
         BLOCK[61] = T2[(unsigned char)(((tmp >> 16) & 255) +13)];
         BLOCK[62] = T2[(unsigned char)(((tmp >>  8) & 255)+14)];
         BLOCK[63] = T2[(unsigned char)((tmp & 255)+15)];
}

/*--------------------------------------------------------------------------------------------------
ROTACIONA BLOCK DE 512 EM SUB-BLOCKS DE 128 BITS
----------------------------------------------------------------------------------------------------*/
void rotate_block2(register unsigned int p0, register unsigned int p1, register unsigned int p2, register unsigned int p3){
      register unsigned int tmp;

      // We eliminate FOR to gain speed
         BLOCK[ 0] = BLOCK[16];
         BLOCK[ 1] = BLOCK[17];
         BLOCK[ 2] = BLOCK[18];
         BLOCK[ 3] = BLOCK[19];
         BLOCK[ 4] = BLOCK[20];
         BLOCK[ 5] = BLOCK[21];
         BLOCK[ 6] = BLOCK[22];
         BLOCK[ 7] = BLOCK[23];
         BLOCK[ 8] = BLOCK[24];
         BLOCK[ 9] = BLOCK[25];
```

```
BLOCK[10] = BLOCK[26];
BLOCK[11] = BLOCK[27];
BLOCK[12] = BLOCK[28];
BLOCK[13] = BLOCK[29];
BLOCK[14] = BLOCK[30];
BLOCK[15] = BLOCK[31];

BLOCK[16] = BLOCK[32];
BLOCK[17] = BLOCK[33];
BLOCK[18] = BLOCK[34];
BLOCK[19] = BLOCK[35];
BLOCK[20] = BLOCK[36];
BLOCK[21] = BLOCK[37];
BLOCK[22] = BLOCK[38];
BLOCK[23] = BLOCK[39];
BLOCK[24] = BLOCK[40];
BLOCK[25] = BLOCK[41];
BLOCK[26] = BLOCK[42];
BLOCK[27] = BLOCK[43];
BLOCK[28] = BLOCK[44];
BLOCK[29] = BLOCK[45];
BLOCK[30] = BLOCK[46];
BLOCK[31] = BLOCK[47];

BLOCK[32] = BLOCK[48];
BLOCK[33] = BLOCK[49];
BLOCK[34] = BLOCK[50];
BLOCK[35] = BLOCK[51];
BLOCK[36] = BLOCK[52];
BLOCK[37] = BLOCK[53];
BLOCK[38] = BLOCK[54];
BLOCK[39] = BLOCK[55];
BLOCK[40] = BLOCK[56];
BLOCK[41] = BLOCK[57];
BLOCK[42] = BLOCK[58];
BLOCK[43] = BLOCK[59];
BLOCK[44] = BLOCK[60];
BLOCK[45] = BLOCK[61];
BLOCK[46] = BLOCK[62];
BLOCK[47] = BLOCK[63];

BLOCK[48] = T1[(unsigned char)(p1 >> 24)];
BLOCK[49] = T1[(unsigned char)(((p1 >> 16) & 255) +1)];
BLOCK[50] = T1[(unsigned char)(((p1 >>  8) & 255)+2)];
BLOCK[51] = T1[(unsigned char)((p1 & 255)+3)];

BLOCK[52] = T2[(unsigned char)((p2 >> 24)+4)];
BLOCK[53] = T2[(unsigned char)(((p2 >> 16) & 255) +5)];
BLOCK[54] = T2[(unsigned char)(((p2 >>  8) & 255)+6)];
BLOCK[55] = T2[(unsigned char)((p2 & 255)+7)];

BLOCK[56] = T1[(unsigned char)((p3 >> 24)+8)];
BLOCK[57] = T1[(unsigned char)(((p3 >> 16) & 255) +9)];
```

```
        BLOCK[58] = T1[(unsigned char)(((p3 >>  8) & 255)+10)];
        BLOCK[59] = T1[(unsigned char)((p3 & 255)+11)];

        BLOCK[60] = T2[(unsigned char)((p0 >> 24)+12)];
        BLOCK[61] = T2[(unsigned char)(((p0 >> 16) & 255) +13)];
        BLOCK[62] = T2[(unsigned char)(((p0 >>  8) & 255)+14)];
        BLOCK[63] = T2[(unsigned char)((p0 & 255)+15)];

}

/*-------------------------------------------------------------------------------------------------
SAFETY ROUTINE TO SUPPLEMENT BLOCK EXCHANGE ON COMPLETION OF HASH ROUTINE
---------------------------------------------------------------------------------------------------*/
void finalizes(){
    register unsigned int tmp1, tmp2;
    register unsigned int resultado;
    register unsigned char ct, posicao;

    posicao=0;

    for(ct=0;ct<64;ct++){
        tmp1 = (T1[posicao] * 256) + T2[posicao];
        tmp2 = (T2[posicao+64] * 256) + T1[posicao+64];

        if (tmp1 == 0){
            tmp1 = 65536;
        }

        if (tmp2 == 0){
            tmp2 = 65536;
        }

        resultado = (tmp1 * tmp2) % 65537;

        BLOCK[ct]= BLOCK[ct] ^ (resultado % 256);
        posicao++;
    }
}

/*-------------------------------------------------------------------------------------------------
ROUTINE TO CHECK THE FILE SIZE
---------------------------------------------------------------------------------------------------*/
unsigned long long int verify_size() {
    unsigned long long int tamanho;

    // Posicionando o arquivo no seu inicio
    fseek (p1, 0, SEEK_SET);

    // Lendo o tamanho do arquivo:
    fseek (p1, 0, SEEK_END);
    tamanho = ftell (p1);

    // Posicionando o arquivo no seu inicio
    fseek (p1, 0, SEEK_SET);

    return(tamanho);
}

/*--------------------------------------------------------------------------------------------------
INITIALIZES THE MAP AND T2 VECTORS
This routine does with the initiation of the Pivot Maps tables
be 256! * 256! according to data from the file to be processed
----------------------------------------------------------------------------------------------------*/
void start_maps(char nome_arquivo[]){
    register unsigned long long int ct;
    register unsigned char controle, posic, tmp1;
    register unsigned int ct2;
    register unsigned int acumula, tmp2;
    unsigned char read_block;
    unsigned char read_block2[256];
    unsigned char troca, posicao;
    unsigned int residuo;

    // Opening the file
    if( (p1=fopen(nome_arquivo,"r"))==NULL ) {
        printf("\nFile cannot be opened!\n");
        exit(1);
    }

    // Reading the file size:
    tamanho = verify_size();

    posic = 0;
    acumula = 0;
    controle = 0;

    if (tamanho < 256) {
        // Processing the byte to byte file
        for(ct=0;ct<tamanho;ct++){
            fread(&read_block,sizeof(read_block),1,p1);

            if (posic == 0){
                troca = T2[read_block];
                tmp1 = T1[read_block];
                posicao = (troca + controle) % 256;
                T1[read_block] = T1[posicao];
                T1[posicao]=tmp1;
                posic = 1;
            } else {
                troca = T1[read_block];
                tmp1 = T2[read_block];
                posicao = (troca + controle) % 256;
                T2[read_block] = T2[posicao];
                T2[posicao]=tmp1;
                posic = 0;
            }
```

```
              controle = (controle + 1) % 256;
             acumula = (acumula + T1[T2[read_block]]) % 65536;
        }

    } else {

             // Processing the file reading more bytes to gain performance
            if (tamanho % 256 == 0){
                residuo = 0;
            } else {
                residuo = tamanho % 256;  // checks file size not multiplied by 256
            }

            tamanho = tamanho - residuo;

             for(ct=0;ct<tamanho;ct=ct+256){
                 fread(&read_block2,sizeof(read_block2),1,p1);

                for (ct2=0;ct2<256;ct2++){
                     if (posic == 0){
                          troca = T2[read_block2[ct2]];
                          tmp1 = T1[read_block2[ct2]];
                          posicao = (troca + controle) % 256;
                          T1[read_block2[ct2]] = T1[posicao];
                          T1[posicao]=tmp1;
                          posic = 1;
                     } else {
                          troca = T1[read_block2[ct2]];
                          tmp1 = T2[read_block2[ct2]];
                          posicao = (troca + controle) % 256;
                          T2[read_block2[ct2]] = T2[posicao];
                          T2[posicao]=tmp1;
                          posic = 0;
                     }

                      controle = (controle + 1) % 256;
                      acumula = (acumula + T1[T2[read_block2[ct2]]]) % 65536;
                }
             }

            // Processing the rest of the file:
            if (residuo > 0){
                 // Processing the byte to byte file
                 for(ct=0;ct<residuo;ct++){
                     fread(&read_block,sizeof(read_block),1,p1);

                     if (posic == 0){
                          troca = T2[read_block];
                          tmp1 = T1[read_block];
                          posicao = (troca + controle) % 256;
                          T1[read_block] = T1[posicao];
                          T1[posicao]=tmp1;
```

```
                         posic = 1;
                     } else {
                         troca = T1[read_block];
                         tmp1 = T2[read_block];
                         posicao = (troca + controle) % 256;
                         T2[read_block] = T2[posicao];
                         T2[posicao]=tmp1;
                         posic = 0;
                     }

                      controle = (controle + 1) % 256;
                      acumula = (acumula + T1[T2[read_block]]) % 65536;
                }
          }
    }

    tmp1 = (unsigned int) acumula / 256;
    tmp2 = acumula % 256;

    // Operation Sum
    for (ct=0;ct<256;ct++){
         T1[ct] = (T1[ct] + tmp1) % 256;
         T2[ct] = (T2[ct] + tmp2) % 256;
    }

    fflush(p1);
    fclose(p1);
}

/*----------------------------------------------------------------------------------------------------
512-BIT BLOCK FINAL MIXING FUNCTION
------------------------------------------------------------------------------------------------------*/
void mixword_final(){
    register unsigned int round = 0, ct, tmp, limite = 0;
    unsigned int palavras[4];
    unsigned char indice1, indice2;
    register unsigned char control_perm=0;
    unsigned int* wpalavra = malloc(sizeof(unsigned int) * 4);   // pointer to exchange the words
    register unsigned int p0, p1, p2, p3;

    // Calculates how many laps will be executed:
    for (ct=0;ct<64;ct++){
         limite = limite ^ T1[BLOCK[ct]];
         limite = limite + T2[BLOCK[ct]];
         limite = (limite + ( (T1[BLOCK[ct]]+1) * (T2[BLOCK[ct]]+1) )) % 8191;
    }

    limite = 8192 + limite;

    for(round=1;round<=limite;round++){

         if (round % 4 == 0) {
             p0 = (TRAN32B(BLOCK[0],BLOCK[16], BLOCK[4],BLOCK[21], BLOCK[8],BLOCK[26] , BLOCK[12],BLOCK[31]) + TRAN32M1(BLOCK[32], BLOCK[38],
BLOCK[40],BLOCK[46])) ^ TRAN32M2(BLOCK[48], BLOCK[55], BLOCK[58], BLOCK[61]);
             p1 = (TRAN32B(BLOCK[1],BLOCK[17], BLOCK[5],BLOCK[22], BLOCK[9],BLOCK[27] , BLOCK[13],BLOCK[28]) + TRAN32M1(BLOCK[33], BLOCK[39],
BLOCK[41],BLOCK[47])) ^TRAN32M2(BLOCK[49], BLOCK[52], BLOCK[59], BLOCK[62]);
             p2 = (TRAN32B(BLOCK[2],BLOCK[18], BLOCK[6],BLOCK[23], BLOCK[10],BLOCK[24], BLOCK[14],BLOCK[29]) + TRAN32M1(BLOCK[34], BLOCK[36],
BLOCK[42],BLOCK[44])) ^ TRAN32M2(BLOCK[50], BLOCK[53], BLOCK[56], BLOCK[63]);
             p3 = (TRAN32B(BLOCK[3],BLOCK[19], BLOCK[7],BLOCK[20], BLOCK[11],BLOCK[25], BLOCK[15],BLOCK[30]) + TRAN32M1(BLOCK[35], BLOCK[37],
BLOCK[43],BLOCK[45])) ^ TRAN32M2(BLOCK[51], BLOCK[54], BLOCK[57], BLOCK[60]);
         } else if (round % 4 == 1) {
             p0 = (TRAN32B(BLOCK[4],BLOCK[16], BLOCK[8],BLOCK[21], BLOCK[12],BLOCK[26] , BLOCK[0],BLOCK[31]) + TRAN32M1(BLOCK[32], BLOCK[38],
BLOCK[40],BLOCK[46]))^ TRAN32M2(BLOCK[48], BLOCK[55], BLOCK[58], BLOCK[61]);
             p1 = (TRAN32B(BLOCK[5],BLOCK[17], BLOCK[9],BLOCK[22], BLOCK[13],BLOCK[27] , BLOCK[1],BLOCK[28])+ TRAN32M1(BLOCK[33], BLOCK[39],
BLOCK[41],BLOCK[47])) ^TRAN32M2(BLOCK[49], BLOCK[52], BLOCK[59], BLOCK[62]);
             p2 = (TRAN32B(BLOCK[6],BLOCK[18], BLOCK[10],BLOCK[23], BLOCK[14],BLOCK[24], BLOCK[2],BLOCK[29]) + TRAN32M1(BLOCK[34], BLOCK[36],
BLOCK[42],BLOCK[44]))^ TRAN32M2(BLOCK[50], BLOCK[53], BLOCK[56], BLOCK[63]);
             p3 = (TRAN32B(BLOCK[7],BLOCK[19], BLOCK[11],BLOCK[20], BLOCK[15],BLOCK[25], BLOCK[3],BLOCK[30]) + TRAN32M1(BLOCK[35], BLOCK[37],
BLOCK[43],BLOCK[45]))^ TRAN32M2(BLOCK[51], BLOCK[54], BLOCK[57], BLOCK[60]);
         } else if (round % 4 == 2) {
             p0 = (TRAN32B(BLOCK[8],BLOCK[16], BLOCK[12],BLOCK[21] , BLOCK[0],BLOCK[26], BLOCK[4],BLOCK[31]) + TRAN32M1(BLOCK[32], BLOCK[38],
BLOCK[40],BLOCK[46]))^ TRAN32M2(BLOCK[48], BLOCK[55], BLOCK[58], BLOCK[61]);
             p1 = (TRAN32B(BLOCK[9],BLOCK[17], BLOCK[13],BLOCK[22] , BLOCK[1],BLOCK[27], BLOCK[5],BLOCK[28])+ TRAN32M1(BLOCK[33], BLOCK[39],
BLOCK[41],BLOCK[47])) ^TRAN32M2(BLOCK[49], BLOCK[52], BLOCK[59], BLOCK[62]);
             p2 = (TRAN32B(BLOCK[10],BLOCK[18], BLOCK[14],BLOCK[23], BLOCK[2],BLOCK[24], BLOCK[6],BLOCK[29]) + TRAN32M1(BLOCK[34], BLOCK[36],
BLOCK[42],BLOCK[44]))^ TRAN32M2(BLOCK[50], BLOCK[53], BLOCK[56], BLOCK[63]);
             p3 = (TRAN32B(BLOCK[11],BLOCK[19], BLOCK[15],BLOCK[20], BLOCK[3],BLOCK[25], BLOCK[7],BLOCK[30])+ TRAN32M1(BLOCK[35], BLOCK[37],
BLOCK[43],BLOCK[45]))^ TRAN32M2(BLOCK[51], BLOCK[54], BLOCK[57], BLOCK[60]);
         } else if (round % 4 == 3) {
             p0 = (TRAN32B(BLOCK[12],BLOCK[16] , BLOCK[0],BLOCK[21], BLOCK[4],BLOCK[26], BLOCK[8],BLOCK[31]) + TRAN32M1(BLOCK[32], BLOCK[38],
BLOCK[40],BLOCK[46]))^ TRAN32M2(BLOCK[48], BLOCK[55], BLOCK[58], BLOCK[61]);
             p1 = (TRAN32B(BLOCK[13],BLOCK[17] , BLOCK[1],BLOCK[22], BLOCK[5],BLOCK[27], BLOCK[9],BLOCK[28])+ TRAN32M1(BLOCK[33], BLOCK[39],
BLOCK[41],BLOCK[47])) ^TRAN32M2(BLOCK[49], BLOCK[52], BLOCK[59], BLOCK[62]);
             p2 = (TRAN32B(BLOCK[14],BLOCK[18], BLOCK[2],BLOCK[23], BLOCK[6],BLOCK[24], BLOCK[10],BLOCK[29])+TRAN32M1(BLOCK[34], BLOCK[36],
BLOCK[42],BLOCK[44]))^ TRAN32M2(BLOCK[50], BLOCK[53], BLOCK[56], BLOCK[63]);
             p3 = (TRAN32B(BLOCK[15],BLOCK[19], BLOCK[3],BLOCK[20], BLOCK[7],BLOCK[25], BLOCK[11],BLOCK[30])+ TRAN32M1(BLOCK[35], BLOCK[37],
BLOCK[43],BLOCK[45]))^ TRAN32M2(BLOCK[51], BLOCK[54], BLOCK[57], BLOCK[60]);
         }

         // Modifying the T1 exchange table
         tmp = T1[ T2[ BLOCK[48] ] ];

         T1[ T2[ BLOCK[48] ] ] = T1[ T2[ BLOCK[55] ] ];
         T1[ T2[ BLOCK[55] ] ] = T1[ T2[ BLOCK[58] ] ];
         T1[ T2[ BLOCK[58] ] ] = T1[ T2[ BLOCK[61] ] ];
         T1[ T2[ BLOCK[61] ] ] = T1[ T2[ BLOCK[49] ] ];

         T1[ T2[ BLOCK[49] ] ] = T1[ T2[ BLOCK[52] ] ];
         T1[ T2[ BLOCK[52] ] ] = T1[ T2[ BLOCK[59] ] ];
         T1[ T2[ BLOCK[59] ] ] = T1[ T2[ BLOCK[62] ] ];
         T1[ T2[ BLOCK[62] ] ] = T1[ T2[ BLOCK[50] ] ];

         T1[ T2[ BLOCK[50] ] ] = T1[ T2[ BLOCK[53] ] ];
         T1[ T2[ BLOCK[53] ] ] = T1[ T2[ BLOCK[56] ] ];
         T1[ T2[ BLOCK[56] ] ] = T1[ T2[ BLOCK[63] ] ];
```

```
T1[ T2[ BLOCK[63] ] ] = T1[ T2[ BLOCK[51] ] ];

T1[ T2[ BLOCK[51] ] ] = T1[ T2[ BLOCK[54] ] ];
T1[ T2[ BLOCK[54] ] ] = T1[ T2[ BLOCK[57] ] ];
T1[ T2[ BLOCK[57] ] ] = T1[ T2[ BLOCK[60] ] ];
T1[ T2[ BLOCK[60] ] ] = tmp;

// Modifying the T2 exchange table
tmp = T2[ T1[ BLOCK[32] ] ];

T2[ T1[ BLOCK[32] ] ] = T2[ T1[ BLOCK[38] ] ];
T2[ T1[ BLOCK[38] ] ] = T2[ T1[ BLOCK[40] ] ];
T2[ T1[ BLOCK[40] ] ] = T2[ T1[ BLOCK[46] ] ];
T2[ T1[ BLOCK[46] ] ] = T2[ T1[ BLOCK[33] ] ];

T2[ T1[ BLOCK[33] ] ] = T2[ T1[ BLOCK[39] ] ];
T2[ T1[ BLOCK[39] ] ] = T2[ T1[ BLOCK[41] ] ];
T2[ T1[ BLOCK[41] ] ] = T2[ T1[ BLOCK[47] ] ];
T2[ T1[ BLOCK[47] ] ] = T2[ T1[ BLOCK[34] ] ];

T2[ T1[ BLOCK[34] ] ] = T2[ T1[ BLOCK[36] ] ];
T2[ T1[ BLOCK[36] ] ] = T2[ T1[ BLOCK[42] ] ];
T2[ T1[ BLOCK[42] ] ] = T2[ T1[ BLOCK[44] ] ];
T2[ T1[ BLOCK[44] ] ] = T2[ T1[ BLOCK[35] ] ];

T2[ T1[ BLOCK[35] ] ] = T2[ T1[ BLOCK[37] ] ];
T2[ T1[ BLOCK[37] ] ] = T2[ T1[ BLOCK[43] ] ];
T2[ T1[ BLOCK[43] ] ] = T2[ T1[ BLOCK[45] ] ];
T2[ T1[ BLOCK[45] ] ] = tmp;

// Rotating the Maps
indice1 = T2[round % 256];
for(ct=0;ct<256;ct++){
    T1[ct] = (T1[ct] + indice1); // % 256;
}

indice2 = T1[(round+128) % 256];
for(ct=0;ct<256;ct++){
    T2[ct] = (T2[ct] + indice2); // % 256;
}

// Diffusion of the words in 4 rounds
for (ct=0;ct<4;ct++){
    p0 ^= (ROTL32(~p1,13) ^ ROTL32(p2,3)) + ROTL32(~p3,27);
    p1 += (ROTL32(p0,14) ^ ROTL32(~p2,11)) + ROTL32(p3,26);
    p2 ^= (ROTL32(~p0,9) ^ ROTL32(p1,20)) + ROTL32(~p3,28);
    p3 += (ROTL32(p0,17) ^ ROTL32(~p1,2)) + ROTL32(p2,1);

    p0 ^= (ROTL32(~p1,25) ^ ROTL32(p2,7)) + ROTL32(~p3,18);
    p1 += (ROTL32(p0,10) ^ ROTL32(~p2,8)) + ROTL32(p3,23);
    p2 ^= (ROTL32(~p0,15) ^ ROTL32(p1,31)) + ROTL32(~p3,29);
    p3 += (ROTL32(p0,30) ^ ROTL32(~p1,16)) + ROTL32(p2,21);
```

```
            p0 ^= (ROTL32(~p1,19) ^ ROTL32(p2,24)) + ROTL32(~p3,12);
            p1 += (ROTL32(p0,22) ^ ROTL32(~p2,4)) + ROTL32(p3,6);
            p2 ^= (ROTL32(~p0,5) ^ ROTL32(p1,8)) + ROTL32(~p3,13);
            p3 += (ROTL32(p0,14) ^ ROTL32(~p1,24)) + ROTL32(p2,20);

            // In this part apply all the permutations resulting from the combinations of the prime numbers up to 31 (are 7920 combinations)
            tmp = (p0 % 7920);
            p0 = ~(ROTL32(p0,PERMUTACAO[tmp][0]));
            p1 = ROTL32(p1,PERMUTACAO[tmp][1]) ;
            p2 = ~(ROTL32(p2,PERMUTACAO[tmp][2])) ;
            p3 = ROTL32(p3,PERMUTACAO[tmp][3]);

            tmp = p0;
            p0 = p1;
            p1 = p2;
            p2 = p3;
            p3 = tmp;
        }

        palavras[0] = p0;
        palavras[1] = p1;
        palavras[2] = p2;
        palavras[3] = p3;

        // Do the binary permutation in sub-block 1
        permutation_binary_128(palavras,wpalavra);

        palavras[0] = *(wpalavra + 0);
        palavras[1] = *(wpalavra + 1);
        palavras[2] = *(wpalavra + 2);
        palavras[3] = *(wpalavra + 3);

        // Rotating the sub-blocks
        rotate_block(palavras);

        // Every 16 laps makes the permutation of 64 bytes
        if (round % 16 == 0) {
            permutation_block(control_perm);
           ++control_perm;
           if(control_perm>7){
               control_perm = 0;
           }
        }

        // Every 64 turns makes the binary permutation in 512 bits
        if (round % 64 == 0){
            permutation_binary_512();
        }

    }

    free(wpalavra);
}
```

```
/*---------------------------------------------------------------------------------------------------
BINARY PERMUTATION FUNCTION IN 512 BITS!!!
---------------------------------------------------------------------------------------------------*/
void permutation_binary_512() {
    unsigned char vetor[512],vetor2[512];
    register unsigned int ct, posicao = 0, marcador, posicao_final;
    register unsigned int contador, contador_block;
    unsigned int palavras[4];
    register unsigned int tmp, tmp1, tmp2 ;
    register int controle;
    register unsigned char t1, t2, t3, t4;

    for (ct=0;ct<512;ct++){
        vetor[ct]=0;
    }

    // Calculating the end position of the block
    posicao_final = 0;
    for (ct=0;ct<64;ct++){
        posicao_final = posicao_final + BLOCK[ct];
    }
    posicao_final = posicao_final % 2; // This variable will change the position of the bits after the permutation

    // Transforming the block for binary notation
    posicao =0;
    marcador = 0;
    for (contador=0;contador<4;contador++){

        palavras[0] = TRAN32(BLOCK[marcador],  BLOCK[4+marcador], BLOCK[8+marcador], BLOCK[12+marcador]);
        palavras[1] = TRAN32(BLOCK[1+marcador],BLOCK[5+marcador], BLOCK[9+marcador], BLOCK[13+marcador]);
        palavras[2] = TRAN32(BLOCK[2+marcador],BLOCK[6+marcador], BLOCK[10+marcador], BLOCK[14+marcador]);
        palavras[3] = TRAN32(BLOCK[3+marcador],BLOCK[7+marcador], BLOCK[11+marcador], BLOCK[15+marcador]);

        // Converting the words to binary
        for(ct=0;ct<4;ct++){
            for(controle=31;controle>=0;controle--){
                if (palavras[ct] >= BINARIO[controle]){
                    vetor[posicao] = 1;
                    palavras[ct] = palavras[ct] - BINARIO[controle];
                }
                ++posicao;
            }
        }

        marcador = marcador + 16;
    }

        // Reordering the 512 bits
        // Part 1:
        posicao=0;
        for(ct=0;ct<256;ct++){
            vetor2[posicao] = vetor[T1[ct]];
            ++posicao;
        }

        // Part 2:
        posicao=256;
        for(ct=0;ct<256;ct++){
            vetor2[posicao] = vetor[T2[ct]+256];
            ++posicao;
        }

        // Position Inversion Routine
        posicao = 256;
        if (posicao_final == 0){
            for (ct=0;ct<512;ct++){
                vetor[ct] = vetor2[ct];
            }
        } else {
            for (ct=0;ct<512;ct++){
                vetor[ct] = vetor2[posicao];
                ++posicao;
                if (posicao > 511){
                    posicao = 0;
                }
            }
        }

        // Converting the 512 bits already exchanged into 8-bit elements in the 64 bytes of the block
        posicao = 0;
        contador_block = 0;

        for(contador=0;contador<4;contador++){
            for (ct=0;ct<4;ct++){
                marcador = 31;
                palavras[ct] = 0;

                for(controle=0+posicao;controle<32+posicao;controle++){
                    palavras[ct] = palavras[ct] + (vetor[controle] * BINARIO[marcador]);
                    --marcador;
                }
                posicao = posicao + 32;
            }

            // Placing the result in the vector block
            for(ct=0;ct<4;ct++){

               tmp = palavras[ct];
                t1 = tmp >> 24;
                t2 = (tmp >> 16) & 255;
                t3 = (tmp >>  8) & 255;
                t4 = tmp & 255;

                BLOCK[contador_block] = t1;
                BLOCK[contador_block+1] = t2;
```

```
                BLOCK[contador_block+2] = t3;
                BLOCK[contador_block+3] = t4;

                contador_block = contador_block + 4;
            }

        }
}

/*--------------------------------------------------------------------------------------------------
128-BIT BINARY EXCHANGE FUNCTION!!!
---------------------------------------------------------------------------------------------------*/
unsigned int* permutation_binary_128(unsigned int palavras[], unsigned int* wpalavra ) {
    unsigned char vetor[128],vetor2[128];
    register unsigned int ct, posicao = 0, marcador;
    register int controle;

    for (ct=0;ct<128;ct++){
        vetor[ct]=0;
    }

    // Converting the words to binary
    for(ct=0;ct<4;ct++){
        for(controle=31;controle>=0;controle--){
            if (palavras[ct] >= BINARIO[controle]){
                vetor[posicao] = 1;
                palavras[ct] = palavras[ct] - BINARIO[controle];
            }
            ++posicao;
        }
    }

    // Reordering the 128 bits
    posicao=0;
    for(ct=0;ct<256;ct++){
        if (T1[ct] <= 127){
            vetor2[posicao] = vetor[T1[ct]];
            ++posicao;
        }
    }

    // Converting the bits to 32-bit words
    posicao = 0;
    for (ct=0;ct<4;ct++){
        marcador = 31;
        for(controle=0+posicao;controle<32+posicao;controle++){
            palavras[ct] = palavras[ct] + (vetor2[controle] * BINARIO[marcador]);
            --marcador;
        }
        posicao = posicao + 32;
    }

    *(wpalavra + 0) = palavras[0];
    *(wpalavra + 1) = palavras[1];
    *(wpalavra + 2) = palavras[2];
    *(wpalavra + 3) = palavras[3];
}

/*--------------------------------------------------------------------------------------------------
FUNCTION TO CALCULATE ALL POSSIBLE COMBINATIONS OF PRIME NUMBERS UP TO 31 OUT OF 4,
TOTALING 7920 COMBINATIONS!!!
----------------------------------------------------------------------------------------------------*/
void calculate_permutations(){
    unsigned char vetor[11] = {2,3,5,7,11,13,17,19,23,29,31};
    unsigned char ordem[4], guarda[4], p[4];
    register unsigned char ct, erro, ct2, tmp;
    register unsigned int contador = 0;

    p[0] = 0;
    p[1] = 0;
    p[2] = 0;
    p[3] = 0;

    for(;;){
        for (ct=0;ct<4;ct++){
            ordem[ct] = vetor[p[ct]];
            guarda[ct] = vetor[p[ct]];
        }

        // Sort the vector to see repeated elements
        ct2 = 0;
        for(;;){
            if (ordem[ct2] > ordem[ct2+1]) {
                tmp = ordem[ct2];
                ordem[ct2] = ordem[ct2+1];
                ordem[ct2+1] = tmp;
                ct2 = 0;
            } else {
                ++ct2;
            }

            if (ct2 > 2){
                break;
            }
        }

        erro = 0;
        for (ct=0;ct<3;ct++){
            if (ordem[ct] >= ordem[ct+1]){
                erro = 1;
                break;
            }
        }

        // Saving the valid permutations:
        if (erro == 0){
            for(ct=0;ct<4;ct++){
                PERMUTACAO[contador][ct] = guarda[ct];
            }
            ++contador;
        }

        ++p[0];
        if (p[0] > 10){
            p[0] = 0 ;
            ++p[1];
        }

        if (p[1] >  10){
            p[1] = 0 ;
            ++p[2];
        }

        if (p[2] >  10){
            p[2] = 0 ;
            ++p[3];
        }

        if (p[3] >  10){
            break;
        }
    }
}

/*-------------------------------------------------------------------------------------------------
THIS FUNCTION INITIALIZES THE VALUES OF T1 AND T2
---------------------------------------------------------------------------------------------------*/
void reset_maps(){

    T1[  0] =  204;
    T1[  1] =  193;
    T1[  2] =   96;
    T1[  3] =   10;
    T1[  4] =  100;
    T1[  5] =  208;
    T1[  6] =  104;
    T1[  7] =  212;
    T1[  8] =  109;
    T1[  9] =   52;
    T1[ 10] =   70;
    T1[ 11] =   95;
    T1[ 12] =  108;
    T1[ 13] =   99;
    T1[ 14] =  103;
    T1[ 15] =   11;
    T1[ 16] =  107;
    T1[ 17] =   98;
    T1[ 18] =  102;
    T1[ 19] =  106;
```

```
T1[ 20] =  118;
T1[ 21] =   22;
T1[ 22] =  122;
T1[ 23] =  111;
T1[ 24] =  130;
T1[ 25] =    1;
T1[ 26] =  154;
T1[ 27] =  162;
T1[ 28] =  166;
T1[ 29] =  115;
T1[ 30] =  186;
T1[ 31] =  198;
T1[ 32] =  119;
T1[ 33] =  238;
T1[ 34] =  250;
T1[ 35] =  123;
T1[ 36] =   30;
T1[ 37] =  127;
T1[ 38] =  216;
T1[ 39] =  131;
T1[ 40] =   61;
T1[ 41] =  135;
T1[ 42] =   14;
T1[ 43] =  139;
T1[ 44] =  143;
T1[ 45] =  147;
T1[ 46] =  151;
T1[ 47] =  112;
T1[ 48] =  220;
T1[ 49] =   54;
T1[ 50] =  155;
T1[ 51] =   57;
T1[ 52] =  159;
T1[ 53] =   60;
T1[ 54] =  163;
T1[ 55] =  167;
T1[ 56] =   63;
T1[ 57] =   17;
T1[ 58] =  171;
T1[ 59] =   69;
T1[ 60] =  205;
T1[ 61] =  175;
T1[ 62] =    7;
T1[ 63] =  179;
T1[ 64] =   75;
T1[ 65] =  183;
T1[ 66] =  187;
T1[ 67] =  116;
T1[ 68] =  191;
T1[ 69] =   97;
T1[ 70] =  165;
T1[ 71] =    2;
T1[ 72] =  195;
```

```
T1[ 73] =   20;
T1[ 74] =   90;
T1[ 75] =  199;
T1[ 76] =   88;
T1[ 77] =  203;
T1[ 78] =  207;
T1[ 79] =  211;
T1[ 80] =  133;
T1[ 81] =  215;
T1[ 82] =  225;
T1[ 83] =  224;
T1[ 84] =   43;
T1[ 85] =  219;
T1[ 86] =   79;
T1[ 87] =  223;
T1[ 88] =  120;
T1[ 89] =  227;
T1[ 90] =   23;
T1[ 91] =  231;
T1[ 92] =  235;
T1[ 93] =  153;
T1[ 94] =  185;
T1[ 95] =  239;
T1[ 96] =    0;
T1[ 97] =  243;
T1[ 98] =  247;
T1[ 99] =  251;
T1[100] =  228;
T1[101] =   26;
T1[102] =  255;
T1[103] =  124;
T1[104] =   29;
T1[105] =  232;
T1[106] =  128;
T1[107] =  121;
T1[108] =  141;
T1[109] =   32;
T1[110] =   13;
T1[111] =  114;
T1[112] =  217;
T1[113] =   12;
T1[114] =   34;
T1[115] =  142;
T1[116] =   18;
T1[117] =  190;
T1[118] =  132;
T1[119] =   33;
T1[120] =  236;
T1[121] =   48;
T1[122] =   35;
T1[123] =  240;
T1[124] =  249;
T1[125] =  136;
```

```
T1[126] =   38;
T1[127] =   72;
T1[128] =   84;
T1[129] =  244;
T1[130] =  237;
T1[131] =   25;
T1[132] =   41;
T1[133] =  177;
T1[134] =    4;
T1[135] =  248;
T1[136] =  140;
T1[137] =   44;
T1[138] =   64;
T1[139] =   73;
T1[140] =  144;
T1[141] =   47;
T1[142] =  252;
T1[143] =  148;
T1[144] =  101;
T1[145] =   50;
T1[146] =  197;
T1[147] =   46;
T1[148] =  152;
T1[149] =   53;
T1[150] =  113;
T1[151] =  145;
T1[152] =   82;
T1[153] =   91;
T1[154] =  156;
T1[155] =   56;
T1[156] =   16;
T1[157] =   55;
T1[158] =  160;
T1[159] =  157;
T1[160] =   37;
T1[161] =   59;
T1[162] =  221;
T1[163] =  164;
T1[164] =    6;
T1[165] =   62;
T1[166] =  169;
T1[167] =  209;
T1[168] =   65;
T1[169] =  168;
T1[170] =  229;
T1[171] =   68;
T1[172] =   28;
T1[173] =  172;
T1[174] =    9;
T1[175] =  110;
T1[176] =  189;
T1[177] =  241;
T1[178] =  134;
```

```
T1[179] =  158;
T1[180] =  170;
T1[181] =  178;
T1[182] =  182;
T1[183] =  194;
T1[184] =   21;
T1[185] =  202;
T1[186] =  206;
T1[187] =  218;
T1[188] =  226;
T1[189] =  234;
T1[190] =  242;
T1[191] =  254;
T1[192] =   27;
T1[193] =   71;
T1[194] =  253;
T1[195] =  176;
T1[196] =   67;
T1[197] =   76;
T1[198] =    5;
T1[199] =   74;
T1[200] =  125;
T1[201] =  180;
T1[202] =   87;
T1[203] =   49;
T1[204] =   77;
T1[205] =   93;
T1[206] =  184;
T1[207] =  137;
T1[208] =   19;
T1[209] =   85;
T1[210] =   80;
T1[211] =  181;
T1[212] =    8;
T1[213] =   83;
T1[214] =  188;
T1[215] =  105;
T1[216] =  149;
T1[217] =   58;
T1[218] =   86;
T1[219] =  192;
T1[220] =  213;
T1[221] =   40;
T1[222] =  126;
T1[223] =  138;
T1[224] =  146;
T1[225] =  150;
T1[226] =   15;
T1[227] =  174;
T1[228] =   94;
T1[229] =  210;
T1[230] =  214;
T1[231] =  222;
```

```
T1[232] =  230;
T1[233] =   24;
T1[234] =  246;
T1[235] =  117;
T1[236] =    3;
T1[237] =  201;
T1[238] =  233;
T1[239] =  245;
T1[240] =   31;
T1[241] =  196;
T1[242] =   36;
T1[243] =   89;
T1[244] =   39;
T1[245] =   42;
T1[246] =   45;
T1[247] =   51;
T1[248] =   66;
T1[249] =  129;
T1[250] =  161;
T1[251] =   92;
T1[252] =   78;
T1[253] =   81;
T1[254] =  200;
T1[255] =  173;

T2[  0] =  240;
T2[  1] =   49;
T2[  2] =  145;
T2[  3] =  148;
T2[  4] =   52;
T2[  5] =  244;
T2[  6] =  152;
T2[  7] =  193;
T2[  8] =   56;
T2[  9] =  248;
T2[ 10] =   41;
T2[ 11] =  229;
T2[ 12] =  241;
T2[ 13] =  156;
T2[ 14] =  137;
T2[ 15] =  157;
T2[ 16] =  165;
T2[ 17] =  252;
T2[ 18] =   60;
T2[ 19] =    6;
T2[ 20] =   14;
T2[ 21] =   50;
T2[ 22] =   66;
T2[ 23] =   21;
T2[ 24] =   74;
T2[ 25] =   77;
T2[ 26] =  114;
T2[ 27] =  118;
```

```
T2[ 28] =  160;
T2[ 29] =  222;
T2[ 30] =  226;
T2[ 31] =  234;
T2[ 32] =  242;
T2[ 33] =  250;
T2[ 34] =   97;
T2[ 35] =  109;
T2[ 36] =   64;
T2[ 37] =  164;
T2[ 38] =    9;
T2[ 39] =   69;
T2[ 40] =  149;
T2[ 41] =  185;
T2[ 42] =  213;
T2[ 43] =   68;
T2[ 44] =  168;
T2[ 45] =  129;
T2[ 46] =    3;
T2[ 47] =   72;
T2[ 48] =    7;
T2[ 49] =  172;
T2[ 50] =   11;
T2[ 51] =   15;
T2[ 52] =   19;
T2[ 53] =   23;
T2[ 54] =  177;
T2[ 55] =   27;
T2[ 56] =   31;
T2[ 57] =   35;
T2[ 58] =   39;
T2[ 59] =   43;
T2[ 60] =   47;
T2[ 61] =  176;
T2[ 62] =   51;
T2[ 63] =   55;
T2[ 64] =   76;
T2[ 65] =   59;
T2[ 66] =  141;
T2[ 67] =   63;
T2[ 68] =   67;
T2[ 69] =   71;
T2[ 70] =    2;
T2[ 71] =   10;
T2[ 72] =   18;
T2[ 73] =   26;
T2[ 74] =   46;
T2[ 75] =   75;
T2[ 76] =   80;
T2[ 77] =   62;
T2[ 78] =   33;
T2[ 79] =   79;
T2[ 80] =   89;
```

```
T2[ 81] =  121;
T2[ 82] =  106;
T2[ 83] =   83;
T2[ 84] =  110;
T2[ 85] =  126;
T2[ 86] =  130;
T2[ 87] =  142;
T2[ 88] =  146;
T2[ 89] =   87;
T2[ 90] =  170;
T2[ 91] =  174;
T2[ 92] =  182;
T2[ 93] =  186;
T2[ 94] =  190;
T2[ 95] =  194;
T2[ 96] =   91;
T2[ 97] =  206;
T2[ 98] =  210;
T2[ 99] =  214;
T2[100] =  218;
T2[101] =   95;
T2[102] =  254;
T2[103] =   99;
T2[104] =  103;
T2[105] =  180;
T2[106] =  107;
T2[107] =  111;
T2[108] =  115;
T2[109] =   84;
T2[110] =  119;
T2[111] =   61;
T2[112] =  123;
T2[113] =  127;
T2[114] =  131;
T2[115] =  135;
T2[116] =  139;
T2[117] =    1;
T2[118] =  205;
T2[119] =  143;
T2[120] =  147;
T2[121] =  151;
T2[122] =  155;
T2[123] =  184;
T2[124] =   53;
T2[125] =   88;
T2[126] =  159;
T2[127] =  101;
T2[128] =  169;
T2[129] =  163;
T2[130] =  167;
T2[131] =  171;
T2[132] =  175;
T2[133] =   81;
```

```
T2[134] =  179;
T2[135] =  233;
T2[136] =  183;
T2[137] =  187;
T2[138] =  191;
T2[139] =  188;
T2[140] =  195;
T2[141] =  199;
T2[142] =  203;
T2[143] =   92;
T2[144] =  207;
T2[145] =  211;
T2[146] =  215;
T2[147] =  197;
T2[148] =  253;
T2[149] =  219;
T2[150] =  223;
T2[151] =  227;
T2[152] =   25;
T2[153] =  231;
T2[154] =  192;
T2[155] =  235;
T2[156] =  239;
T2[157] =  243;
T2[158] =  247;
T2[159] =   96;
T2[160] =  251;
T2[161] =  245;
T2[162] =  255;
T2[163] =  196;
T2[164] =    0;
T2[165] =  161;
T2[166] =  100;
T2[167] =    4;
T2[168] =  104;
T2[169] =   45;
T2[170] =  189;
T2[171] =  200;
T2[172] =   73;
T2[173] =  113;
T2[174] =  217;
T2[175] =   37;
T2[176] =  108;
T2[177] =  225;
T2[178] =    8;
T2[179] =   13;
T2[180] =  133;
T2[181] =  181;
T2[182] =  209;
T2[183] =  204;
T2[184] =  112;
T2[185] =  208;
T2[186] =   12;
```

```
T2[187] =   93;
T2[188] =   16;
T2[189] =  116;
T2[190] =  212;
T2[191] =   20;
T2[192] =  173;
T2[193] =  120;
T2[194] =   30;
T2[195] =   34;
T2[196] =   42;
T2[197] =   85;
T2[198] =   82;
T2[199] =  153;
T2[200] =  237;
T2[201] =   98;
T2[202] =  102;
T2[203] =  134;
T2[204] =  138;
T2[205] =  150;
T2[206] =   24;
T2[207] =  158;
T2[208] =  162;
T2[209] =  166;
T2[210] =  178;
T2[211] =  202;
T2[212] =  238;
T2[213] =  216;
T2[214] =  124;
T2[215] =   28;
T2[216] =    5;
T2[217] =  220;
T2[218] =  125;
T2[219] =  105;
T2[220] =   32;
T2[221] =  128;
T2[222] =  224;
T2[223] =   57;
T2[224] =  132;
T2[225] =   36;
T2[226] =   65;
T2[227] =   17;
T2[228] =   40;
T2[229] =  228;
T2[230] =  136;
T2[231] =   29;
T2[232] =  201;
T2[233] =   22;
T2[234] =   38;
T2[235] =  140;
T2[236] =   54;
T2[237] =   58;
T2[238] =   70;
T2[239] =   78;
```

```
    T2[240] =   86;
    T2[241] =   90;
    T2[242] =   94;
    T2[243] =  232;
    T2[244] =  122;
    T2[245] =  154;
    T2[246] =  198;
    T2[247] =  230;
    T2[248] =  246;
    T2[249] =   44;
    T2[250] =  236;
    T2[251] =  117;
    T2[252] =  144;
    T2[253] =  221;
    T2[254] =  249;
    T2[255] =   48;
}
```