Watch your Constants: Malicious Streebog

Riham AlTawy and Amr M. Youssef
Concordia Institute for Information Systems Engineering,
Concordia University, Montréal, Québec, Canada

Abstract. In August 2012, the Streebog hash function was selected as the new Russian cryptographic hash standard (GOST R 34.11-2012). In this paper, we investigate the new standard in the context of malicious hashing and present a practical collision for a malicious version of the full hash function. In particular, we apply the rebound attack to find three solutions for three different differential paths for four rounds, and using the freedom of the round constants we connect them to obtain a collision for the twelve rounds of the compression function. Additionally, and due to the simple processing of the counter, we bypass the barrier of the checksum finalization step and transfer the compression function collision to the hash function output with no additional cost. The presented attack has a practical complexity and is verified by an example. While the results of this paper may not have a direct impact on the security of the current Streebog hash function, it presents an urge for the designers to publish the origin of the used parameters and the rational behind their choices in order for this function to gain enough confidence and wide spread adoption by the security community.

Keywords: Cryptanalysis, Hash functions, Malicious hashing, Rebound attacks, GOST R 34.11-2012, Streebog.

1 Introduction

Research on malicious cryptographic primitives has always been thought of as the work of intelligence agencies. The belief that governmental spy agencies work hard to incorporate backdoors in their primitives, which enables the efficient manipulation of certain security properties, has always been lurking in the cryptographic community. This belief was further strengthened last year after Edward Snowden exposed the existence of the NSA’s Bullrun decryption project [32]. Leaked documents have shown that the NSA has deliberately inserted a backdoor in the standardized pseudorandom number generator Dual_EC_DRBG [33]. This backdoor provides the knowledge of the internal state of the generator and accordingly its subsequent outputs. Additionally, it is also speculated that NSA paid RSA Security $10 million in a secret deal to use Dual_EC_DRBG as the default pseudorandom number generator in the RSA BSAFE cryptography library [33]. With Dual_EC_DRBG being recommended by NIST at that time, these revelations have raised suspicions with respect to the NIST standards being manipulated by the NSA, particularly, after voices from the cryptographic community began suggesting the possibility of the NSA compromising the NIST’s recommended elliptic curve constants [29].
Only few papers have been peer reviewed in public venues in the area of malicious cryptography. Young and Young were among the firsts to address the topic of malicious cryptography through their cryptovirology project [34]. Later Rijmen and Preneel proposed malicious versions of CAST and LOKI by hiding linear relations in the used Sboxes [27]. Work related to malicious ciphers, implementations and pseudorandom generators includes [7, 25, 8, 26]. Although most of the previous work focused on ciphers, just recently the concept of malicious hashing have been introduced in [6, 2]. Specifically, Albertini et al. proposed a malicious version of SHA-1 by which collisions can be produced in an efficient way. They have used the freedom of the round constants to satisfy a given differential path and generate one block message collisions.

Streebog was proposed in 2010 [18]. It has an output length of 512/256-bit. The compression function employs a 12-round AES-like cipher with $8 \times 8$-byte internal state. The compression function operates in Miyaguchi-Preneel (MP) mode and is plugged in a modified Merkle-Damgård domain extender with a modular checksum finalization step [1]. Streebog officially replaces the previous standard GOST R 34.11-94 which has been theoretically broken in [21, 20]. The new GOST is standardized by IETF as RFC 6896 [11] as well. Unlike the specifications of other hash functions, the reference of the new GOST standard [1] gives no information about how or why the parameters of the function (e.g., round constants, matrix constants, and the number of rounds) have been chosen. This fact opens the door to our analysis, which makes use of exactly two parameters: the heavily random looking independent constants and the number of rounds, to present practical collisions for a malicious version of Streebog. Early works related to the cryptanalysis of Streebog have been introduced in [3–5, 16] and in [13, 10, 30], where practical semi free-start collision and near collision examples for reduced round versions have been presented in only [3, 30].

In this work, we investigate a malicious version of Streebog. We exploit the randomness of the independent round constants and the number of rounds of the compression function to efficiently generate collisions for the compression function. More precisely, we first employ the rebound attack technique proposed in [30] to find three pairs of messages and keys that satisfy a specific three 4-round differential paths independently. In the sequel, we use the freedom of five out of the twelve round constants to connect the three obtained solutions and obtain collisions for the twelve round compression function. Finally, we tune the last constant of the compression function to adjust its output after the feedforward to cancel the effect of the counter $N$ addition of the following compression function call, and append another identical colliding message pair. Hence, we generate a two block messages $2^2$ multicollision structure where two of them have the same modular sum and thus a collision at the output of the hash function. While previous work [6] stated that compression function collisions are not sufficient to generate hash function collision in constructions that incorporate checksum, our results prove that this is not the case with Streebog. Table 1 provides the six new constant used in our malicious version of Streebog. An example of the two
block message collision along with its corresponding digest is provided in Table 3.

The rest of the paper is organized as follows. In the next section, the description of the Streebog hash function along with the notation used throughout the paper are provided. A brief overview of the rebound attack is given in Section 3. Afterwards, in Sections 4, we provide a detailed description of the used approach, the malicious compression function attack and its corresponding complexity. In Section 5, we show how collisions of the malicious hash function are generated using the attack presented in Sections 4. Finally, the paper is concluded and a short discussion is provided in Section 6.

2 Description of Steebog

Streebog outputs a 512 or 256-bit hash value, where half the last state is truncated when adopting the 256-bit output. The standard specifies two different IVs to be used with the two output lengths. The function can process messages of length up to $2^{512} - 1$. The compression function iterates over 12 rounds of an AES-like cipher with an $8 \times 8$ byte internal state and a final round of key mixing. The compression function operates in Miyaguchi-Preneel mode and is plugged in Merkle-Damgård domain extender with a finalization step. The input message M is padded into a multiple of 512 bits by appending one followed by zeros. The message length for MD-strengthening is further included as an extra separate block, followed by a block of a checksum evaluated by the modulo $2^{512}$ addition of all message blocks as a finalization step. More precisely, let $n = \lfloor \frac{|M|}{512} \rfloor$ and the input message $M = x || m_n || \ldots || m_1 || m_0$, where $|M|$ is length of $M$, and $x$ is an un-complete or an empty block. The message is padded as follows: let $m_{n+1} = 0^{511-|x|} \| 1 \| x$, then the padded message $M = m_{n+1} || m_n || \ldots || m_1 || m_0$. Let $\Sigma = m_{n+1} + \ldots + m_1 + m_0$. The compression function $g_N$ is fed with three inputs:

![Fig. 1. Streebog’s compression function $g_N$](image)

the chaining value $h_{i-1}$, a message block $m_{i-1}$, and the counter of bits hashed
so far $N_{i-1} = 512 \times i$. (see Figure 1). Let $h_i$ be a 512-bit chaining variable. The first state is loaded with the initial value $IV$ and assigned to $h_0$. The hash value of $M$ is computed as follows:

$$h_i \leftarrow g_N(h_{i-1}, m_{i-1}, N_{i-1}) \text{ for } i = 1, 2, \ldots, n + 2$$

$$h_{n+3} \leftarrow g_0(h_{n+2}, |M|, 0)$$

$$h(M) \leftarrow g_0(h_{n+3}, \sum, 0),$$

where $h(M)$ is the hash value of $M$, and $g_0$ is $g_N$ with $N = 0$. As depicted in Figure 1, the compression function $g_N$ consists of:

- $K_N$: a nonlinear whitening round of the chaining value. It takes a 512-bit chaining variable $h_{i-1}$ and a counter of the bits hashed so far $N_{i-1}$ and outputs a 512-bit key $K$.
- $E$: an AES-based cipher that iterates over the message for 12 rounds in addition to a finalization key mixing round. The cipher $E$ takes a 512-bit key $K$ and a 512-bit message block $m$ as a plaintext. As shown in Figure 2, it consists of two similar parallel flows for the state update and the key scheduling.

![Fig. 2. The internal block cipher (E)](image-url)

Both $K_N$ and $E$ operate on an $8 \times 8$ byte key state $K$. $E$ updates an additional $8 \times 8$ byte message state $M$. In one round, a given state is updated by the following sequence of transformations:

- AddKey(X): XOR with either a round key, a constant, or the counter of bits hashed so far ($N$).
- SubBytes (S): A nonlinear byte bijective mapping.
- Transposition (P): Byte permutation.
- Linear Transformation (L): Row multiplication by an MDS matrix in GF(2).

Initially, state $K$ is loaded with the chaining value $h_{i-1}$ and updated by $K_N$ as follows:

$$k_0 = L \circ P \circ S \circ X[N_{i-1}](K).$$
Now $K$ contains the key $k_0$ to be used by the cipher $E$. The message state $M$ is initially loaded with the message block $m$ and $E(k_0, m)$ runs the key scheduling function on state $K$ to generate 12 round keys, $k_1, k_2, \ldots, k_{12}$, as follows:

$$k_i = L \circ P \circ S \circ X[C_{i-1}](k_{i-1}), \text{ for } i = 1, 2, \ldots, 12,$$

where $C_{i-1}$ is the $i^{th}$ round constant. The state $M$ is updated as follows:

$$M_i = L \circ P \circ S \circ X[k_{i-1}](M_{i-1}), \text{ for } i = 1, 2, \ldots, 12.$$

The final round output is given by $E(k_0, m) = M_{12} \oplus k_{12}$. The output of $g_N$ in the Miyaguchi-Preneel mode is $E(K_N(h_{i-1}, N_{i-1}), m_{i-1}) \oplus m_{i-1} \oplus h_{i-1}$ as shown in Figure 1. For further details, the reader is referred to [1].

2.1 Notation

Let $M$ and $K$ be $(8 \times 8)$-byte states denoting the message and key state, respectively. The following notation is used throughout the paper:

- $M_i$: The message state at the beginning of round $i$.
- $M_i^U$: The message state after the $U$ transformation at round $i$, where $U \in X, S, P, L$.
- $M_i[r, c]$: A byte at row $r$ and column $c$ of state $M_i$.
- $M_i[\text{row } r]$: Eight bytes located at row $r$ of $M_i$ state.
- $M_i[\text{col } c]$: Eight bytes located at column $c$ of $M_i$ state.

Same notation applies to $K$.

3 The rebound attack

The rebound attack [22] was proposed by Mendel et al. for the cryptanalysis of AES-based hash functions. It is a differential attack that follows the inside-out or start from the middle approach which is used in the boomerang attack [31]. The attack is composed of three phases, one inbound and two outbounds. The compression function, internal block cipher or permutation of the hash function is divided into three parts. If $C$ is a block cipher, then $C$ is expressed as $C = C_{fw} \circ C_{in} \circ C_{bw}$. The middle part is the inbound phase and the forward and backward parts are the two outbound phases. In the inbound phase, a low probability XOR differential path is used and all possible degrees of freedom are used to satisfy the inbound path. In the two outbound phases, high probability truncated paths [14] are used. In other words, one starts from the middle satisfying $C_{in}$, then hash forward and backward to satisfy $C_{fw}$ and $C_{bw}$ probabilistically. For an $8 \times 8$ byte state, the basic rebound attack finds two states satisfying an inbound phase over two rounds $8 \xrightarrow{C} 64 \xrightarrow{r+1} 8$. The main idea of the attack is to pick random differences at each of the two eight active bytes sates. Then propagate both backward and forward until the output and input of the Sbox, respectively.
Using the Sbox differential distribution table (DDT), we find values that satisfy input and output differentials. This process is further illustrated in Figure 3. The last step of the attack is called the Sbox matching phase and its complexity depends on the Sbox DDT. If the probability of differentials that have solutions is $p$, then the matching probability is given by $p^8$.

![Fig. 3. The rebound attack.](image)

Literature related to the rebound attack includes Mendel et al. first proposal on the ISO standard Whirlpool and the SHA-3 finalist Grøstl [22, 23]. In particular, Mendel et al. presented a 4.5-round collision, 5.5-round semi free-start collision and 7.5-round near collision attacks on the Whirlpool compression function. As for Grøstl-256, a 6-round semi free-start collision is given. Subsequently, rebound attacks have been applied to other AES-based hash functions such as LANE [17], JH [28], Echo [12], Streebog [3], and Grøstl [24]. Various tweaks have been applied to the basic rebound attack in order to construct differential paths that cover more rounds such as merging multiple in-bounds [15], super Sbox cryptanalysis [9], extended 5-round inbound [15], and linearized match-in-the-middle and start-from-the-middle techniques [19]. Lastly, Kölbl and Rechberger presented a practical method to find semi free-start collision for a 4-round AES-based compression function [30]. More precisely, they have proposed a way to first find a specific differential path for $1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$ transition, then use the freedom in the key to find two messages that follow the given path. They have implemented their approach on Streebog and presented a semi free-start collision for the 4-round reduced compression function. In what follows, we show how we used this approach to generate collisions for a malicious version of Streebog compression function.

### 4 Malicious compression function collision

In this section, we give the details of our malicious adaptation of the Streebog compression function that allows us to efficiently construct collisions for the twelve round compression function. Our approach makes use of the heavily random looking independent round constants and the twelve rounds of the compression function. In fact, the specific number of rounds (12) used in Streebog enables us to find three independent solutions for the commonly known $1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$ four round differential path and by changing five constants we can successfully connect them and generate a collision.
Our attack starts by finding the first solution which is a pair of messages and a key that follow the given differential path shown in Figure 4. In doing so, we employ the approach proposed in [30] which is composed of two procedures and is briefly described as follows:

4.1 Building the differential characteristic

In this procedure, one determines the exact differential transitions of the truncated differential trail given above.

1. Choose a random difference at $M^L_{[3, 3]}$ and propagate it backward until the full active state $M_S^3$.
2. For each byte difference in $M_S^3$, save a set of all possible input differences.
3. Create a table $T_L$ of all possible 255 byte difference values $d_3$ (candidates for $M^P_{[+, 3]}$) and their corresponding 8 byte difference values $L(d_3)$ (candidates for $M_X^3$[row 3]). These values are the result of applying the linear transformation $L$ to a difference at column 3.
4. For each row of $M_X^3$, check if there is a possible match with the rows in $T_L$.
5. To achieve the transition from 1 active byte in $M^P_{[+, 3]}$ to 8 active bytes in $M_X^3$[row 3], steps 2 and 4 must be repeated for only one row between states $M_S^3$ and $M_X^3$.

According to the Streebog Sbox differential distribution properties, [30] states that finding the differential characteristic has a complexity of $\approx 2^{20}$ and consequently, the above procedure is repeated $2^{20}$ times.

4.2 Finding a solution for the differential path

Once we have found a characteristic, we now need to find a message pair that follows it. This can be done by performing the following steps:
1. Set the message state at $M_3^X$ with a solution that satisfies the full active state differentials from the above procedure.

2. Use $K_3[\text{col }3]$ to satisfy the solutions of the Sbox differentials at $M_2^X[\text{col }3]$. Also use $K_3[\text{row }3]$ to satisfy the solutions of the Sbox differentials at $M_4^X[\text{col }3]$. Since there is one byte, $K_3[3, 3]$, shared between the two solutions, one needs to repeat the above procedure $2^8$ times. For more details on the specifics of the used technique, the reader is referred to [30].

To this end, we have found a solution to the first differential path with a key input different from that is produced by the standard IV. This solution gives us a specific input and output differences $\Delta_{\text{in}}^1$ and $\Delta_{\text{out}}^1$ at $M_1[3, 3]$ and $M_1^X[3, 3]$, respectively. In the sequel, we restart the above two procedures to search for the second differential characteristic and its solution such that this second search covers rounds five to eight and have an input difference $\Delta_{\text{in}}^2$ at $M_5[3, 3]$ equals to the output difference $\Delta_{\text{out}}^1$ of the first path. Since we restrict the input difference of the second path to a specific value, the complexity of the second procedure of our search is increased by a factor of $2^8$. However, the overall search complexity is still dominated by the first procedure which is about $2^{40}$. Finally, we search for the third and last differential path and its solution which covers rounds nine to twelve. For this path, we have to restrict its input difference $\Delta_{\text{in}}^3$ at $M_9[3, 3]$ to be equal $\Delta_{\text{out}}^2$ at $M_8^X[3, 3]$ and its output difference $\Delta_{\text{out}}^3$ at $M_{12}[3, 3]$ to be equal $\Delta_{\text{in}}^1$ at $M_1[3, 3]$, so that the latter cancels out after the feedforward.

**Connecting the three solutions:** Now that we have the three solutions, we can start tuning specific round constants to connect them. We first work on the first solution’s key output $K_1$, which is different than that generated by the standard IV. To solve this problem, we fix the new $C_1 = LPS(IV) \oplus (K_1^X)$. By doing this, we guarantee that the resulting new key satisfies the first differential path. Thus, the new colliding messages are $m_1 = (M_1^X \oplus LPS(IV))$ and $m_2 = m_1 \oplus \Delta_{\text{in}}^1$.

To connect the first and second solutions, we have to change $K_5$. However, altering $K_5$ affects both $K_4$ and $K_6$, which are restricted by the solutions of the first and second paths, respectively. In order to cancel the propagation of alteration to the latter two round keys, we compute the new two constants $C_5$ and $C_4$ as follows:

$$
K_5 = M_4^X \oplus M_3^X
$$

$$
C_5 = K_5 \oplus K_5^X
$$

$$
C_4 = S^{-1}PL^{-1}(K_5) \oplus K_4,
$$

where $M_4^X$ and $K_4$ are solutions of the first path, while $M_5^X$ and $K_5^X$ are solutions of the second path. To connect the second and third paths, we perform the same procedure to compute the new $C_8$ and $C_9$. Having all the new five constants in place, Table 3, gives an example of a colliding message pair which have the same compression function output using $IV=0$ and $N=0$. 


Collision attack on the full malicious Streebog

While previous work [6] speculated that collisions of the compression function cannot be reflected at the output of the hash function when employing a checksum finalization step, in this section, we show how to turn the previous compression function collision to a hash function collision. On top of the modular checksum finalization step, Streebog incorporates a counter \( N \) with each compression function call. However, \( N \) is mixed with the chaining value with a simple XOR operation. It should be noted that once the constants of the compression function are fixed to some values, they remain the same for all successive executions of the compression function. Accordingly, one cannot search for a different collision with the same constants.

![Malicious Streebog collision](image)

**Fig. 5.** Malicious Streebog collision.

Our approach is to replicate the first collision two times, thus creating a 2-block multicollision structure with the same \( H_2 \) input to the padding call \( g_N(H_2, m_p) \) as depicted in Figure 5. By doing this, it is guaranteed that four messages collide at \( H_2 \), and only two of them collide at the output of the hash function. Namely, those two that have the same modular checksum which are \( M = m_1 || m_2 \) and \( M' = m_2 || m_1 \). However, using the same collision twice implies that the second collision should have a chaining input \( H_1 \) equal to that of the first collision which is \( IV = 0 \). For this, we compute a new \( C_{12} \) to enforce the output of the first collision \( H_1 \) to be 512, which is equal to the value of \( N \) used in the following compression function call. To this end, at the input of the second compression function call \( H_1 \) cancels the effect of \( N \) and the second colliding message pair has a chaining input equal to the IV which is used at the first call.

Conclusion and Discussion

In this paper, we have investigated a malicious version of Streebog. We took advantage of the heavily random looking constants and the number of rounds of the compression function to present a 2-block message pair with the same
Our approach first searches for three solutions for three different 4-round differential paths and use the freedom of five constants to connect them to produce a compression function collision. Finally, we employed the freedom in the last constant used in the round key generation to cancel the effect of the counter used in the second compression function call. Hence, we were able to append a second similar message pair, thus creating a 2² multicolliion structure where only two of them have the same modular checksum and accordingly the same digest.

It should be noted that these results has no impact on the security of the original standard. Additionally, this new set of constants does not provide collision for GOST-256 as it uses a different IV. However, they are interesting in the light of the absence of the source of the used parameters of the standard. Our results also show one of the first examples of compression function collisions being sufficient to generate hash function collisions. It is interesting to mention that, due to the versatility of the used differential path where the one byte difference can virtually be anywhere in the state, we get the freedom to satisfy the magic number as well as other constraints that are needed to produce meaningful collisions for some specific file formats (cf. section 4 in [2]). As a future direction, one may investigate the applicability of the attack if the number of rounds is not a multiple of four. Also, one might try searching for a malicious adaptation that holds for the two versions of the hash function simultaneously. Finally, we see that this paper provides an incentive for the designers of Streebog to publish the origin of the used parameters and the rational behind their choices.

<table>
<thead>
<tr>
<th>C₁</th>
<th>C₂</th>
<th>C₃</th>
<th>C₄</th>
<th>C₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>3b 7b 5d ca f1 e4 23 2f</td>
<td>4c 5f 6a 9f c9 6f 3f</td>
<td>f4 3f 4e de 2b ca 9f</td>
<td>3b 5f 5d</td>
<td>9f 3f 6a</td>
</tr>
<tr>
<td>de dd 27 78 6d 9b 1c 9f</td>
<td>42 52 38 55 1b 1c 9f</td>
<td>f4 3f 4e</td>
<td>de 2b ca</td>
<td>9f 3f 6a</td>
</tr>
<tr>
<td>1f 9f 94 dd 27 02 f3 a2</td>
<td>6e 5b 20 23 c9 f9 8f 3d</td>
<td>f4 3f 4e</td>
<td>de 2b ca</td>
<td>9f 3f 6a</td>
</tr>
<tr>
<td>4b 8e ad 06 8d 6f 3a 6f</td>
<td>a5 cc 0b e3 78 9b 9d 52</td>
<td>f4 3f 4e</td>
<td>de 2b ca</td>
<td>9f 3f 6a</td>
</tr>
<tr>
<td>fb da e2 5c b1 2a 0f 3a</td>
<td>bc 30 cc de 99 39 07 69</td>
<td>f4 3f 4e</td>
<td>de 2b ca</td>
<td>9f 3f 6a</td>
</tr>
<tr>
<td>0f 7d 6d 18 ba 6b 0c e9</td>
<td>ch 69 60 cf 89 f9 20 cd</td>
<td>f4 3f 4e</td>
<td>de 2b ca</td>
<td>9f 3f 6a</td>
</tr>
<tr>
<td>27 b7 42 a3 7d 68 cd 64</td>
<td>c7 e6 7c 81 ef d7 97 6c</td>
<td>f4 3f 4e</td>
<td>de 2b ca</td>
<td>9f 3f 6a</td>
</tr>
<tr>
<td>5b 41 e8 61 c2 cb 9d a6</td>
<td>71 ac 16 c5 bf cc 1b c1</td>
<td>f4 3f 4e</td>
<td>de 2b ca</td>
<td>9f 3f 6a</td>
</tr>
</tbody>
</table>

Table 1. The six new constants.
Table 2. The six unchanged (original) constants.

<table>
<thead>
<tr>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$\Delta m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>02 07 5d 81 b1 63 d8 cc</td>
<td>02 07 5d 81 b1 63 d8 cc</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>63 16 bb de 0e 61 85 d6</td>
<td>63 16 bb de 0e 61 85 d6</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>97 89 a3 e6 55 cf 46 c7</td>
<td>97 89 a3 e6 55 cf 46 c7</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>37 de 22 19 54 d6 01 95</td>
<td>37 de 22 19 54 d6 01 95</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>13 44 b8 4d a4 3d 46 4c</td>
<td>13 44 b8 4d a4 3d 46 4c</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>a3 50 36 27 f3 51 7f ee</td>
<td>a3 50 36 27 f3 51 7f ee</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>58 23 1d 88 80 1b 09 62</td>
<td>58 23 1d 88 80 1b 09 62</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>08 9d bc 4d aa 1f 73 2a</td>
<td>08 9d bc 4d aa 1f 73 2a</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

$H(m_1||m_2) = H(m_2||m_1)$

Table 3. Example of a 2-block message collision for the malicious Streebog hash function.

<table>
<thead>
<tr>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$\Delta m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6f a3 6b 8a a9 9d 2f 1a</td>
<td>6f a3 6b 8a a9 9d 2f 1a</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>4f c3 9d 46 0f 70 b5 d7</td>
<td>4f c3 9d 46 0f 70 b5 d7</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>6f ca e2 0e 4e 90 35 9e b1</td>
<td>6f ca e2 0e 4e 90 35 9e b1</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>9a b5 17 6b 12 d6 99 58</td>
<td>9a b5 17 6b 12 d6 99 58</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>ef fa a6 b7 1c 9a b5 04</td>
<td>ef fa a6 b7 1c 9a b5 04</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>5e b5 61 c2 db 0a 7a ca</td>
<td>5e b5 61 c2 db 0a 7a ca</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>ef a3 b5 3b 7a 30 39 c0</td>
<td>ef a3 b5 3b 7a 30 39 c0</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

References


