The Usage of Counter Revisited: Second-Preimage Attack on New Russian Standardized Hash Function

Jian Guo[1], Jérémy Jean[1], Gaëtan Leurent[2], Thomas Peyrin[1], and Lei Wang[1]

[1] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
`{GuoJian,JJean,Thomas.Peyrin,Wang.Lei}@ntu.edu.sg`

[2] INRIA, France
`Gaetan.Leurent@inria.fr`

Abstract. `Streebog` is a new Russian hash function standard. It follows the HAIFA framework as domain extension algorithm and claims to resist recent generic second-preimage attacks with long messages. However, we demonstrate in this article that the specific instantiation of the HAIFA framework used in `Streebog` makes it weak against such attacks. More precisely, we observe that `Streebog` makes a rather poor usage of the HAIFA counter input in the compression function, which allows to construct second-preimages on the full `Streebog-512` with a complexity as low as $n \times 2^{n/2}$ (namely 2^{266}) compression function evaluations for long messages. This complexity has to be compared with the expected 2^{512} computations bound that an ideal hash function should provide. Our work is a good example that one must be careful when using a design framework for which not all instances are secure. HAIFA helps designers to build a secure hash function, but one should pay attention to the way the counter is handled inside the compression function.

Key words: `Streebog`, cryptanalysis, second-preimage attack, diamond structure, expandable message, HAIFA.

1 Introduction

Hash functions are among the most fundamental primitives in modern cryptography. Informally, a cryptographic hash function maps an arbitrarily long message into a short random looking digest, which acts as the fingerprint of the original message. As for any cryptographic primitive, one expects some security properties to be fulfilled and in the case of hash functions we can point to three classical notions:

- **Collision Resistance:** it should be computationally infeasible for an adversary to find a pair of distinct messages that have the same hash digest.
- **Second-Preimage Resistance:** for any given message M, it should be computationally infeasible for an adversary to find a distinct message M' that leads to the same hash digest than M.

- **Preimage Resistance:** for any given hash digest h, it should be computationally infeasible for an adversary to find a message M that leads to the hash digest h.

By "computationally infeasible", we mean that an attacker should not be able to break that property with less than a certain number of computations that depends on n, the bit length of the hash digest. More precisely, we expect that the best attacks on a cryptographic hash function are generic attacks. In the case of an ideal hash function, one expects to find a (second)-preimage only after trying about 2^n distinct messages, and to find a collision only after trying about $2^{n/2}$ distinct messages (due to the birthday paradox).

A cryptographic hash function is commonly built by iterating a fixed input-length function called *compression function* in order to handle arbitrarily long messages, and the iteration algorithm is referred to as *domain extension*. In this article, we mainly discuss the domain extension schemes for cryptographic hash functions, and consider the compression function as an ideal component.

Generic attacks. The well-known Merkle-Damgård scheme [13, 26] has been the most popular domain extension scheme in order to build a hash function, e.g., MD5, SHA-1 and SHA-2 are built upon such design strategy. However, since 2004, several weaknesses of Merkle-Damgård scheme have been discovered. In particular, Kelsey and Schneier published a generic second-preimage attack for long messages against the Merkle-Damgård scheme [23] in 2005. The attack complexity is roughly 2^{n-k} compression function calls if the original given message is 2^k-block long, with $k \leq n/2$. Later, Andreeva et al. gave an alternative attack using a diamond structure [3]. Their attack also require 2^{n-k} compression function calls if the original given message is 2^k-block long, but only for $k \leq n/3$. On the other hand, it is applicable to a wider range of designs; in particular it can accommodate a small dithering input in the compression function. It also gives some more freedom to the adversary: as mentioned in [3], this variant allows "the attacker to leave most of the target message intact in the second preimage, or to arbitrarily choose the contents of roughly the first half of the second preimage, while leaving the remainder identical to the target message."

Therefore, regardless of how the compression function is designed, a Merkle-Damgård hash function can simply not achieve the security of 2^n with respect to second-preimage resistance. Consequently, the research community designed new domain extension schemes in order to overcome the inherent weaknesses of the original Merkle-Damgård construction. In their original second-preimage attack, Kelsey and Schneier already suggest this approach, and mention that "XORing in a monotomic counter as part of the round function would resist the attacks". Later, Biham and Dunkelman proposed the HAIFA domain extension scheme [7], which became quite popular. The main feature of HAIFA is that it adds a counter (which corresponds to the number of previously hashed message bits) as an extra input parameter to the compression function during the iteration process, in order to make each compression function call different. On the one hand, this is widely believed to provide resistance against second-preimage attacks, and this can be proved under strong randomness assumptions for the compression function [10]. On the other hand, this means the compression function must accept an extra input, which must be processed securely to avoid security issues. In particular, compression function attacks can take advantage of this input [4,9,16,19], even though the effect on the iterated function is not obvious. Recently, many new dedicated hash functions have been designed following the HAIFA framework, including some `SHA-3` candidates (`BLAKE` [5], `ECHO` [6], `Shavite-3` [8], `Shabal` [12], `Skein` [14]), as well as `Streebog`, which has been standardized by the Russian government as GOST R 34.11-2012 [27] and by IETF as RFC 6896 [20].

Our contributions. In this article, we focus on the security of `Streebog` hash function with respect to the second-preimage resistance. According to the designers, `Streebog` is based on the HAIFA framework, and is explicitly claimed to resist second-preimage attacks with long message [17,30][3].

While we are not aware of any generic second-preimage attack on the HAIFA framework, we emphasize that HAIFA acts as a *generic* framework, without explicitly specifying how the counter should be involved in the compression function computation. On the other hand, `Streebog`, as an instantiation of the HAIFA framework, has fully specified the way how the counter is used inside the compression function. This instantiation is quite provocative as the counter is simply XORed to the internal state variable of the compression function. Thus, it is necessary to evaluate whether this simple approach is sound or not (at least with respect to the second-preimage resistance). This analysis will also shed some light on the statement of Kelsey and Schneier that "XORing in a monotomic counter" is sufficient to avoid those attacks.

Unfortunately, we show in this article that `Streebog`'s method to incorporate the counter *does not strengthen its security with respect to second-preimage resistance*. More precisely, we observe that during the sequential iteration of the compression function, the counter injection at block i interacts with the counter injection at next block $i+1$. The iteration of the compression function in `Streebog` can then be transformed into an equivalent form, for which a counter-independent function is used multiple times during the hashing process. This behavior reduces to almost zero the extra security brought by the HAIFA framework over the regular Merkle-Damgård construction. Thanks to our findings, we describe two second-preimage attacks on the full `Streebog-512`. In Section 4, we give an attack using a diamond structure, similar to the attack of [3]. It requires about 2^{342} compression function evaluations for long messages with at least 2^{179} blocks. In Section 5, we give attack using an expandable message, similar to the attack of [23]. It requires only 2^{266} compression function evaluations for long messages with at least 2^{259} blocks. For short messages of 2^x blocks, the first attack gives a complexity of about $2x \cdot 2^{512-x}$ when $x < 179$, while the second attack gives a

[3] These documents also claim that Streebog is resistant to the herding attack from Kerlsey and Kohno [22], but it is well known that this attack is applicable to HAIFA if no salt is used [7].

complexity of about 2^{523-x} when $x < 259$. Note that this increases *linearly* with the decrease of the message block length (ignoring the logarithmic factor).

The rest of the article is organized as follows. In Section 2, we provide a description of the `Streebog` hash function, and then discuss our main observation on the usage of the counter value in Section 3. We detail how this observation can be used in order to mount second-preimage attacks of the full `Streebog-512` hash function in Section 4 (using a diamond structure), and in Section 5 (using an expandable message). Finally, we draw conclusions in Section 6.

2 Specifications of `Streebog`

2.1 Domain extension of `Streebog`

`Streebog`is a family of two hash functions, `Streebog-256` and `Streebog-512` that has hash output sizes 256 and 512 bits respectively [20, 27]. In this article, we only consider the large version `Streebog-512` and we simply refer to it as `Streebog`.

During the computation process, `Streebog` updates the internal state h as well as two other internal variables: Σ that denotes the checksum of the message blocks already processed, and the counter N that refers to the number of already hashed bits. Both the message block size and the intermediate hash variable size are 512 bits. The dedicated domain extension consists of three stages that we describe below (see also Figure 1). Let M be the input message, and we denote $|M|$ its bit length. In the rest of the article, we also denote h_i the internal state variable h after the i-th application of the compression function g, which is defined in more details in Section 2.2.

Stage 1. This phase initializes the hash state. The three variables Σ, N and h are assigned to 0, 0 and IV respectively, where IV refers to the initialization vector of `Streebog`, and has been publicly defined by the designers.

Stage 2. The input message M is divided into 512-bit blocks $m_1||m_2||\cdots||m_t$, where $t = \left\lceil \frac{|M|}{512} \right\rceil$. The block m_i, $1 \leq i \leq t$, is processed according to the following operations:

$$h_i \longleftarrow g(N, h_{i-1}, m_i); \qquad N \longleftarrow N + 512; \qquad \Sigma \longleftarrow \Sigma + m_i.$$

Stage 3. Pad the last block with `10`$\cdots$`0` so that it becomes full, and we denote this padded block m. Then, process this padded last block with:

$$h_{t+1} \longleftarrow g(N, h_t, m); \qquad N \longleftarrow N + (|M| \mod 512); \qquad \Sigma \longleftarrow \Sigma + m.$$

After all the message blocks have been processed, two extra compression function calls are applied:

$$h_{t+2} \longleftarrow g(0, h_{t+1}, |M|); \qquad h_{t+3} \longleftarrow g(0, h_{t+2}, \Sigma).$$

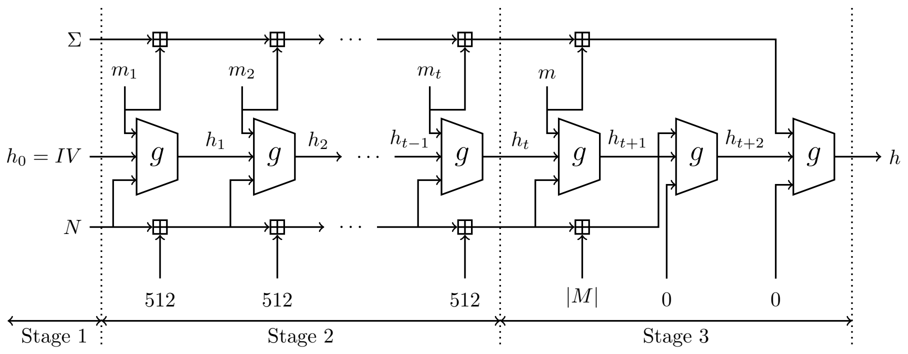


Figure 1: The domain extension algorithm of `Streebog`.

Finally, h_{t+3} is the hash digest for `Streebog-512`. In the case of `Streebog-256`, the 256 MSBs of h_{t+3} are outputted as hash digest.

2.2 The compression function of `Streebog`

As described in the introduction, the designers of `Streebog` have chosen to adopt the HAIFA model in the design of the compression function g. This framework has been initially introduced to differentiate the successive applications of the compression function calls by adding a counter as additional input parameter. Here, we mainly focus on how the counter N is used in the compression function $g(N, h_{i-1}, m_i)$, which is described in Figure 2. Particularly, we emphasize that f is a deterministic function independent of the counter N. Since the detailed algorithm of f is not related to our attack, we omit its description in this paper, and refer the interested reader to the original document [20, 27]. Yet we would like to point out that f shares high similarity with the compression function of Whirlpool hash function [28], which leads to the analysis results on `Streebog` [1, 2, 31] that share similarity with the attacks on Whirlpool [25, 29].

For the sake of simplicity, we consider that the counter value equals the number of compression calls rather than the number of processed bits. Practically, this only consists in performing a right-shift operations of 9 bit positions on the counter value. This simplification does not change any of the results described in this article, while easing the reading of the technical contents.

3 Our observation

In this section, we propose an equivalent representation of the domain extension algorithm of `Streebog`, which we use in the next section to launch a second-preimage attack on the full hash function.

First of all, we describe this equivalent description of the compression function, which is depicted in Figure 3. The counter variable N coming from the HAIFA

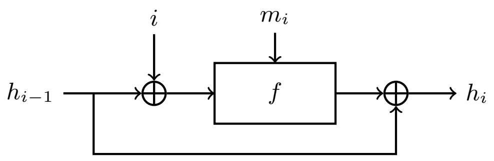


Figure 2: The compression function $g(N, h_{i-1}, m_i)$ of `Streebog` produces the new chaining variable h_i.

design is simply XORed to the internal state h_{i-1} prior to the application of the function f (but after the feed-forward branching, see Figure 2), which makes it possible to linearly shift the addition before and after the feed-forward in the original compression function. Formally, we have the following equivalence:

$$h_i = h_{i-1} \oplus f(h_{i-1} \oplus i, m_i) \qquad \Longleftrightarrow \qquad \begin{cases} h_i = F(h_{i-1} \oplus i, m_i) \oplus i, \\ F(x, m_i) = f(x, m_i) \oplus x. \end{cases}$$

Note that the counter value i is now XORed to both the input hash variable and the output hash variable of F (see Figure 3), while F itself is a deterministic function which is independent of the counter parameter i.

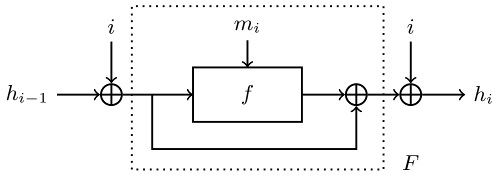


Figure 3: An equivalent representation of `Streebog`'s compression function: the internal function F has been made independent of the counter value.

We now pay attention to the sequential iteration of the above equivalent compression function in Stage 2 of the domain extension. For the sake of simplicity, we detail here the case of two consecutive blocks (see Figure 4).

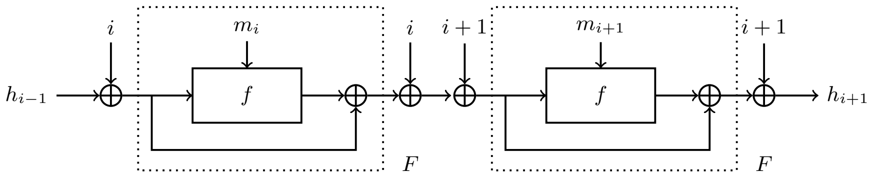


Figure 4: Two consecutive compression function calls in the equivalent representation: the counter addition in between the two calls can be combined and controlled.

As we can see, during the end of the i-th message block computation until the beginning of the $(i+1)$-th message block computation, the output of F is updated twice by XORing consecutively the counter values i and $i+1$. We define

$$\Delta(i) \overset{\text{def}}{=} i \oplus (i+1),$$
$$F_{\Delta(i)}(X,Y) \overset{\text{def}}{=} F(X,Y) \oplus \Delta(i).$$

From this observation, we get yet another equivalent representation of the consecutive compression function iterations during Stage 2 of `Streebog`, as shown in Figure 5.

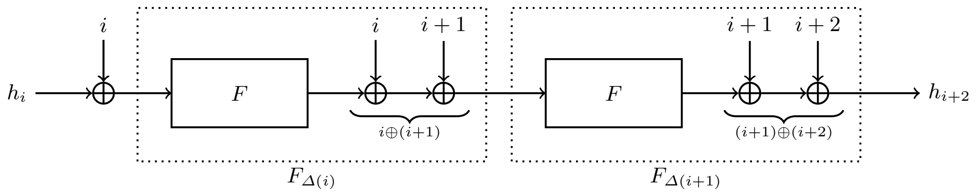


Figure 5: Two consecutive compression function blocks in the equivalent representation.

Next, we investigate the relation between the functions $F_{\Delta(i)}$, $1 \leq i \leq t$. In the most simple case, we can easily see that $\Delta(i) = i \oplus (i+1) = 1$ always holds as long as i is an even integer. Consequently, the very same function F_1 is used every even integer index during the iterations in Figure 5. We list the first values of $\Delta(i)$ in Table 1, and one can see that there is a lot of structure: sequences of length $2^s - 1$ seem to repeat every 2^s steps. More formally, we compare the functions $F_{\Delta(i)}$ and $F_{\Delta(i+2^s)}$ for any $0 \leq i < 2^s - 1$, where s can be any positive integer smaller than 512. Let $\langle \mathtt{i} \rangle$ denote the s-bit binary representation of that integer i. We have:

$$\Delta(i) = \langle \mathtt{i} \rangle \oplus \langle \mathtt{i+1} \rangle$$
$$\Delta(i+2^s) = (\mathtt{1}||\langle \mathtt{i} \rangle) \oplus (\mathtt{1}||\langle \mathtt{i+1} \rangle) = \langle \mathtt{i} \rangle \oplus \langle \mathtt{i+1} \rangle.$$

Thus, we conclude that $F_{\Delta(i)}$ and $F_{\Delta(i+2^s)}$ are the same function for any $0 \leq i < 2^s - 1$. By extending this simple reasoning, we can generalize and demonstrate that $F_{\Delta(i)}$ and $F_{\Delta(i+j\times 2^s)}$ are the same function for any $0 \leq i < 2^s - 1$ and any integer j. This is illustrated in Figure 6.

Table 1: First values of $\Delta(i)$.

i:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
$\Delta(i)$:	1	3	1	7	1	3	1	15	1	3	1	7	1	3	1	31	1	3	1	7	1	3	1	15

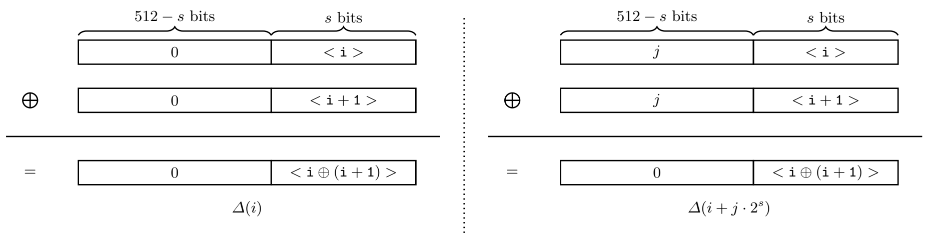


Figure 6: Functions $F_{\Delta(i)}$ and $F_{\Delta(j\times 2^s+i)}$ are the same

Finally, we present an equivalent representation of the sequential iteration in Stage 2 of the domain extension of `Streebog` in Figure 7, where F_i denotes the function for $F_{\Delta(j\times 2^s+i)}$ with $0 \le i \le 2^s - 2$, and G_j denotes the functions $F_{\Delta(j\times 2^s-1)}$, where j is any integer. Let l be $\left\lfloor \frac{t}{2^s} \right\rfloor$ and p be the reminder of $t \bmod 2^s$.

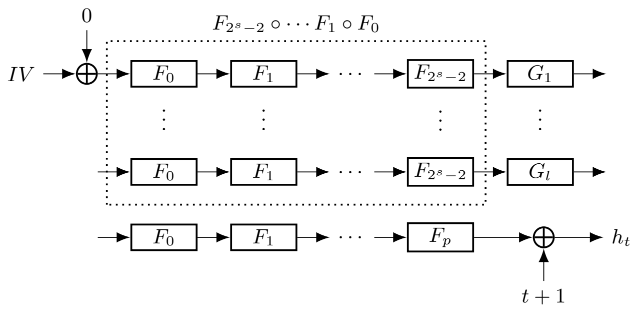


Figure 7: The equivalent representation of Stage 2

4 Second-preimage attack on full `Streebog` with a diamond

Based on the equivalent description of the Stage 2 computation of `Streebog` presented in the previous section, we now describe a second-preimage attack on the full `Streebog-512` hash function with time complexity equivalent to 2^{342} compression function evaluations for an original message of at least 2^{179} blocks.

Our main observation provides a way to remove the security benefits brought by the counter of the HAIFA design in the `Streebog` hash function. This is due to a poor usage of this counter, which allows an adversary to reuse previously known second-preimage techniques on the classical Merkle-Darmgård construction. In particular, we can use the *diamond structure* introduced by Kelsey and Kohno [22] on the function $F_{2^s-2} \circ \cdots F_1 \circ F_0$, which is reused several times. Indeed, this technique allows to construct a large multicollision set of 2^d d-block messages, all hashing to a single chaining variable $h_\diamond$. This is similar to the second-preimage attack on dithered hash functions by Andreeva et al. [3].

We first give in Section 4.1 a detailed explanation concerning the construction of this structure with $2^{(n+d)/2}$ computations, and we later describe in Section 4.2 how to use it inside a second-preimage attack for the full `Streebog-512`.

4.1 The diamond structure

As depicted in Figure 8, a 2^d-diamond construction refers to a complete binary tree of depth d, i.e., the distance from the leaves to the root is d. There are exactly 2^{d-l} nodes at level l, for $0 \leq l \leq d$, where $l = 0$ refers to the leaf level and $l = d$ to the root level. All nodes except the leaves have two children from lower level. In [22], Kelsey and Kohno introduced this structure to launch herding attacks. In this diamond, a node refers to a chaining value, and an edge represents a message connecting one chaining value to another.

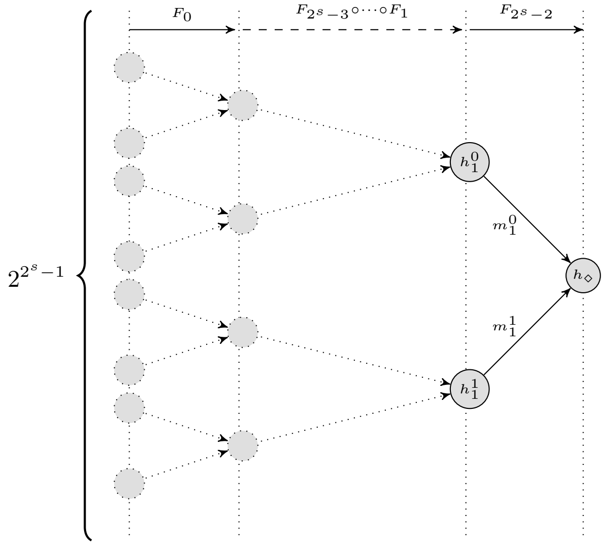


Figure 8: The diamond structure of depth $2^s - 1$ used in our second-preimage attack.

Given the leaves, i.e., 2^d chaining values at level 0, one can construct the diamond in $2^{(n+d)/2}$ compression function evaluations. The construction algorithm was initially proposed by Kelsey and Kohno [22] and later refined by Kortelainen and Kortelainen [24] and verified in [18]. The algorithm works level by level recursively and independently. Below in Algorithm 1, we show how the next level of 2^{d-1} chaining values are computed given the current level of 2^d nodes and compression function $f = F_0$ as input. The output L_{out} of the current level is then fed into the algorithm as input L_{in} for next level, until root is reached. The overall complexity has been estimated as $2^{(n+d)/2}$ in [24].

Algorithm 1 Construction of one level of a diamond

```
Input: input chaining value list $L_{in}$ of size $2^d$
Input: compression function $f$
Output: next layer chaining values list $L_{out}$ of size $2^{d-1}$
 1: initialize an empty hash table $T$, a message list $L_M$.
 2: while $L_{in}$ is not empty do
 3:     pick random message block $M$, add to $L_M$.
 4:     for all $h_{in} \in L_{in}$ do
 5:         evaluate $h_{out} = f(h_{in}, M)$
 6:         if $T[h_{out}]$ is not empty then
 7:             fetch the pair $(h'_{in}, M')$ in entry $T[h_{out}]$
 8:             add $h_{out}$ to $L_{out}$, along with $(h_{in}, M)$, $(h'_{in}, M')$ as the connecting edges.
 9:             remove $f(h_{in}, m)$ and $f(h'_{in}, m)$ from $T$ for all $m \in L_M$.
10:             remove $h_{in}$ and $h'_{in}$ from $L_{in}$.
11:         else
12:             add $(h_{in}, M)$ to $T[h_{out}]$.
```

4.2 Details of the attack

At this point, we are able to build a diamond structure, and we would like to use it for a second-preimage attack. An overview of our attack is given in Figure 9, where one can see that we use $d = 2^s - 1$ in order to fully control the effect of the counter. The diamond structure is constructed with the function $F_{2^s-1} \circ \cdots F_1 \circ F_0$. Then, as in the original second-preimage attack using the diamond structure, we use a single message block $m_{\diamond}^{\searrow}$ to connect the root chaining value $h_{\diamond}$ to the known message we are attacking. The connection is done after the next F function, but before the addition of the counter, i.e. we match of set of values $\{F(h_{\diamond}, m) \,|\, m \leftarrow \$\}$, and $\{h_i \oplus i \,|\, i \equiv 0 \mod 2^s\}$. If the original message m consists of t 2^s-bit blocks, we have $l = \left\lfloor \frac{t}{2^s} \right\rfloor$ possible connecting points, meaning that we expect to pick about $2^n/l$ random message blocks $m_{\diamond}^{\searrow}$ before hitting a known point $h'_{\diamond}$.

This point of connection gives the value $l' \times 2^s - 1$ of the counter N used in `Streebog` at that position. Once we have found the 1-block connecting message $m_{\diamond}^{\searrow}$ at the end of the diamond structure, we need to connect one of the 2^d leaves of the diamond structure to the IV of the hash function.

Before finding a valid second-preimage, there are two additional points that we need to consider. First, the second-preimage needs to have the exact same length $|M|$ as the first message since `Streebog` processes the length of the message at the end of the hashing process. Second, the additive checksum computed over the new blocks of the second-preimage needs to match the targeted one Σ of the original message.

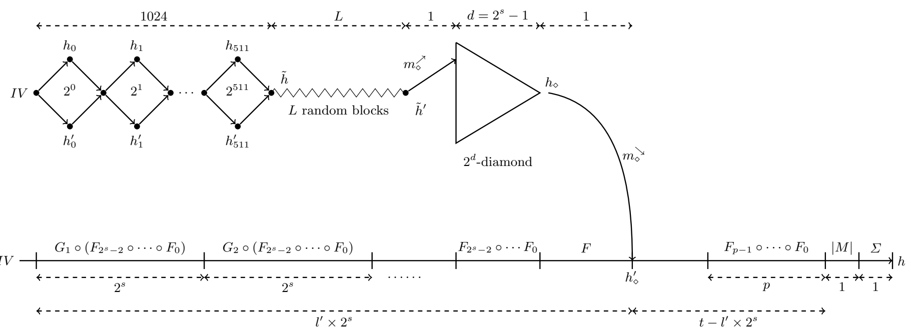


Figure 9: Overview of the second-preimage attack.

In order to overcome both of these two points, we first construct a 2^{512}-multicollision (with a technique similar to the one from Joux [21]) over the first $2 \times 512 = 1024$ message blocks so as to handle the checksum issue. This step can be performed efficiently with $512 \times 2^{n/2}$ computations using the technique described in [15]. The idea is that, at each step i of the multicollision search, we compute two sets of 2-block messages: $\{(A_i)||(-A_i)\}$, for $2^{n/2}$ random choices of A_i, and $\{(B_i + 2^i)||(-B_i)\}$, for $2^{n/2}$ random choices of B_i, in order to find a collision between the two sets.

Then, starting from the IV, we reach a chaining value $\tilde{h}$, such that we can find a 1024-block message that verifies any given additive checksum value σ. Indeed, the binary decomposition of σ gives precisely the path to follow (and incidentally the message blocks to use) in the multicollision graph we just built in order to reach σ.

We would like now to match the correct message length $|M|$. For that task, we first evaluate the number of blocks already fixed by the attack. The diamond uses $d = 2^s - 1$ blocks, the multicollision uses 1024 blocks, and we use one block for $m_\diamond^{\nearrow}$ to connect to $h'_\diamond$ in the original message chain. After the collision on $h'_\diamond$, we use the same values as in the original message, such that we want to use exactly $l' \times 2^s$ blocks between the IV and $h'_\diamond$. We use an additional message block $m_\diamond^{\nearrow}$ to connect to one leaf of the diamond, so that in total there are $L = l' \times 2^s - 1024 - 1 - 2^s - 1$ blocks left between $\tilde{h}$ and $\tilde{h}'$. We pick random values for all those blocks, obtain the value of $\tilde{h}'$, and then pick about 2^{n-d} random blocks $m_\diamond^{\nearrow}$ to hit one of the 2^d leaves of the diamond.

Finally, we compute the reduced checksum value σ of all the message blocks except the 1024 first ones, and we choose the correct 1024 message blocks in the graph so as to match the local checksum $\Sigma - \sigma$. At this point, the attack is over: all the message blocks are fixed, and the second-preimage is constructed.

Overall, the total complexity of this attack requires $2^{(n+d)/2}$ computations to construct the diamond, $2^n/l$ computations to connect the root of the diamond to the original message chain, and $512 \times 2^{n/2}$ computations for the Joux's multicollision. The time complexity

$$512 \times 2^{n/2} + 2^{n-d} + 2^{(n+d)/2} + 2^{n-\log_2(l)}$$

can be minimized by fixing $d = n/3$ and $l \geq 2^{n/3}$, which reaches an overall time complexity of about $2^{2n/3}$ computations for the second-preimage attack. With the parameters of `Streebog-512`, $n = 512$ gives the integer value $s = 8$ and $d = n/3$, and a total time complexity equivalent to about 2^{342} compression function evaluations. We note that our attack imposes a certain length on the original message as $n - \log_2(l) \leq 341$ imposes $l \geq 2^{171}$, which constraints M to have at least $2^{171+8} = 2^{179}$ message blocks.

For shorter messages with 2^x blocks and $x < 179$, the complexity is mainly dominated by the complexity of linking IV to one leaf node of the diamond structure, which is 2^{n-d}, and the complexity of linking $h_\diamond$ to $h'_\diamond$, which is $2^{n-x+\lceil \log_2(d) \rceil}$. Let $x = d$, and we get the complexity is upper bounded by $2x \cdot 2^{n-x}$. Thus the complexity increases *linearly* with the decrease of the message block length (ignoring logarithmic factors).

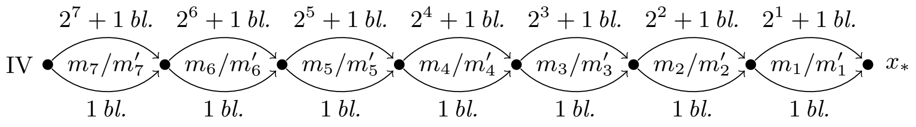
5 Second-preimage attack on full `Streebog` with an expandable message

The equivalent description of `Streebog` given in the previous sections can also be used to mount a variant of the attack of Kelsey and Schneier using an *expandable message* [23]. This gives a second-preimage attack on the full `Streebog-512` hash function with time complexity equivalent to 2^{266} compression function calls for an original message of at least 2^{259} blocks.

We first give in Section 5.1 a detailed explanation concerning the construction of this structure with $n/2 \times 2^{n/2}$ computations, and we later describe in Section 5.2 how to use it inside a second-preimage attack for the full `Streebog-512`.

5.1 The expandable message

In order to build an expandable message, we use the technique of [23], i.e. we build a multicollision where the messages in each colliding pair have a different length, as shown by Algorithm 2. If we have colliding pairs with length $(1, 2^k + 1)$, for $0 \leq k < t$, this implicitly defines a set of 2^t messages with length in the range $[t, 2^t + t - 1]$, that all reach the same final chaining value x_*. More precisely, one can build a message of length $t + L$ using the binary expression of L to select a message in each pair.



In a second-preimage attack, we hash random blocks starting from x_* until we find a link to one of the intermediate values reached when hashing the challenge message. This gives the required length for the expandable message, and we build the second preimage using the expandable message, the linking block, and the end of the challenge message.

Algorithm 2 Construction of an expandable message (Merkle-Damgård)

```
Input: Initial chaining value $x$
Input: Compression function $f$
Output: Message pairs $(m_i, m'_i)$, final chaining value $x$
 1: for $0 \leq i < n/2$ do
 2:     Initialize an empty hash table $T$
 3:     for $0 \leq r < 2^{n/2}$ do
 4:         $T[f(x, r)] \leftarrow r$
 5:     $y \leftarrow x$
 6:     for $0 \leq j < 2^i$ do
 7:         $y \leftarrow f(y, 0)$
 8:     repeat $r \leftarrow \$$
 9:     until $T[f(y, r)]$ not empty
10:     $m_i \leftarrow [0]^{2^i} \| r$
11:     $m'_i \leftarrow T[f(y, r)]$
12:     $x \leftarrow f(y, r)$
```

However, this does not work for a HAIFA compression function: depending on which message is selected in the pair k (m_k or m'_k), the message length before the following block will be different, and the counter will have a different value. Therefore, the collision (m_{k-1}, m'_{k-1}) will only be valid in one case.

In the case, of `Streebog`, the weak use of the counter makes this attack still possible thanks to the equivalent representation of Section 3. Indeed, the sequence $\Delta(i)$ has a lot of regularity and repetitions (as seen in Table 1), and with a careful construction, we can ensure that the message pairs (m_i, m'_i) are only used at positions with same sequences of $\Delta(i)$. More precisely, we must build pairs with large difference first, and use differences that are powers of two, while more general constructions can be used for plain Merkle-Damgård. We must also stop the construction a few steps before reaching a difference of 1 (as explained later, the smallest difference is $O(n)$). This means that we can only use a fraction of the intermediate states reached by the challenge message.

In the following, we call an expandable message that can reach lengths between a and b by increment of c an (a, b, c)-expandable message. Let us assume we have built an $(l, l + L, 2^i)$-expandable message for `Streebog`, with $l < 2^{i-1} - 1$. Since $l < 2^i - 1$, we have $\Delta(l+x) = \Delta(l+x+j \cdot 2^i)$, for all $0 \leq x < 2^i - l - 1$ and $j \geq 0$. In particular, if we append a new message pair (m, m') with $|m| = 2^{i-1} + 1$, $|m'| = 1$ to the expandable message, the sequence of $\Delta(i)$ used for the messages will be same for every choice of the $(l, l+L, 2^i)$-expandable message. This allows to extend the $(l, l+L, 2^i)$-expandable message into a $(l+1, l+L+1+2^{i-1}, 2^{i-1})$-expandable message. If we iterate this construction, starting from a single message of length l and a maximal increment of 2^t, we can build a $(l+t-s, l+t-s+2^{t+1}-2^s, 2^s)$-expandable message for Streebog, assuming that $l+t-s < 2^s - 1$.

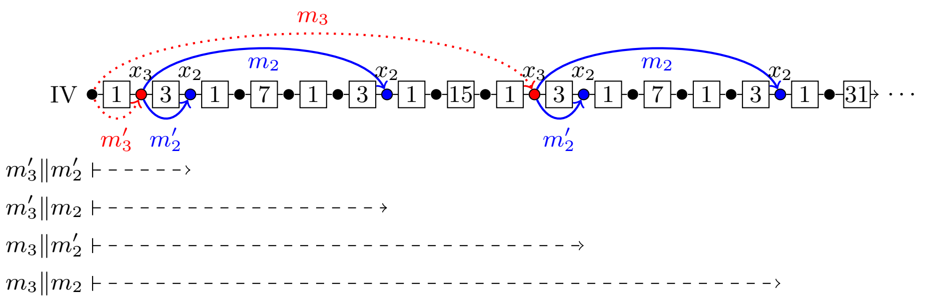


Figure 10: Construction of a $(2, 14, 4)$-expandable message for Streebog. Note that m_2 and m_2' have the same Δ indices in both positions, and the Δ for the block after $m_3' \| m_2'$, $m_3' \| m_2$, $m_3 \| m_2'$, or $m_3 \| m_2$ is the same (here, $\Delta = 1$).

5.2 Details of the attack

The second preimage attack on full Streebog-512 uses an initial multicollision with 1024 blocks in order to adjust the checksum, like the attack of Section 4. Then, we build the expandable message starting for the final value of the multicollision. With the parameters of Streebog-512, we use $l = 1024$, $s = 11$, $t = 258$, i.e. we build a $(1271, 2^{259} - 777, 2048)$-expandable message. After building the expandable message, the attack mostly follows the procedure given by Kelsey and Schenier. An overview of our attack is given in Figure 11.

We first use a message block m_* to connect the final chaining value h_* to the known message we are attacking. More precisely, if the original message m consists of t 2^s-bit blocks, we have $l = \left\lfloor \frac{t}{2^s} \right\rfloor$ possible connecting points, meaning that we expect to pick about $2^n/l$ random message blocks m_* before hitting a known point h'_*. With the parameters used for Streebog-512, we use connecting points[4] with $i \equiv 1272 \mod 2048$. This point of connection gives the value of the counter N used in Streebog at that position, and the length $L = N - 1024 - 1$ required for the expandable message. In order to build the second preimage, we select the message with the correct length L in the expandable message, and we select a message in the initial multicollision to adjust the checksum.

Overall, the attack requires about $512 \times 2^{n/2}$ computations for the Joux's multicollision, $256 \times 2^{n/2}$ for the expandable message, and $2^n/l$ computations to

[4] This correspond to the set of positions such that $i+1$ can be reached by a $(1271, 2^{259} - 777, 2048)$-expandable message

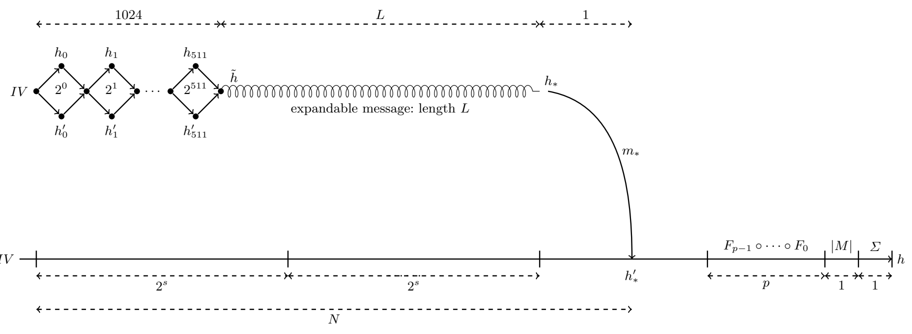


Figure 11: Overview of the second-preimage attack.

connect the end of the expandable message to the original message chain. The time complexity

$$768 \times 2^{n/2} + 2^n/l$$

can be minimized with $l > 2^{n/2}/n$, and reaches an overall time complexity in the order of $n \cdot 2^{n/2}$ computations for the second-preimage attack. With the parameters of `Streebog-512`, we have $n = 512$ and $s = 11$, and a total time complexity equivalent to about 2^{266} compression function evaluations, if the message has more than 2^{259} blocks (so that $2^n/l \leq 256 \times 2^{n/2}$).

6 Open discussion and conclusion

In this article, we have studied the security of the Russian hash function standard `Streebog`. We showed that an attacker can find second-preimages much faster than what is expected from an ideal hash function, even though `Streebog` uses HAIFA as the domain extension algorithm. Our main observation is that the counter is not very well handled in `Streebog` and this enables the attacker to apply a more complex variation of the now classical generic second-preimage attacks. As a result, `Streebog` is only marginally stronger than a plain Merkle-Damgård iteration.

This analysis also contradicts the remark by Kelsey and Schneier that "XOR-ing in a monotomic counter" would be sufficient to avoid the second-preimage attacks with long messages: there is at least one way to XOR the counter that do not provide any extra security.

Our work is a good example why one should be careful when using a design framework: problems might arise if bad instances in that framework exist. In the particular case of HAIFA, it is crucial to make sure the counter is properly handled. We have the intuition that the security property that a compression function in HAIFA has to follow with regards to the counter input is quite strong (even if the counter might controlled by the adversary, he must not be able to distinguish the output). Clearly, `Streebog` would not meet that criteria (inserting a difference δ in both the counter and the chaining variable input, one always get δ on the output). It would be interesting to study what is exactly the minimal security assumption that is required on the counter input for HAIFA in order to ensure only secure instances.

7 Acknowledgment

We would like to thank the anonymous reviewers for their detailed feedback and comments. Jian Guo, Jérémy Jean, Thomas Peyrin and Lei Wang were supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. AlTawy, R., Kircanski, A., Youssef, A.M.: Rebound attacks on Stribog. IACR Cryptology ePrint Archive **2013** (2013) 539
2. AlTawy, R., Youssef, A.M.: Preimage Attacks on Reduced-Round Stribog. In Pointcheval, D., Vergnaud, D., eds.: Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings. Volume 8469 of Lecture Notes in Computer Science., Springer (2014) 109–125
3. Andreeva, E., Bouillaguet, C., Fouque, P.A., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: Second Preimage Attacks on Dithered Hash Functions. In Smart, N.P., ed.: EUROCRYPT 2008. Volume 4965 of LNCS., Springer (April 2008) 270–288
4. Aumasson, J.P., Guo, J., Knellwolf, S., Matusiewicz, K., Meier, W.: Differential and Invertibility Properties of BLAKE. In Hong, S., Iwata, T., eds.: FSE. Volume 6147 of LNCS., Springer (2010) 318–332
5. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal BLAKE. Submission to NIST (Round 3) (2010)
6. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 Proposal: ECHO. Submission to NIST (updated) (2009)
7. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278 (2007)
8. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function. Submission to NIST (Round 2) (2009)
9. Biryukov, A., Gauravaram, P., Guo, J., Khovratovich, D., Ling, S., Matusiewicz, K., Nikolic, I., Pieprzyk, J., Wang, H.: Cryptanalysis of the LAKE Hash Family. In Dunkelman, O., ed.: FSE. Volume 5665 of LNCS., Springer (2009) 156–179
10. Bouillaguet, C., Fouque, P.A.: Practical Hash Functions Constructions Resistant to Generic Second Preimage Attacks Beyond the Birthday Bound. Submitted to Information Processing Letters (2010)
11. Brassard, G., ed.: CRYPTO'89. In Brassard, G., ed.: CRYPTO'89. Volume 435 of LNCS., Springer (August 1989)
12. Bresson, E., Canteaut, A., Chevallier-Mames, B., Clavier, C., Fuhr, T., Gouget, A., Icart, T., Misarsky, J.F., Naya-Plasencia, M., Paillier, P., Pornin, T., Reinhard, J.R., Thuillet, C., Videau, M.: Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition. Submission to NIST (2008)

13. Damgård, I.: A Design Principle for Hash Functions. [11] 416–427
14. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to NIST (Round 3) (2010)
15. Gauravaram, P., Kelsey, J.: Linear-XOR and Additive Checksums Don't Protect Damgård-Merkle Hashes from Generic Attacks. In Malkin, T., ed.: CT-RSA 2008. Volume 4964 of LNCS., Springer (April 2008) 36–51
16. Gauravaram, P., Leurent, G., Mendel, F., Naya-Plasencia, M., Peyrin, T., Rechberger, C., Schläffer, M.: Cryptanalysis of the 10-Round Hash and Full Compression Function of SHAvite-3-512. In Bernstein, D.J., Lange, T., eds.: AFRICACRYPT. Volume 6055 of Lecture Notes in Computer Science., Springer (2010) 419–436
17. Grebnev, S., Dmukh, A., Dygin, D., Matyukhin, D., Rudskoy, V., Shishkin, V.: Asymmetrical Reply to SHA-3: Russian Hash Function Draft Standard. CTCrypt 2012, abstract available from http://agora.guru.ru/csr2012/files/6.pdf
18. Guo, J.: A program confirmation of the diamond construction by Kortelainen and Kortelainen. Available online: http://guo.crypto.sg/diamond.zip (Feburary 2014)
19. Guo, J., Karpman, P., Nikolic, I., Wang, L., Wu, S.: Analysis of BLAKE2. In Benaloh, J., ed.: CT-RSA. Volume 8366 of LNCS., Springer (2014) 402–423
20. IETF: GOST R 34.11-2012: Hash Function. RFC6896 (2013)
21. Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Franklin, M., ed.: CRYPTO 2004. Volume 3152 of LNCS., Springer (August 2004) 306–316
22. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In Vaudenay, S., ed.: EUROCRYPT 2006. Volume 4004 of LNCS., Springer (May / June 2006) 183–200
23. Kelsey, J., Schneier, B.: Second Preimages on n-Bit Hash Functions for Much Less than $2n$ Work. In Cramer, R., ed.: EUROCRYPT 2005. Volume 3494 of LNCS., Springer (May 2005) 474–490
24. Kortelainen, T., Kortelainen, J.: On Diamond Structures and Trojan Message Attacks. In Sako, K., Sarkar, P., eds.: ASIACRYPT (2). Volume 8270 of Lecture Notes in Computer Science., Springer (2013) 524–539
25. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Matsui, M., ed.: ASIACRYPT 2009. Volume 5912 of LNCS., Springer (December 2009) 126–143
26. Merkle, R.C.: One Way Hash Functions and DES. [11] 428–446
27. REGULATION, F.A.O.T., METROLOGY: Information technology - CRYPTOGRAPHIC DATA SECURITY - Hash-function. GOST R 34.11-2012 (2012)
28. Rijmen, V., Barreto, P.S.L.M.: The WHIRLPOOL Hashing Function. Submitted to NISSIE (September 2000)
29. Sasaki, Y., Wang, L., Wu, S., Wu, W.: Investigating Fundamental Security Requirements on Whirlpool: Improved Preimage and Collision Attacks. In Wang, X., Sako, K., eds.: ASIACRYPT 2012. Volume 7658 of LNCS., Springer (December 2012) 562–579
30. https://www.streebog.net/: GOST R 34.11-2012: Streebog Hash Function
31. Wang, Z., Yu, H., Wang, X.: Cryptanalysis of GOST R Hash Function. Cryptology ePrint Archive, Report 2013/584 (2013) http://eprint.iacr.org/.