Streebog as a Random Oracle

Liliya Akhmetzyanova, Alexandra Babueva and Andrey Bozhko

CryptoPro LLC, Moscow, Russia
{lah, babueva, bozhko}@cryptopro.ru

Abstract

The random oracle model is an instrument used for proving that protocol has no structural flaws when settling with standard hash properties is impossible or fairly difficult. In practice, however, random oracles have to be instantiated with some specific hash functions, which are not random oracles. Hence, in the real world, an adversary has broader capabilities than considered in the random oracle proof — it can exploit the peculiarities of a specific hash function to achieve its goal. In a case when a hash function is based on some building block, one can go further and show that even if the adversary has access to that building block, the hash function still behaves like a random oracle under some assumptions made about the building block. Thereby, the protocol can be proved secure against more powerful adversaries under less complex assumptions. The indifferentiability notion formalizes that approach.

In this paper we study whether Streebog, a Russian standardized hash function, can instantiate a random oracle from that point of view. We prove that Streebog is indifferentiable from a random oracle under an ideal cipher assumption for the underlying block cipher.

Keywords: Streebog, GOST, Random Oracle, Indifferentiability

1 Introduction

The random oracle model, introduced by Bellare and Rogaway in [9], assumes that every party of the protocol and an adversary has access to a random oracle, which is used instead of a hash function. A random oracle [9] is an ideal primitive which models a random function. It provides a random output for each new query, and identical input queries are given the same answer. The random oracle model allows proving that the protocol does not have any structural flaws in situations when it is impossible or fairly difficult to settle with standard hash properties, which is the case for many efficient and elegant solutions. For example, such protocols and mechanisms as TLS [3], IPSec [2], and Schnorr signature [16, 15] were analyzed in the random oracle model; Russian standardized versions of TLS [4] and IPSec [6], as well as SESPAKE protocol [5, 8], shortened ElGamal signature [7], to-be-standardized RSBS blind signature [18], and postquantum Shipovnik signature [19] are also analyzed in the random oracle model.

In practice, however, being idealized primitives, random oracles do not exist and have to be instantiated with some specific hash functions, which are not random oracles. Hence,

in the real world, an adversary has broader capabilities than considered in the random oracle proof — it can exploit the peculiarities of a specific hash function to achieve its goal. To address such a situation, one can go further and consider the design of the hash function to show that, under some less complex and more specific assumptions than the whole function being a random oracle, it behaves like a random oracle. To do that, one must first understand what "behaves like a random oracle" mean and what assumptions to make.

These questions for a particular class of hash functions are addressed by Coron et al. in [10, 11]. They study the case when an arbitrary-length hash function is built from some fixed-length building block (like an underlying compression function or a block cipher). They come up with a definition, based on the indifferentiability notion of Maurer et al. [14], of what it means to implement a random oracle with such construction, in the assumption that the building block itself is an ideal primitive. The definition is chosen in a way that any hash function satisfying it can securely instantiate a random oracle in a higher-level application (under the assumption that the building block is an ideal primitive). Hence, idealized assumptions are made about less complex and lower-level primitive, and, as a result, more adversarial capabilities are accounted for.

In this paper we study whether Streebog, a Russian standardized hash function [1], can instantiate a random oracle. We recall that Streebog has always been a popular target for analysis. An overview of the results which study standard properties of the algorithm can be found in [17]. A recent paper [13] by Kiryukhin studies keyed version of Streebog as a secure pseudorandom function in a related-key resilient PRF model for an underlying block cipher, highlighting some important high-level design features of Streebog.

Since Streebog is a modified Merkle-Damgard construction based on LSX-style block cipher in Miyaguchi-Preneel mode, we adopt the notion of Coron et al. The paper's main result is presented in Section 3 – we prove that Streebog is indifferentiable from a random oracle under an ideal cipher assumption for the underlying block cipher. We benefit greatly from the work done in [10, 11] since their analysis is focused on Merkle-Damgard constructions with a block cipher in Davis-Meyer mode. However, Streebog's design features and a different structure of compression function do not allow us to use the paper's results and provoke several challenges.

2 Definitions

Let $|a|$ be the bit length of the string $a \in \{0,1\}^*$, the length of an empty string is equal to 0. For a bit string a we denote by $|a|_n = \lceil |a|/n \rceil$ the length of the string a in n-bit blocks. Let 0^u be the string consisting of u zeroes.

For a string $a \in \{0,1\}^*$ and a positive integer $l \leqslant |a|$ let $\mathrm{msb}_\ell(a)$ be the string, consisting of the leftmost l bits of a. For nonnegative integers l and i let $\mathrm{str}_l(i)$ be l-bit representation of i with the least significant bit on the right, let $\mathrm{int}(M)$ be an integer i such that $\mathrm{str}_l(i) = M$. For bit strings $a \in \{0,1\}^{\leqslant n}$ and $b \in \{0,1\}^{\leqslant n}$ we denote by $a + b$ a string $\mathrm{str}_n(\mathrm{int}(a) + \mathrm{int}(b) \bmod 2^n)$. If the value s is chosen from a set S uniformly at random, then we denote $s \xleftarrow{\mathcal{U}} S$.

A block cipher E with a block size n and a key size k is the permutation family $\big(E_K \in Perm(\{0,1\}^n) \mid K \in \{0,1\}^k\big)$, where K is a key.

2.1 Streebog hash function

The Streebog hash function is defined in [1]. For the purposes of the paper we will define Streebog as a modification of Merkle-Damgard construction, which is applied to a prefix-free encoding of the message; in that we follow the approach of [10, 11]. We will also make the use of the equivalent representation of Streebog from [12]. For Streebog the length of an internal state in Merkle-Damgard construction is $n = 512$ and the length of the output k is either 256 or 512.

Let us define a compression function $h: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, which is based on 12-rounds LSX-like block cipher $E: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, where the first argument is a key, in Miyaguchi-Preneel mode:

$$h(y,x) = E(y,x) \oplus x \oplus y.$$

We also define a prefix-free encoding $g: \{0,1\}^* \to (\{0,1\}^n, \{0,1\}^n)^*$, which takes as an input a message X:

$$g(X) = (x_1, \Delta_1) \| (x_2, \Delta_2) \| \dots \| (x'_l \| 10^{n-1-|x'_l|}, \tilde{\Delta}_l) \| (L, 0) \| (\Sigma, 0),$$

where $L = |X|$, $l = \lfloor L/n \rfloor + 1$, $X = x_1 \| \dots \| x'_l$, where $x_1, \dots, x_{l-1} \in \{0,1\}^n$, $x'_l \in \{0,1\}^{<n}$ and x'_l is an empty string if L is already divisible by n; $\Delta_i = \mathrm{str}_n(i \cdot n) \oplus \mathrm{str}_n((i-1) \cdot n)$, $\tilde{\Delta}_i = \mathrm{str}_n((i-1) \cdot n)$ and $\Sigma = \sum_{i=1}^{l-1} x_i + (x'_l \| 10^{n-1-|x'_l|})$. The encoding pads the message with $10^{n-1-|x'_l|}$, then it splits the message in blocks of length n, computes the counter value for each block and appends two last blocks of the encoding, the bit length L and the checksum Σ, which correspond to the finalizing step of Streebog.

Finally, we define the hash function Streebog on Figure 1, where IV, $|IV| = 512$ is a predefined constant, different for $k = 256$ and $k = 512$. On Figure 2 Streebog is depicted schematically.

Streebog(X)

$l \leftarrow \lfloor |X|/n \rfloor + 1$
$(x_1, c_1) \| (x_2, c_2) \| \dots \| (x_l, c_l) \| (x_{l+1}, c_{l+1}) \| (x_{l+2}, c_{l+2}) \leftarrow g(X)$
$y_1 \leftarrow IV$
for $i = 1 \dots l+2$ **do** :
  $y_{i+1} \leftarrow h(y_i, x_i) \oplus c_i$
return $\mathrm{msb}_k(y_{l+3})$

Figure 1: Streebog hash function

We will call a sequence of triples $(y_1, x_1, z_1), (y_2, x_2, z_2), \dots, (y_{l+2}, x_{l+2}, z_{l+2})$, where $z_i = h(y_i, x_i) \oplus y_i \oplus x_i$, which appears during a computation of Streebog on an input X, a *computational chain* for X.

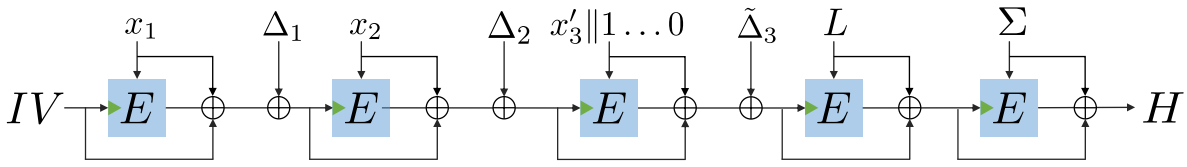

Figure 2: Streebog computation, $l = 3$

2.2 Indifferentiability

The following strategy is often applied to prove the security of a cryptosystem with some component (or primitive). One first proves that the system is secure in case of using idealized primitive. Secondly, one proves that the real primitive is indistinguishable from an idealized one. Informally, two algorithms A and B are computationally indistinguishable if no (efficient) algorithm $\mathcal{D}$ is able to distinguish whether it is interacting with A or B.

In the current paper we consider two types of the ideal primitives: random oracles and ideal ciphers. A random oracle [9] is an ideal primitive which models a random function. It provides a random output for each new query, identical input queries are given the same answer. An ideal cipher is an ideal primitive that models a random block-cipher $\mathcal{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$, each key $K \in \{0,1\}^\kappa$ defines a random permutation on $\{0,1\}^n$. The ideal cipher provides oracle access to $\mathcal{E}$ and $\mathcal{E}^{-1}$; that is, on query $(+, K, x)$, it answers $c = E(K, x)$, and on query $(-, K, c)$, it answers x such that $c = E(K, x)$.

Obviously, a random oracle (ideal cipher) is easily distinguishable from a hash function (block cipher) if one knows its program and the public parameter. Thus in [14] the extended notion of indistinguishability — *indifferentiability* — was introduced. It was proven, that if a component A is indifferentiable from B, then the security of any cryptosystem $C(A)$ based on A is not affected when replacing A by B. According to the authors, indifferentiability is the weakest possible property allowing for security proofs of the generic type described above. Thus, to prove the security of some cryptosystem using hash function we may prove its security in the random oracle model and then prove that hash function is indifferentiable from a random oracle within some underlying assumptions. In the current paper we assume that the base block cipher is modelled as an ideal cipher.

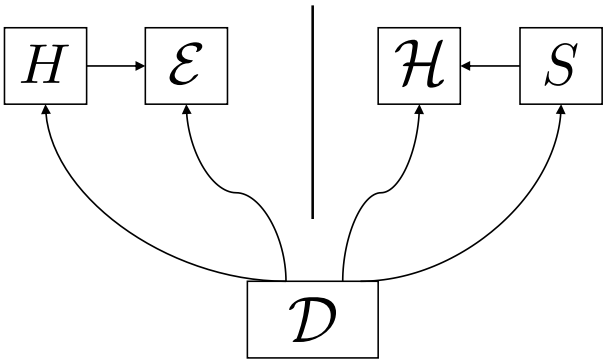

Figure 3: The indifferentiability of hash function H and random oracle $\mathcal{H}$

Let us formally define what does the indifferentiability from an ideal primitive mean. We will give the definition directly for the hash function (based on the ideal cipher) and random oracle. This definition is a particular case of more general indifferentiability notion introduced in [14].

Definition 1. *A hash function H with oracle access to an ideal cipher $\mathcal{E}$ is said to be $(T_\mathcal{D}, q_H, q_E, \varepsilon)$-indifferentiable from a random oracle $\mathcal{H}$ if there exists a simulator S, such that for any distinguisher $\mathcal{D}$ with binary output it holds that:*

$$\left|\Pr\left[\mathcal{D}^{H,\mathcal{E}} \to 1\right] - \Pr\left[\mathcal{D}^{\mathcal{H},S} \to 1\right]\right| < \varepsilon.$$

The simulator has oracle access to $\mathcal{H}$. The distinguisher runs in time at most $T_{\mathcal{D}}$ and makes at most q_H and q_E queries to its oracles.

The indifferentability notion is illustrated at Figure 3. The distinguisher interacts with two oracles, further we denote them by left and right oracles respectively. In the one world left oracle implements the hash function H (with oracle access to the ideal cipher), while the right oracle directly implements the ideal cipher $\mathcal{E}$. In another world the left oracle implements the random oracle $\mathcal{H}$ and the right oracle is implemented by the simulator S. The task of the simulator is to model the ideal cipher using the oracle access to $\mathcal{H}$ so that no distinguisher could notice the difference. To achieve that, the output of S should be consistent with what the distinguisher can obtain from $\mathcal{H}$. Note that the simulator does not have access to the queries of the distinguisher to $\mathcal{H}$.

3 Streebog indifferentiability

In this section we introduce the main result of the paper, which demonstrates that Streebog is indifferentiable from a random oracle in the ideal cipher model for the base block cipher.

At first, we discuss the choice of the underlying assumption. Indeed, the straightforward solution is to prove Streebog indifferentiability in assumption that the compression function is a random oracle. Although such proof may be constructed much easier than in the ideal cipher model, we show that the Miyaguchi-Preneel compression function cannot be modeled as a random oracle. Indeed, for this function the following condition always holds:

$$x = E^{-1}(y, h(y, x) \oplus x \oplus y).$$

Thus, the distinguisher can easily identify whether it interacts with the real compression function or the random one by making the query (y, x) to the left oracle and the query $(-, y, h(y, x) \oplus x \oplus y)$ to the right oracle.

We give an indifferentiability theorem for Streebog. The full proof is provided for the Streebog variant with output size $k = 512$. For the shortened Streebog variant argumentation is completely similar. Formally, the only thing which has to be adjusted is the construction of the simulator; we will highlight the difference in the proof. The general structure of the proof and some techniques are adopted from [10, 11].

Theorem 1. *The hash function* Streebog *with $k = 512$ or 256 using a cipher $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is $(t_D, q_H, q_E, \varepsilon)$-indifferentiable from a random oracle in the ideal cipher model for E for any t_D with*

$$\varepsilon = \frac{(1 + l_m)q}{2^{n-4}} + \frac{(1 + n + l_m)q^2}{2^{n-7}},$$

where $q = q_E + q_H(l_m + 2)$ and l_m is the maximum message length (in blocks, including padding) queried by the distinguisher to its left oracle.

Proof. The main goal of the proof is to show that no distinguisher can tell apart two words: in the first one, it has access to the Streebog construction using an ideal cipher as an underlying block cipher and to the ideal cipher itself; in the second one it has access to a random oracle and a simulator. The first step of the proof is to present a simulator for which it would be possible to achieve that goal.

Our simulator for the ideal cipher $\mathcal{E}$ is quite elaborate. On every distinguisher query, it tries to detect whether the distinguisher seeks to compute Streebog for some message itself. If that is the case, it chooses the reply consistently with the random oracle; otherwise, it chooses the answer randomly.

The Simulator. Before we proceed with the simulator itself, let us define an auxiliary function $g_0 : \{0,1\}^* \to (\{0,1\}^n, \{0,1\}^n)^*$:

$$g_0(X) = (x_1, \Delta_1)\|(x_2, \Delta_2)\| \dots \|(x'_l\|10^{n-1-|x'_l|}, \tilde{\Delta}_l)\|(L, 0),$$

where $L = |X|$, $l = \lfloor L/n \rfloor + 1$, $X = x_1\| \dots \|x'_l$, where $x_1, \dots, x_{l-1} \in \{0,1\}^n, x'_l \in \{0,1\}^{<n}$ and x'_l is an empty string if L is already divisible by n. Clearly, if $\Sigma = \sum_{i=1}^{l-1} x_i + (x'_l\|10^{n-1-|x'_l|})$, then $g_0(X)\|(\Sigma, 0) = g(X)$.

The simulator accepts two types of queries: either a forward ideal cipher query $(+, y, x)$, where $x \in \{0,1\}^n$ corresponds to a plaintext and $y \in \{0,1\}^n$ to a cipher key, on which it returns a ciphertext $z \in \{0,1\}^n$; or an inverse query $(-, y, z)$, on which it returns a plaintext x. The simulator maintains a table T, which contains triples $(y, x, z) \in \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n$.

Forward query. When the simulator gets a forward query $(+, y, x)$ it searches the table T for a triple (y, x, z) for some z. It returns z if such triple exists. If there is no such triple, the simulator chooses z randomly, puts the triple (y, x, z) in the table and returns z to the distinguisher. Additionally, in that case the simulator proceeds with the following routine. It searches the table for a sequence $(y_1, x_1, z_1), \dots, (y_l, x_l, z_l)$ of length $l = \lfloor \mathrm{int}(x)/n \rfloor + 1$ such that:

- there exists X such that $g_0(X) = (x_1, \Delta_1)\|(x_2, \Delta_2)\| \dots \|(x_l, \tilde{\Delta}_l)\|(x, 0)$;
- it is the case that $y_1 = IV$;
- for each $i = 2, \dots, l$, it is the case that $y_i = x_{i-1} \oplus y_{i-1} \oplus z_{i-1} \oplus \Delta_{i-1}$;
- it is the case that $y = x_l \oplus y_l \oplus z_l \oplus \tilde{\Delta}_l$.

If such sequence exists, the simulator forms a pair (y_{l+2}, x_{l+2}) such that $y_{l+2} = x \oplus y \oplus z$ and $x_{l+2} = \sum_{i=1}^{l-1} x_i + x'_l$, where $X = x_1\| \dots \|x'_l$. It is easy to see that $g(X) = (x_1, \Delta_1)\| \dots \|(x_l, \tilde{\Delta}_l)\|(x, 0)\|(x_{l+2}, 0)$. The simulator does nothing if there already exists a triple (y_{l+2}, x_{l+2}, z') for some z' in the table T. Otherwise, it computes z' to form a triple (y_{l+2}, x_{l+2}, z'), which will be consistent with a random oracle output on X, in advance. To do that it queries the random oracle to get the output $Z = \mathcal{H}(X)$, computes $z' = Z \oplus x_{l+2} \oplus y_{l+2}$ and stores the triple (y_{l+2}, x_{l+2}, z') into the table T [1].

Inverse query. On an inverse query $(-, y, z)$ the simulator acts almost similarly. It searches the table T for a triple (y, x, z) for some x. It returns x if such triple exists. If there is no such triple, the simulator chooses x randomly, puts the triple (y, x, z) in the table and returns x to the distinguisher. In that case it proceeds with completely the same routine as described above.

We will denote the number of entries in the table T by q. It is clear that $q_E \leqslant q \leqslant 2q_E$ since for every adversarial query to S at most one additional record might be added to the table T besides the answer to the query itself.

[1] In $k = 256$ case the simulator first pads Z with 256 randomly chosen bits and then computes $z' = Z \oplus x_{l+2} \oplus y_{l+2}$.

Proof of Indifferentiability. Due to the definition of indifferentiability, if the following inequality holds for every distinguisher $\mathcal{D}$:

$$\left|\Pr\left[\mathcal{D}^{H,\mathcal{E}} \to 1\right] \Pr\left[\mathcal{D}^{\mathcal{H},S} \to 1\right]\right| \leqslant \varepsilon,$$

then the theorem follows. Hence, we have to prove that no distinguisher $\mathcal{D}$ can tell apart these two words unless with the probability ε. We will do that using the game hopping technique, starting in the world with the random oracle $\mathcal{H}$ and the simulator S and moving through the sequence of indistinguishable games to the world with the Streebog construction and the ideal cipher $\mathcal{E}$.

Game 1 → Game 2. The Game 1 is the starting point, where $\mathcal{D}$ has access to the random oracle $\mathcal{H}$ and the simulator S. In the Game 2 we give $\mathcal{D}$ access to the relay algorithm R_0 instead of direct access to $\mathcal{H}$. R_0, in its turn, has access to the random oracle and on distinguisher's queries simply answers with $\mathcal{H}(X)$. Let us denote by G_i the events that $\mathcal{D}$ returns 1 in Game i. It is clear that $\Pr[G_1] = \Pr[G_2]$.

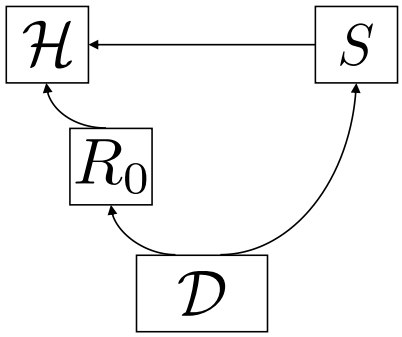

Figure 4: Game 2

Game 2 → Game 3. In the Game 3 we modify the simulator S by introducing failure conditions. The simulator explicitly fails (i.e. returns an error symbol $\perp$) while answering the distinguisher's query, if it computes the response satisfying one of the failure conditions below. Let S_0 denote the modified simulator.

We introduce two types of failure conditions. Each of the conditions captures different relations between the simulator's answers, which could be exploited by the distinguisher. By failing the simulator 'gives' the distinguisher an immediate win. Our longterm goal is to show that unless the failure happens, distinguisher cannot tell apart Game 2 form the ideal cipher world. The simulator S_0 chooses response to the forward or inverse query similarly to the simulator S and then checks the resulting triple (y, x, z) for the conditions defined bellow. For each type of conditions we also provide a brief motivation behind it, i.e., how the distinguisher can exploit corresponding situations to tell apart two worlds.

Conditions of type 1. Conditions of type 1 are checked if the answer to the query was chosen randomly or if it is the first time the value, which was chosen by the simulator to be consistent with the random oracle and put in the table earlier, is returned to the distinguisher.

1. *Condition* B_{11}. It is the case that $x \oplus y \oplus z = IV$.
2. *Condition* B_{12}. It is the case that there exists $l \in [1, l_m]$ such that $x \oplus y \oplus z \oplus \tilde{\Delta}_l = IV$.

3. *Condition* B_{13}. It is the case that there exist a triple $(y', x', z') \in T$ and $i \in [1, 2^n]$ such that $x \oplus y \oplus z = x' \oplus y' \oplus z' \oplus \Delta_i$. Note that $|\{\Delta_i,\ i \in [1, 2^n]\}| = n$.

4. *Condition* B_{14}. It is the case that there exist a triple $(y', x', z') \in T$ and $l \in [1, l_m]$ such that $x \oplus y \oplus z = x' \oplus y' \oplus z' \oplus \tilde{\Delta}_l$.

5. *Condition* B_{15}. It is the case that there exists a triple $(y', x', z') \in T$ such that $x \oplus y \oplus z = x' \oplus y' \oplus z'$.

The type 1 conditions correspond to a situation when internal states of two Streebog computational chains of different messages collide. The distinguisher can exploit that situation in a number of ways, for example, it can force these two chains to end with the same block, which will give the same result for two different messages. From that, the distinguisher can easily distinguish the two worlds by querying its left oracle with these messages. Other bad situations which correspond to that type of conditions are analyzed in the proof of Lemma 1.

Conditions of type 2. Conditions of type 2 are checked if only the answer to the query was chosen by the simulator randomly (i.e., the answer was not taken from the table).

1. *Condition* B_{21}. It is the case that there exists a triple $(y', x', z') \in T$ such that $x \oplus y \oplus z = y'$.

2. *Condition* B_{22}. It is the case that there exist a triple $(y', x', z') \in T$ and $i \in [1, 2^n]$ such that $x \oplus y \oplus z = y' \oplus \Delta_i$.

3. *Condition* B_{23}. It is the case that there exist a triple $(y', x', z') \in T$ and $l \in [1, l_m]$ such that $x \oplus y \oplus z = y' \oplus \tilde{\Delta}_l$.

The conditions of type 2 correspond to a situation when some block in the computational chain is queried sometime after the query corresponding to the next block was made. In that case, that query may be made even after the query for the last block in the chain was. The distinguisher then can easily tell apart two words since the simulator did not choose the answer to the last query to be consistent with the random oracle. Notice that conditions of that type are only checked when the simulator chooses the answer randomly itself. Otherwise, the distinguisher can easily force the failure event using the random oracle – for example, it can choose an arbitrary X, query the random oracle for $Z = \mathcal{H}(X)$, then query the right oracle with $(+, Z, x)$ for some x and finally compute the Streebog construction for X using its right oracle, the simulator would fail then due to condition B_{21} when answering for the last block of the computational chain. However, such a situation will not help the distinguisher since it, in some sense, corresponds to an extension of a computational chain of some message with new blocks, which will not lead to another valid computational chain due to our prefix-free encoding g. Bad situations which correspond to that type of conditions are analyzed in the proof of Lemma 2.

The probability of the event that the simulator fails due to one of the failure conditions is estimated as follows:

$$\Pr[S_0 \text{ fails}] \leqslant \frac{(1 + l_m)q_E}{2^{n-1}} + \frac{(1 + n + l_m)q_E^2}{2^{n-4}}.$$

That bound directly follows from Lemma 3 with $q_S = q_E$, which is given in Appendix A. The proof of that statement is rather technical and is also provided in Appendix A.

Since Game 2 and Game 3 are different only in situations, when the simulator S_0 fails, it is clear that

$$|\Pr[G_2] - \Pr[G_3]| \leqslant \Pr[S_0 \text{ fails}] \leqslant \frac{(1+l_m)q_E}{2^{n-1}} + \frac{(1+n+l_m)q_E^2}{2^{n-4}}.$$

Now, before we proceed to the next game, our aim is to show, that unless the simulator fails, its outputs are always consistent with random oracle outputs, i.e. it does not matter if the distinguisher is computing the Streebog construction with its right oracle (maybe in some unusual way) or queries the random oracle, the results would be the same. To do that we prove two lemmas, where Lemma 2 formalizes the outlined goal.

The first lemma states that in the table T there do not exist two sequences of triples, which correspond to computational chains of two different inputs, such that the last block of one chain is the first, middle or last block of the other, unless S_0 fails.

Lemma 1. *If the simulator S_0 does not fail, then there are no two different sequences of triples $(y_1, x_1, z_1), \ldots, (y_{l+2}, x_{l+2}, z_{l+2})$ and $(y'_1, x'_1, z'_1), \ldots, (y'_{p+2}, x'_{p+2}, z'_{p+2})$, where $l, p \leqslant l_m$, in the table T such that the following conditions hold:*

- *there exist X and X' such that $g(X) = (x_1, \Delta_1)\| \ldots \|(x_{l+1}, 0)\|(x_{l+2}, 0)$ and $g(X') = (x'_1, \Delta_1)\| \ldots \|(x'_{p+1}, 0)\|(x'_{p+2}, 0)$;*
- *it is the case that $y_1 = y'_1 = IV$;*
- *for each $i = 2, \ldots, l$ and $j = 2, \ldots, p$, it is the case that $y_i = x_{i-1} \oplus y_{i-1} \oplus z_{i-1} \oplus \Delta_{i-1}$ and $y'_j = x'_{j-1} \oplus y'_{j-1} \oplus z'_{j-1} \oplus \Delta_{j-1}$;*
- *it is the case that $y_{l+1} = x_l \oplus y_l \oplus z_l \oplus \tilde{\Delta}_l$ and $y'_{p+1} = x'_p \oplus y'_p \oplus z'_p \oplus \tilde{\Delta}_l$;*
- *it is the case that $y_{l+2} = x_{l+1} \oplus y_{l+1} \oplus z_{l+1}$ and $y'_{p+2} = x'_{p+1} \oplus y'_{p+1} \oplus z'_{p+1}$;*
- *there exists $s \in [1, l+2]$ such that $(y_s, x_s, z_s) = (y'_{p+2}, x'_{p+2}, z'_{p+2})$*

Proof. Let us suppose that there exist two sequences $(y_1, x_1, z_1), \ldots, (y_{l+2}, x_{l+2}, z_{l+2})$ and $(y'_1, x'_1, z'_1), \ldots, (y'_{p+2}, x'_{p+2}, z'_{p+2})$ in the table T, which satisfy conditions of the theorem. Then there exists the maximum $r \in [1, \min(s, p+2)]$ such that

$$(y_{s-i}, x_{s-i}, z_{s-i}) = (y'_{p-2-i}, x'_{p-2-i}, z'_{p-2-i}),\ i = 0, \ldots, r-1.$$

In other words, r is the length of subsequence of equal triples which ends with $(y_s, x_s, z_s) = (y'_{p+2}, x'_{p+2}, z'_{p+2})$. We will now consider several cases depending on values of r and l. Notice that $r \leqslant s \leqslant l+2$.

Consider the case $r = 1$. Since it is true that $(y_s, x_s, z_s) = (y'_{p+2}, x'_{p+2}, z'_{p+2})$ we can deduce that one of the following equalities has to hold:

1. if $s = 1$, then $y_s = IV$. Hence, $x'_{p+1} \oplus y'_{p+1} \oplus z'_{p+1} = y'_{p+2} = y_s = IV$;
2. if $s \in [2, l]$, then $y_s = x_{s-1} \oplus y_{s-1} \oplus z_{s-1} \oplus \Delta_{s-1}$. Hence, $x'_{p+1} \oplus y'_{p+1} \oplus z'_{p+1} = x_{s-1} \oplus y_{s-1} \oplus z_{s-1} \oplus \Delta_{s-1}$;
3. if $s = l+1$, then $y_s = x_{s-1} \oplus y_{s-1} \oplus z_{s-1} \oplus \tilde{\Delta}_{s-1}$. Hence, $x'_{p+1} \oplus y'_{p+1} \oplus z'_{p+1} = x_{s-1} \oplus y_{s-1} \oplus z_{s-1} \oplus \tilde{\Delta}_l$;

4. if $s = l+2$, then $y_s = x_{s-1} \oplus y_{s-1} \oplus z_{s-1}$. Hence, $x'_{p+1} \oplus y'_{p+1} \oplus z'_{p+1} = x_{s-1} \oplus y_{s-1} \oplus z_{s-1}$.

However, it is easy to see, that equalities above match failure conditions $B_{11}, B_{13}, B_{14}, B_{15}$ correspondingly. Hence, one of those failure conditions would have been triggered, when a forward or inverse query which corresponds to the triple $(y_{s-1}, x_{s-1}, z_{s-1})$ or $(y'_{p+1}, x'_{p+1}, z'_{p+1})$ (depending on which of them was made later) was made.

Consider the case $r \geqslant 2, l > 1$ **and** $r = 3, l = 1$**.** Since $r \geqslant 2$ it is easy to see, that the same inequality holds for s. Thereof, from $y'_{p+2} = y_s$ and the theorem statement we have that $x'_{p+1} \oplus y'_{p+1} \oplus z'_{p+1} \oplus 0 = x_{s-1} \oplus y_{s-1} \oplus z_{s-1} \oplus c$ for some $c \in \{\Delta_1, \ldots, \Delta_{l-1}, \tilde{\Delta}_l, 0\}$. However, since from $r \geqslant 2$ we have $(y_{s-1}, x_{s-1}, z_{s-1}) = (y'_{p+1}, x'_{p+1}, z'_{p+1})$, the constant c has to be equal to 0. It is also easy to see that none of the values $\{\Delta_1, \ldots, \Delta_{l-1}, \tilde{\Delta}_l\}$ is equal to 0 when $l > 1$. Hence, due to the encoding g, it is only possible that the triple (y_s, x_s, z_s) is the last one in the sequence and $s = l + 2$.

Thereof, $x_{l+1} = x'_{p+1}$, where, due to the definition of g, x_{l+1} and x'_{p+1} are equal to $|X|$ and $|X'|$ correspondingly. Consequently, since by definition $l = \lfloor |X|/n \rfloor + 1$ and $p = \lfloor |X'|/n \rfloor + 1$, we have that $p = l$.

Finally, consider triples $(y_{l+2-r}, x_{l+2-r}, z_{l+2-r}) \neq (y'_{l+2-r}, x'_{l+2-r}, z'_{l+2-r})$. Notice that $r < l + 2$ or else the considered sequences are equal (that excludes the $r = 3, l = 1$ case at all). Since $y_{l+2-r+1} = y'_{l+2-r+1}$ the following equality has to hold:

$$y_{l+2-r} \oplus x_{l+2-r} \oplus z_{l+2-r} \oplus c = y'_{l+2-r} \oplus x'_{l+2-r} \oplus z'_{l+2-r} \oplus c,$$

where c is equal either to Δ_{l+2-r} or $\tilde{\Delta}_{l+2-r}$. However, it is easy to see that in either way the equality matches the failure condition B_{15}. Hence, the it would have been triggered, when a forward or inverse query which corresponds to the triple $(y_{l+2-r}, x_{l+2-r}, z_{l+2-r})$ or $(y'_{l+2-r}, x'_{l+2-r}, z'_{l+2-r})$ (depending on which of them was made later) was made.

Consider the case $r = 2$ **and** $l = 1$ In this case we have, that $\tilde{\Delta}_l$ is equal to 0, hence two situations are possible. The first one is when $s = 3$, the reasoning here is completely the same as in the last case since equal triples are the two last triples in the sequences.

The second one is when $s = 2$. From that and since $r = 2$ we have that $(y_1, x_1, z_1) = (y'_{p+1}, x'_{p+1}, z'_{p+1})$. From the theorem statement, $y_1 = IV$ and $y'_{p+1} = x'_p \oplus y'_p \oplus z'_p \oplus \tilde{\Delta}_p$, thereof the following equality has to hold:

$$x'_p \oplus y'_p \oplus z'_p \oplus \tilde{\Delta}_p = IV.$$

However, it is easy to see, that the equality matches the failure condition B_{12}. Hence, it would have been triggered, when a forward or inverse query which corresponds to the triple (y'_p, x'_p, z'_p) was made.

Now we notice that we have considered all possible pairs (r, l). Hence, we can conclude that no such sequences can exist if the simulator S_0 does not fail. □

Now we prove, that the outputs of the simulator are consistent with the random oracle, unless it fails. To do that we show, that, if the distinguisher at some point computes the Streebog construction itself, it has to do that block-by-block with the last triple of the computational chain being consistent with the random oracle.

Lemma 2. *Consider any sequence of triples* $(y_1, x_1, z_1), \ldots, (y_{l+2}, x_{l+2}, z_{l+2})$*, where* $l \leqslant l_m$*, from the table* T *such that the following conditions hold:*

- *there exists* X *such that* $g(X) = (x_1, \Delta_1) \| \ldots \| (x_{l+1}, 0) \| (x_{l+2}, 0)$*;*

– *it is the case that* $y_1 = IV$*;*

– *for each* $i = 2, \ldots, l$*, it is the case that* $y_i = x_{i-1} \oplus y_{i-1} \oplus z_{i-1} \oplus \Delta_{i-1}$*;*

– *it is the case that* $y_{l+1} = x_l \oplus y_l \oplus z_l \oplus \tilde{\Delta}_l$*;*

– *it is the case that* $y_{l+2} = x_{l+1} \oplus y_{l+1} \oplus z_{l+1}$*.*

Then, if the simulator S_0 *does not fail, then it must be the case the triples* $(y_1, x_1, z_1), \ldots, (y_{l+1}, x_{l+1}, z_{l+1})$ *were put in the table* T *exactly in that order and answers to the corresponding queries were chosen randomly by the simulator. It also must be that the triple* $(y_{l+2}, x_{l+2}, z_{l+2})$ *was put in the table simultaneously with the triple* $(y_{l+1}, x_{l+1}, z_{l+1})$*, chosen to be consistent with the random oracle output* $\mathcal{H}(X)$*.*

Proof. Let us suppose that there exists $i \in [1, \ldots, l+1]$ such that the triple (y_i, x_i, z_i) was put in the table as a result of the corresponding forward of inverse query, when the triple $(y_{i+1}, x_{i+1}, z_{i+1})$ already existed in the table T. For that pair of triples the following equality holds:

$$y_i \oplus x_i \oplus z_i \oplus c = y_{i+1},$$

where c is one of the values $\{\Delta_i, \tilde{\Delta}_i, 0\}$, depending on the value of i. From Lemma 1 it follows, that the triple (y_i, x_i, z_i) could not be the last one in the computational chain of some message $X' \neq X$. In other words, the answer to the corresponding query was not chosen to be consistent with the random oracle, but chosen randomly by the simulator. Hence, on the query corresponding to the triple (y_i, x_i, z_i) one of the failure conditions of type 2 would have been triggered.

Thereby, when the query corresponding to the triple $(y_{l+1}, x_{l+1}, z_{l+1})$ is made, triples $(y_1, x_1, z_1), \ldots, (y_l, x_l, z_l)$ already exist in the table and the triple $(y_{l+2}, x_{l+2}, z_{l+2})$ does not. These triples satisfy conditions of the simulator's routine and it has to choose the triple $(y_{l+2}, x_{l+2}, z_{l+2})$ to be consistent with the random oracle and put it in the table with the triple $(y_{l+1}, x_{l+1}, z_{l+1})$. □

Game 3 → Game 4. In Game 4 we modify the relay algorithm R_0. Let R_1 denote the modified algorithm. R_1 does not have access to the random oracle. On a distinguisher query X it applies the Streebog construction to X, using the simulator for the block cipher E. Notice that now at most $q_E + q_H(l_m + 2)$ queries are made to S_0.

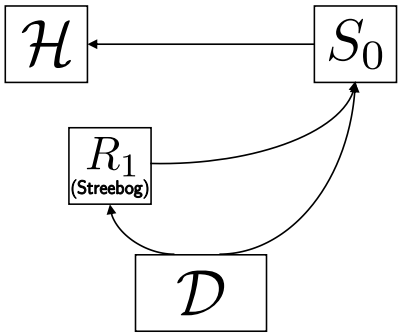

Figure 5: Game 4

Let us denote by $fail_3$ and $fail_4$ the events that the simulator fails in corresponding game. From Lemma 2 it follows that, unless the simulator does not fail, answers of the

modified relay algorithm R_1 are exactly the outputs of the random oracle on corresponding messages, since the simulator's answers are consistent with the random oracle. Hence, if the simulator does not fail in either world, the view of the distinguisher remains unchanged from Game 3 to Game 4:

$$\Pr\left[G_3|\overline{fail_3}\right] = \Pr\left[G_4|\overline{fail_4}\right].$$

Probability of the event $fail_3$ is estimated earlier in the transition from Game 2 to Game 3. Probability of the event $fail_4$ is estimated from Lemma 3, where $q_S = q_E + q_H(l_m + 2)$. Thus, we have:

$$\begin{aligned}
|\Pr[G_3] - \Pr[G_4]| = \left|\Pr\left[G_3|\overline{fail_3}\right]\Pr\left[\overline{fail_3}\right] + \Pr[G_3|fail_3]\Pr[fail_3] - \Pr\left[G_4|\overline{fail_4}\right]\cdot\right. \\
\left.\cdot\Pr\left[\overline{fail_4}\right] + \Pr[G_4|fail_4]\Pr[fail_4]\right| \leqslant \Pr\left[G_3|\overline{fail_3}\right]\cdot\left|\Pr\left[\overline{fail_3}\right] - \Pr\left[\overline{fail_4}\right]\right| + \\
+ |\Pr[G_3|fail_3]\Pr[fail_3] - \Pr[G_4|fail_4]\Pr[fail_4]| \leqslant |\Pr[fail_4] - \Pr[fail_3]| + \\
+ |\Pr[G_3|fail_3]\Pr[fail_3] - \Pr[G_4|fail_4]\Pr[fail_4]| \leqslant \max\left(\Pr[fail_3], \Pr[fail_4]\right) + \\
+ \max\left(1\cdot\Pr[fail_3] - 0\cdot\Pr[fail_4], 0\cdot\Pr[fail_3] + 1\cdot\Pr[fail_4]\right) \leqslant \\
\leqslant 2\max\left(\Pr[fail_3], \Pr[fail_4]\right) \leqslant 2\left(\frac{(1+l_m)(q_E + q_H(l_m+2))}{2^{n-1}} + \right. \\
\left. + \frac{(1+n+l_m)(q_E + q_H(l_m+2))^2}{2^{n-4}}\right).
\end{aligned}$$

Game 4 → Game 5. In Game 5 we modify the simulator. Let S_1 denote the modified simulator. S_1 does not consult the random oracle when answering the query, it still maintains a table T of triples (x, y, z). On a forward query $(+, y, x)$ it searches the table T for a triple (y, x, z) for some z. It returns z if such triple exists. If there is no such triple, the simulator chooses z randomly, puts the triple (y, x, z) in the table and returns z to the distinguisher. It acts similarly to answer the inverse query $(-, y, z)$, but chooses a random x, if there is no corresponding triple.

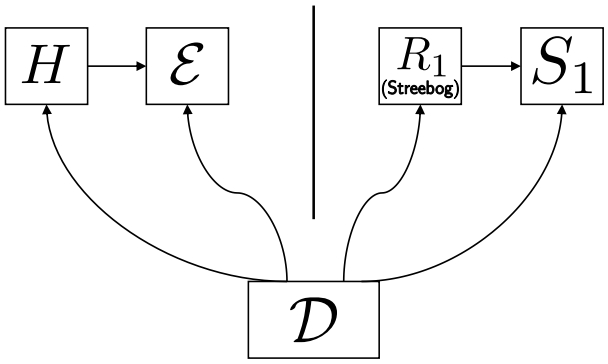

Figure 6: The ideal cipher world and Game 5

The responses of the simulators in these two games are identical, apart from the failure conditions of S_0. It is the case since, even when S_0 chooses the answer using the random oracle, all its answers look uniformly distributed to the distinguisher as it does not have a direct access to the random oracle in Game 4. Hence, the view of the distinguisher is identical in both games if the simulator does not fail in Game 4, and if in Game 5 the simulator does not give a response, which would have led to failure in Game 4. The probabilities of these events are equal since the number of queries to the simulators is the

same in both games, and the distribution of the responses of the simulators is identical. Let us denote the event "S_1 should have failed" by $fail_5$. Hence, the following inequality holds:

$$|\Pr[G_4] - \Pr[G_5]| \leqslant \left|\Pr\left[G_4|\overline{fail_4}\right]\Pr\left[\overline{fail_4}\right] + \Pr[G_4|fail_4]\Pr[fail_4] - \Pr\left[G_5|\overline{fail_5}\right]\cdot\right.$$
$$\left.\cdot\Pr\left[\overline{fail_5}\right] + \Pr[G_5|fail_5]\Pr[fail_5]\right| \leqslant |\Pr[G_4|fail_4]\Pr[fail_4] + \Pr[G_5|fail_5]\Pr[fail_5]| \leqslant$$
$$\leqslant \Pr[fail_4] + \Pr[fail_5] = 2\Pr[fail_4] \leqslant 2\left(\frac{(1+l_m)(q_E + q_H(l_m+2))}{2^{n-1}} + \right.$$
$$\left. + \frac{(1+n+l_m)(q_E+q_H(l_m+2))^2}{2^{n-4}}\right).$$

Game 5 → Game 6. In the final game we replace the simulator S_1 with the ideal cipher $\mathcal{E}$. Since the relay algorithm R_1 is the Streebog construction and now it uses the ideal cipher for E, the Game 6 is exactly the ideal cipher model.

We now have to show that the view of the distinguisher remains almost unchanged. The outputs of the ideal cipher and the simulator S_1 have different distributions – the ideal cipher is a permutation for each key and S_1 chooses its answers randomly. Hence, the distinguisher can tell apart two games only if forward/inverse outputs of the simulator collide for the same key. The probability of that event is at most the birthday bound through all queries. Thus, we have

$$|\Pr[G_5] - \Pr[G_6]| \leqslant \frac{(q_E + q_H(l_m+2))^2}{2^n}.$$

Finally, combining all transitions and since Game 6 is exactly the ideal cipher model, we can deduce that

$$\left|\Pr\left[\mathcal{D}^{H,\mathcal{E}} \to 1\right] - \Pr\left[\mathcal{D}^{\mathcal{H},S} \to 1\right]\right| \leqslant \frac{(1+l_m)q_E}{2^{n-1}} + \frac{(1+n+l_m)q_E^2}{2^{n-4}} +$$
$$+ 4\left(\frac{(1+l_m)(q_E+q_H(l_m+2))}{2^{n-1}} + \frac{(1+n+l_m)(q_E+q_H(l_m+2))^2}{2^{n-4}}\right) +$$
$$+ \frac{(q_E+q_H(l_m+2))^2}{2^n}.$$

The statement of Theorem 1 hence follows. □

4 Conclusion

In the paper we prove that the Streebog hash function is indifferentiable from a random oracle under the ideal cipher assumption for the underlying block cipher. It is still an open problem to determine if it is possible to prove indifferentiability of Streebog and other hash functions under idealized assumptions for even lower-level objects than a block cipher.

References

[1] GOST R 34.11-2012. National standard of the Russian Federation. Information technology. Cryptographic data security. Hash function, 2012.

[2] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., Kivinen, T. (2014). *Internet Key Exchange Protocol Version 2 (IKEv2)*, RFC 7296, October 2014.

[3] Rescorla, E. (2018) *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, August 2018.

[4] Smyshlyaev, S., Ed., Alekseev, E., Griboedova, E., Babueva, A., Nikiforova, L. (2023). *GOST Cipher Suites for Transport Layer Security (TLS) Protocol Version 1.3*, RFC 9367, February 2023.

[5] Smyshlyaev, S., Ed., Alekseev, E., Oshkin, I., Popov, V. (2017). *The Security Evaluated Standardized Password-Authenticated Key Exchange (SESPAKE) Protocol*, RFC 8133, March 2017.

[6] Smyslov, V. (2022). *Using GOST Ciphers in the Encapsulating Security Payload (ESP) and Internet Key Exchange Version 2 (IKEv2) Protocols*, RFC 9227, March 2022.

[7] Akhmetzyanova, L., Alekseev, E., Babueva, A., Smyshlyaev, A. (2021). *On methods of shortening ElGamal-type signatures*, Mat. Vopr. Kriptogr., 12:2, 2021, 75–91

[8] Alekseev, E., Smyshlyaev, S., (2020). *On security of the SESPAKE protocol*, Prikl. Diskr. Mat., n. 5, 2020, 5–41

[9] Bellare, M., Rogaway, P. (1993) *Random oracles are practical: A paradigm for designing efficient protocols*. In Proceedings of the 1st ACM conference on Computer and communications security, pp. 62–73. 1993.

[10] Coron, J. S., Dodis, Y., Malinaud, C., Puniya, P. (2005). *Merkle-Damgård revisited: How to construct a hash function*. In Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25 (pp. 430–448). Springer Berlin Heidelberg.

[11] Coron, J. S., Dodis, Y., Malinaud, C., Puniya, P. (2005). *Merkle-Damgård revisited: How to construct a hash function*. Full version. `https://cs.nyu.edu/~dodis/ps/merkle.pdf`.

[12] Guo J., Jean J., Leurent G., Peyrin T., Wang L., (2014). *The Usage of Counter Revisited: Second-Preimage Attack on New Russian Standardized Hash Function*, LNCS, Selected Areas in Cryptography – SAC 2014, 8781, ed. Joux A., Youssef A., Springer, Cham, 2014.

[13] Kiryukhin, V. (2022). *Keyed Streebog is a secure PRF and MAC*. Cryptology ePrint Archive.

[14] Maurer, U.M., Renner, R., Holenstein, C. (2004). *Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology*. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 21–39. Springer, Heidelberg (2004)

[15] Pointcheval, D., Stern, J. (1996). *Security proofs for signature schemes*. In Advances in Cryptology — EUROCRYPT'96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings 15 (pp. 387-398). Springer Berlin Heidelberg.

[16] Schnorr, C.P. (1990). *Efficient Identification and Signatures for Smart Cards*. In: Brassard, G. (eds) Advances in Cryptology — CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol 435. Springer, New York, NY. `https://doi.org/10.1007/0-387-34805-0_22`

[17] Smyshlyaev, S., Shishkin, V., Marshalko, G., Rudskoy, V., Lavrikov, I. (2019). *Overview of hash-function GOST R 34.11-2012 cryptoanalysis*, Information Security Problems. Computer Systems, 2019

[18] Tessaro, S, Zhu, C. (2022) *Short pairing-free blind signatures with exponential security*. In Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part II 2022 May 25 (pp. 782-811). Cham: Springer International Publishing.

[19] Vysotskaya, V., Chizhov, I. (2022) *The security of the code-based signature scheme based on the Stern identification protocol*, Prikl. Diskr. Mat., 2022, no. 57, 67–90

A Probability of the simulator's failure event

Lemma 3. *Let S_0 be a simulator defined in the proof of Theorem 1. Then the probability of the event that the simulator S_0 explicitly fails due to one of the failure conditions*

$B_{11}, \ldots, B_{23}$, defined in the proof of Theorem 1, satisfies the following bound:

$$\Pr[S_0 \textit{ fails}] = \frac{(1+l_m)q_S}{2^{n-1}} + \frac{(1+n+l_m)q_S^2}{2^{n-4}},$$

where q_S is a number of queries made to the simulator.

Proof. Let us denote by q the maximum number of entries in the table T, $q_S \leqslant q \leqslant 2q_S$. To estimate the desired probability, we consider each failure condition and bound the probability that there exists a query to the simulator satisfying the condition. Let us begin with conditions of type 1.

- *Condition B_{11}.* It is the propability that one of at most q random n-bit strings (where the randomness is due to either the simulator's random choice or the random oracle output) is equal to fixed IV. Hence,

$$\Pr[\exists \textit{ query satisfying } B_{11}] \leqslant \frac{q}{2^n}.$$

- *Condition B_{12}.* It is the propability that one of at most q random n-bit strings is equal to one of l_m strings $IV \oplus \tilde{\Delta}_l,\ l \in [1, l_m]$.

$$\Pr[\exists \textit{ query satisfying } B_{12}] \leqslant \frac{l_m \cdot q}{2^n}.$$

- *Condition B_{13}.* To estimate the probability of that event we will consider three separate situations.

  The first one is that there exists a query, satisfying the condition, answer to which was chosen by the simulator randomly. The probability of that situation is the propability that one of at most $q_S \leqslant q$ random n-bit strings is equal to one of nq strings $x' \oplus y' \oplus z' \oplus \Delta_i,\ (y', x', z') \in T,\ i \in [1, 2^n]$ (recall that $|\{\Delta_i,\ i \in [1, 2^n]\}| = n$). Hence,

$$\Pr[\exists \textit{ query satisfying } B_{12} \text{ and Situation 1}] \leqslant \frac{n \cdot q^2}{2^n}.$$

  The second one is that there exists a query, satisfying the condition, answer to which was chosen by the simulator to be consistent with the random oracle ($x \oplus y \oplus z$ is exactly the random oracle output then), and the triple $(y', x', z') \in T$ was constructed independently from the random oracle (the answer to the corresponding query was chosen randomly by the simulator itself). The probability of that situation is the propability that one of at most $q_S \leqslant q$ random oracle n-bit outputs is equal to one of nq strings $x' \oplus y' \oplus z' \oplus \Delta_i,\ (y', x', z') \in T,\ i \in [1, 2^n]$. Hence,

$$\Pr[\exists \textit{ query satisfying } B_{12} \text{ and Situation 2}] \leqslant \frac{n \cdot q^2}{2^n}.$$

  The third one is that there exists a query, satisfying the condition, answer to which was chosen by the simulator to be consistent with the random oracle, and the triple $(y', x', z') \in T$ was also constructed to be consistent with the random oracle. Then both $x \oplus y \oplus z$ and $x' \oplus y' \oplus z'$ are the random oracle outputs on different messages X and X' (they are different since both triples have to be the last blocks of some computational chains and there is only one computational chain for every X). The

probability of that situation is the propability that two random oracle outputs Z and Z' out of at most $q_S \leqslant q$ satisfy any of n equalities $Z \oplus Z' = \Delta_i$. Hence,

$$\Pr[\exists\ query \text{ satisfying } B_{12} \text{ and Situation 3}] \leqslant \frac{n \cdot q^2}{2^n}.$$

Finally, it easy to see that

$$\Pr[\exists\ query \text{ satisfying } B_{12}] \leqslant \Pr[\exists\ query \text{ satisfying } B_{12} \text{ and Situation 1}] +$$
$$+\Pr[\exists\ query \text{ satisfying } B_{12} \text{ and Situation 2}] + \Pr[\exists\ query \text{ satisfying } B_{12} \text{ and Situation 3}].$$

Hence,

$$\Pr[\exists\ query \text{ satisfying } B_{13}] \leqslant 3 \cdot \frac{n \cdot q^2}{2^n}.$$

– *Condition* B_{14}. The probability of that event is estimated similarly to the previous one with the difference that $|\{\tilde{\Delta}_l,\ l \in [1, l_m]\}| = l_m$. Hence,

$$\Pr[\exists\ query \text{ satisfying } B_{14}] \leqslant 3 \cdot \frac{l_m \cdot q^2}{2^n}.$$

– *Condition* B_{15}. The probability of that event is estimated similarly to the two previous ones:

$$\Pr[\exists\ query \text{ satisfying } B_{15}] \leqslant 3 \cdot \frac{q^2}{2^n}.$$

We proceed with conditions of type 2.

– *Condition* B_{21}. It is the propability that one of at most $q_S \leqslant q$ random n-bit strings, where the randomness is due to either the simulator's random choice or the random oracle output and independent from the distinguisher's random tape, is equal to one of q strings $y',\ (y', x', z') \in T$, where all y' are chosen by the distinguisher. Hence,

$$\Pr[\exists\ query \text{ satisfying } B_{21}] \leqslant \frac{q^2}{2^n}.$$

– *Condition* B_{22}. The probability of that event is estimated similarly to the previous one, with the difference that there are at most nq different strings $y' \oplus \Delta_i,\ (y', x', z') \in T,\ i \in [1, 2^n]$. Hence,

$$\Pr[\exists\ query \text{ satisfying } B_{22}] \leqslant \frac{n \cdot q^2}{2^n}.$$

– *Condition* B_{23}. The probability of that event is estimated similarly to the previous ones with the difference that there are at most $l_m \cdot q$ different strings $y' \oplus \tilde{\Delta}_l,\ (y', x', z') \in T,\ l \in [1, l_m]$. Hence,

$$\Pr[\exists\ query \text{ satisfying } B_{23}] \leqslant \frac{l_m \cdot q^2}{2^n}.$$

Finally, we estimate the probability of the event that the simulator fails:

$$\Pr[S_0 \text{ fails}] \leqslant \Pr[\exists\ query \text{ satisfying some bad condition}] \leqslant$$
$$\leqslant \frac{(1 + l_m)q}{2^n} + \frac{(4 + 4n + 4l_m)q^2}{2^n} = \frac{(1 + l_m)q_S}{2^{n-1}} + \frac{(1 + n + l_m)q_S^2}{2^{n-4}},$$

where the last inequality is due to $q \leqslant 2q_S$. □