Practical-Time Related-Key Attack on GOST with Secret S-boxes

Orr Dunkelman[1,⋆], Nathan Keller[2,⋆⋆], and Ariel Weizmann[2,⋆⋆⋆]

[1] Computer Science Department, University of Haifa, Israel
orrd@cs.haifa.ac.il
[2] Department of Mathematics, Bar-Ilan University, Israel
Nathan.Keller@biu.ac.il,relweiz@gmail.com

Abstract. The block cipher GOST 28147-89 was the Russian Federation encryption standard for over 20 years, and is still one of its two standard block ciphers. GOST is a 32-round Feistel construction, whose security benefits from the fact that the S-boxes used in the design are kept secret. In the last 10 years, several attacks on the full 32-round GOST were presented. However, they all assume that the S-boxes are known. When the S-boxes are secret, all published attacks either target a small number of rounds, or apply for small sets of weak keys.

In this paper we present the first practical-time attack on GOST with secret S-boxes. The attack works in the related-key model and is faster than all previous attacks in this model which assume that the S-boxes are known. The complexity of the attack is less than 2^{27} encryptions. It was fully verified, and runs in a few seconds on a PC. The attack is based on a novel type of related-key differentials of GOST, inspired by local collisions.

Our new technique may be applicable to certain GOST-based hash functions as well. To demonstrate this, we show how to find a collision on a Davies-Meyer construction based on GOST with an arbitrary initial value, in less than 2^{10} hash function evaluations.

1 Introduction

The block cipher GOST 28147-89 (usually shortened to GOST) was developed in the USSR in the 1970's, as an alternative for DES. From 1989 to 2015, it was the official encryption standard of the USSR, and then of the Russian Federation

⋆ The first author was supported in part by the Center for Cyber, Law, and Policy in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office and by the Israeli Science Foundation through grants No. 880/18 and 3380/19.

⋆⋆ The second author and the third author were supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

⋆⋆⋆ The third author was supported by the President Scholarship for Ph.D. students at the Bar-Ilan University.

(RF), and was obligatory to use in the RF in all data processing systems providing public services [16]. Since 2015, an instantiation of GOST with specified S-boxes named Magma is one of the two ciphers in the RF encryption standard GOST R 34.12-2015 [15]. Consequently, GOST is still very widely used in the Russian Federation.

GOST is a 32-round Feistel construction whose round function uses eight 4-to-4 bit S-boxes. The structure of the S-boxes was kept secret, and reportedly, different sets of S-boxes were used in different industry branches. The set used in the banking industry had leaked and was published in [30], and most previous attacks on GOST used that set of S-boxes. In the new standard GOST R 34.12-2015, the S-boxes were specified (to another set of values). Another central feature of the design of GOST is the key schedule. The 256-bit key is represented as an array of eight 32-bit words $(K_1, K_2, \ldots, K_8)$, and the subkeys used in the 32 rounds are taken directly from the array in a structured form. This property was exploited in several attacks on GOST.

Previous works. In the last 30 years, GOST has been the target of numerous cryptanalytic attempts. Most of these attempts assumed that the S-boxes are known (thus, not targeting the original strong version of the cipher). Under this assumption, in the standard single-key model, several attacks can break the full 32-round GOST faster than exhaustive key search [11, 12, 20], but all of them have an impractical time complexity of at least 2^{179} encryptions (see also [10] and the multiple references therein). In the related-key model (in which the adversary may request encryptions under pairs of unknown keys with a known relation and her goal is to recover the keys), several practical-time attacks on the full 32-round GOST were obtained. After two works that could attack only reduced-round variants [21, 31], Ko et al. [24] were the first to obtain a related-key attack on the full 32-round variant. Their attack, which uses the related-key differential technique [21], requires 2^{36} chosen plaintexts and time of 2^{36} encryptions. Biryukov and Nikolic [7] presented an attack on the full 32 rounds which uses complementation properties and requires 2^{38} chosen plaintexts and 2^{38} encryptions. Rudskoy [28] and Pudovkina and Khoruzhenko [26] presented related-key boomerang attacks [22] with semi-practical complexities. The best among these attacks, by Ko et al. [24], has a complexity of 2^{36}.

Only several papers targeted the original variant of GOST, with secret S-boxes. Saarinen [29] presented an attack with a complexity of 2^{32} that applies for the 2^{32} keys of the form $(K, K, \ldots, K, K)$. Bar-On et al. [2] presented an attack on 24 rounds that applies for all keys and has a complexity of 2^{63}, as well as an attack on the full 32-round version that applies for the 2^{128} keys of the form $(K_1, K_2, K_3, K_4, K_4, K_3, K_2, K_1)$ and has a complexity of 2^{40}. All these attacks are based on variants of the slide technique [8]. Zhao et al. [34] presented an attack on the full 32-round variant using algebraic fault analysis. They showed that insertion of 270 faults and time of a few hours are sufficient to recover the secret S-boxes. Neither of these attacks endanger the security of the full GOST with secret S-boxes – the attacks either target partial encryption, or apply only for a small set of weak keys, or require using the side-channel attack model.

No. of Rounds	Fraction of Keys	Secret S-boxes?	Data[a]	Time[b]	Technique and Source[c]
21	all	no	2^{56} CP	2^{56}	RK Diff. [31]
24	all	no	?[d]	?[d]	RK Diff. [21]
25	all	no	5 CP	2^{32}	RK Diff. [27]
32	all	no	2^{36} CP	2^{36}	RK Diff. [24]
32	all	no	2^{38} CP	2^{38}	Complementation [7]
32	all	no	2^{10} ACPC	2^{71}	RK Boom. [28]
32	all	no	?[d]ACPC	?[d]	RK Boom. [26]
24	all	yes	2^{63} CP	2^{63}	Slide [2]
32	2^{-224}	yes	2^{32} CP	2^{32}	Slide [29]
32	2^{-128}	yes	2^{40} CP	2^{40}	Slide [2]
32	all	yes	2^{27} CP	2^{27}	RK Diff. (Sec. 4)

[a] Time is measured in GOST encryptions
[b] "CP" — Chosen plaintext, "ACPC" — Adaptive Chosen Plaintext and Ciphertext
[c] "RK" — Related-Key, "Diff." — Differential, "Boom." — Boomerang
[d] The notation '?' means that the attack complexity was not specified

Table 1: Comparison of Our Results with Previous Attacks on GOST

Our contributions. In this paper we present the first practical-time attack on the full GOST with secret S-boxes. Our attack, which works in the related-key model, recovers the secret S-boxes and the secret key, requiring only 2^{27} chosen plaintexts and time of 2^{27} encryptions in the worst case (among 100 experiments), and about 2^{24} chosen plaintexts and time of 2^{24} encryptions on average. Thus, our attack is significantly faster than all previously known related-key attacks on GOST, although those attacks assume that the S-boxes are known. Needless to say, our attack is significantly stronger than all previous attacks on GOST with secret S-boxes, as none of those attacks can break the full GOST for all keys. The attack was fully verified experimentally and runs in a few seconds on a PC. A comparison of the complexity of our attack with the complexities of previously known attacks on GOST is presented in Table 1.

Like the attack of Ko et al. [24], our attack uses the related-key differential technique [21]. However, the differential characteristic we use differs significantly from the characteristic used in [24]. Our characteristic has the form of *local collisions* between two encryption processes over three rounds of GOST, in which in the first round, a state difference is created by a subkey difference, in the second round the state difference is 'kept from spreading', and in the third round the state difference is canceled by another properly selected subkey difference. Such local collisions, first proposed by Chabaud and Joux [9], were very effective in collision attacks against hash functions from the SHA family (e.g., [5,32,33]). We use them for the GOST block cipher at the first time.

Being a collision-based related-key attack, our attack is naturally effective against certain types of GOST-based hash functions. We demonstrate this effectiveness by showing that for a Davies-Meyer hash function based on GOST, one can find a collision in less than 2^{10} hash function evaluations, for an arbitrary initial value. Previously, the techniques from Mendel et al.'s attack [25] on the GOST hash function [17] could be used to find a collision in a Davies-Meyer hash function based on GOST almost instantly, but only if the 64-bit initial value is of the form (x, x) for a 32-bit value x. Otherwise, no ways to find a collision in less than 2^{32} hash function evaluations were known.

We present two additional applications of our techniques: the first S-box recovery attack on the cipher GOST2 [13], and a simplification of the S-box recovery attack of Bar-On et al. [2] on GOST with a palindromic key schedule.

Organization of the paper. In Section 2 we describe the structure of GOST and briefly recall the related-key differential technique used in the paper. In Section 3 we present the new type of differentials of GOST we employ. The related-key attack on the full 32-round GOST is presented in Section 4. Potential applications of the techniques to hash functions based on GOST, including the GOST hash function [17], are discussed in Section 5. We conclude the paper with a summary and discussion in Section 6.

2 Preliminaries

2.1 The Structure of GOST

GOST 28147-89 is a 64-bit block size, 256-bit key size block cipher, composed of 32 Feistel rounds. For each $1 \leq i \leq 32$, the i'th round is defined as follows (see Figure 1):

$$F_{K_i}(X_L, X_R) = (X_R, X_L \oplus \lll_{11} (S(X_R \boxplus K_i))),$$

where:

- $\oplus$ denotes bit-wise XOR and $\boxplus$ denotes modular addition modulo 2^{32}.
- For each 32-bit word A, $\lll_{11} (A)$ denotes cyclic left-rotation of A by 11 bits.
- K_i is the round key. The key schedule is very simple: Divide the 256-bit key into eight 32-bit subkeys $K_1, \ldots, K_8$. These subkeys are used in this order three times in rounds 1–24, and in the reverse order $K_8, \ldots, K_1$ in the last 8 rounds 25–32.
- S is an S-box layer of eight 4-to-4 bit S-boxes[1] $S_0 \ldots, S_7 : \{0,1\}^4 \to \{0,1\}^4$, where S_0 is performed on the four least significant bits, and S_7 is performed on the four most significant bits. These S-boxes are kept secret. In addition, they are not necessarily permutations.

[1] The somewhat nonstandard notations used here follow the notations presented in the up-to-date official document describing GOST [15].

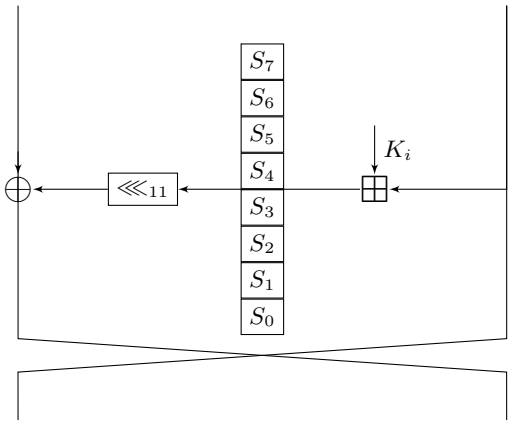


Fig. 1: One GOST Round.

2.2 Related-Key Differential Attacks

Differential attacks. Differential cryptanalysis was introduced by Biham and Shamir [6]. It analyzes the development of differences through the encryption process of pairs of plaintexts.

Let E be an n-bit block cipher consisting of r rounds. A differential with probability p of E is a statistical property of the form $\Pr[E(P) \oplus E(P') = \Omega_O \mid P \oplus P' = \Omega_I] = p$, denoted by $\Omega_I \xrightarrow{p} \Omega_O$. If $p \gg 2^{-n}$, the differential can be used to distinguish E from a random permutation, given $O(p^{-1})$ pairs of plaintexts with difference Ω_I.

Differentials can be used for key-recovery attacks as well, in a procedure called *iR-attack*. In this procedure, the adversary finds a differential $\Omega_I \xrightarrow{p} \Omega_O$ for the first $r - i$ rounds of E and uses it to recover key material in the last i rounds. First, the adversary asks for the encryption of $O(p^{-1})$ pairs (P, P') of plaintexts that satisfy $P \oplus P' = \Omega_I$. Then, she guesses some of the subkey bits used in the last i rounds, partially decrypts the ciphertext pairs through the last i rounds and checks whether the difference at the input to the $(r - i + 1)$'th round is equal to Ω_O at least several times. As the data is expected to contain several plaintext pairs that satisfy the differential, it is expected that the check succeeds for the correct key guess and fails for wrong key guesses with a high probability.

Related-key differential cryptanalysis. Related-key (in short, RK) attacks were introduced by Biham [3] and by Knudsen [23], independently. The attack model in these attacks is that the adversary may obtain the encryption of plaintexts under several related unknown keys, where the relation between the keys is known to (or can be chosen by) the adversary. The goal of the adversary is to recover the keys.

In [21], Kelsey et al. introduced the related-key differential cryptanalysis. In a related-key differential attack, the adversary can ask for the encryption of plaintext pairs with a chosen difference Ω_I (i.e., $P, P' = P \oplus \Omega_I$), under unknown keys with a chosen difference Ω_K (i.e., $K, K' = K \oplus \Omega_K$). A related-key differential with a probability of p of a block cipher E under two keys $K, K' = K \oplus \Omega_K$ is a statistical property of the form $\Pr[E_K(P) \oplus E_{K'}(P') = \Omega_O \mid P \oplus P' = \Omega_I] = p$, denoted by $\Omega_I \xrightarrow[\Omega_K]{p} \Omega_O$. Related-key differentials can be used for key-recovery in a similar way to ordinary differentials.

3 The New Related-Key Differential of GOST

In this section we present the new related-key differential of GOST which we use in our attacks. The differential is based on a 3-round 'local collision', inspired by local collisions in hash functions, first proposed by Chabaud and Joux [9].

For the sake of concreteness, we first present the differential for the special case of GOST with the S-boxes used in the banking industry, which was considered in most previous works on GOST. Afterwards, we show how to use the differential when the S-boxes are unknown.

3.1 The Basic 3-Round Iterative Related-Key Differential

Consider the encryption through the first three rounds of GOST of a plaintext P under two related keys K, K' such that $K'_1 = K_1 \oplus e_{31}, K'_2 = K_2 \oplus e_{10}, K'_3 = K_3 \oplus e_{31}$ (see Fig. 2).

At the first round, since the state difference is zero and the subkey difference is in the most significant bit, the modular addition behaves like XOR with respect to differences, and thus, the XOR difference after the key addition is e_{31}. As the S-box S_7 in the set used in the banking industry satisfies the differential $8 \xrightarrow{1/4} 8$, with a probability of $\frac{1}{4}$ the difference after the S-box layer is e_{31}, which is mapped to e_{10} by the left rotation. At the second round, the input difference e_{10} is canceled by the sub-key difference with a probability of $\frac{1}{2}$. At the third round, the difference after the key addition is e_{31} (like in the first round), and thus, with a probability of $\frac{1}{4}$ the difference after the rotation is e_{10}, which is canceled in the XOR operation at the end of the round, resulting in a zero state difference at the input of the fourth round. Hence, we get the following 3-round iterative differential characteristic:

$$(0,0) \xrightarrow[\Omega_{K_1}=e_{31}]{\frac{1}{4}} (0,e_{10}) \xrightarrow[\Omega_{K_2}=e_{10}]{\frac{1}{2}} (e_{10},0) \xrightarrow[\Omega_{K_3}=e_{31}]{\frac{1}{4}} (0,0),$$

as depicted in Fig. 2. Since this related-key differential characteristic is of the form $0 \to 0$, it can be viewed as a *local collision*.

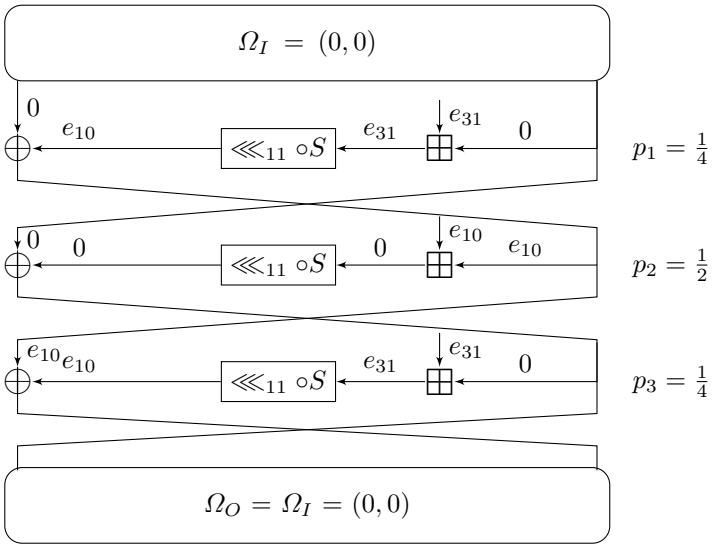


Fig. 2: Our 3-Round iterative RK differential characteristic on GOST (using the banking industry S-boxes).

3.2 The Full 32-Round Differential

Consider the encryption of a plaintext P under two keys K, K' with key difference

$$\Omega_K = (e_{31}, e_{10}, e_{31}, 0, 0, 0, 0, 0).$$

By the 3-round iterative differential characteristic described above we have the following 8-round iterative differential:

$$(0,0) \xrightarrow[\Omega_K]{2^{-5}} (0,0).$$

Since the eight sub-keys used in rounds 9–16 and 17–24 are the same as in rounds 1–8, we obtain the 24-round iterative differential:

$$(0,0) \xrightarrow[\Omega_K]{2^{-15}} (0,0).$$

In rounds 25–32, the subkeys are used in reverse order, and hence, there is no key difference until round 29 (inclusive). Thus, we get the following 29-round differential:

$$(0,0) \xrightarrow[\Omega_K]{2^{-15}} (0,0).$$

At the last three rounds, the subkey differences are e_{31}, e_{10}, e_{31}, respectively. Hence, we may apply again the basic three-round differential characteristic, to obtain a 32-round related-key differential with probability of 2^{-20}. In order to

reduce the data complexity of attacks exploiting the differential, we prefer to use in these rounds a truncated differential characteristic (i.e., a differential characteristic that predicts the difference only in part of the state) which holds with a probability close to 1.

For this sake, we examine the development of the difference in the last three rounds. The input difference to round 30 is zero and the subkey difference is e_{31}. Thus, after the key addition the difference is e_{31}, and after the S-box layer the difference is of the form $?0000000_x$ (where ? is an unknown 4-bit value), which is mapped by the left rotation to $00000XY0_x$ (where $X \in \{0,\ldots,7\}, Y \in \{0,8\}$).

At round 31, with a probability of over 99% the truncated difference after the key addition (in which the subkey difference is e_{10}) is of the form $000???Q0_x$ (where $Q \in \{0,8\}$), since we require the addition carry to go through at most 9 bits. After the S-box layer the difference is $000????0_x$, which is mapped to $Z???W000_x$ (where $W \in \{0,8\}, Z \in \{0,\ldots,7\}$) by the left rotation. This difference is copied to the right half of the ciphertext.

At round 32, the difference after the key addition (in which the subkey difference is e_{31}) is $????T000_x$. After the S-box layer, the difference is $?????000_x$ which is mapped by the left rotation to $??U00V??_x$ (where $V \in \{0,\ldots,7\}, U \in \{0,8\}$). This is the difference in the left half of the ciphertext.

Hence, we have a related key truncated differential on the entire cipher

$$(0,0) \xrightarrow[\Omega_K]{2^{-15}} (??U00V??_x, Z???W000_x),$$

where $U,W \in \{0,8\}, V,Z \in \{0,\ldots,7\}$, and ? is an unknown value, as depicted in Figure 3. To conclude, this related-key truncated differential predicts a zero difference in 28 bits with a probability of about 2^{-15}.

3.3 The Related-Key Differential for GOST with Secret S-boxes

Recall that our 3-round iterative differential characteristic,

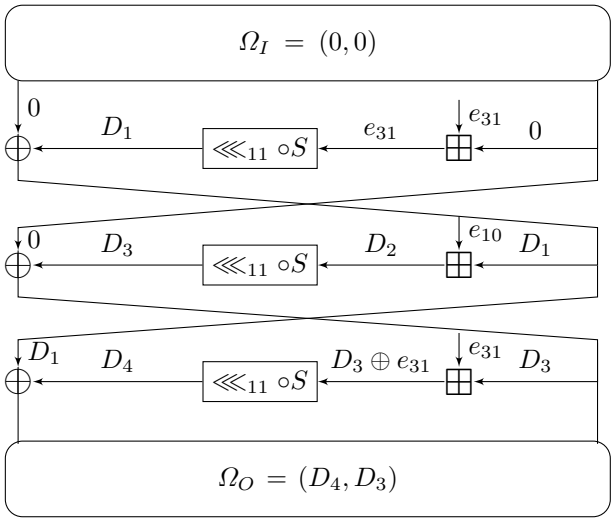
$$(0,0) \xrightarrow[\Omega_{K_1}=e_{31}]{\frac{1}{4}} (0,e_{10}) \xrightarrow[\Omega_{K_2}=e_{10}]{\frac{1}{2}} (e_{10},0) \xrightarrow[\Omega_{K_3}=e_{31}]{\frac{1}{4}} (0,0),$$

relies on the differential transition $8 \xrightarrow{\frac{1}{4}} 8$ in the S-box S_7. When S_7 is a secret S-box, this differential might be impossible, and we have no direct way to check its probability.

To overcome this, we consider four additional related keys, obtained by changing the difference in K_2. Namely, while we leave the key difference in K_1, K_3 fixed at e_{31}, we consider four differences in K_2:

$$\Omega^0_{K_2} = e_7, \Omega^1_{K_2} = e_8, \Omega^2_{K_2} = e_9, \Omega^3_{K_2} = e_{10}.$$

If in S_7, a differential of the form $8 \xrightarrow{p} (2^i + 2^{i+1} + \ldots + 2^{i+b-1})$ is satisfied (where $i \in \{0,1,2,3\}$ and $b \in \{1,\ldots,4-i\}$; note that the sum contains between



$D_1 = 00000XY0_x, D_2 = 000???Q0_x D_3 = Z???W000_x, D_4 = ??U00V??_x$, where $X, Z, V \in \{0, \ldots, 7\}, Q, Y, W, U \in \{0, 8\}$.

Fig. 3: The last three Rounds of our RK truncated differential characteristic on the full GOST (considering the banking industry S-boxes).

one and four terms), then the following 3-round differential characteristic holds,[2] with overall probability of $p^2 2^{-b}$:

$$(0,0) \xrightarrow[\Omega_{K_1}=e_{31}]{p} (0, e_{7+i,7+i+1,\ldots,7+i+b-1}) \xrightarrow[\Omega^i_{K_2}=e_{7+i}]{2^{-b}} (e_{7+i,7+i+1,\ldots,7+i+b-1}, 0) \xrightarrow[\Omega_{K_3}=e_{31}]{p} (0,0).$$

Furthermore, as we are interested only in the probability of having a zero output difference after three rounds for a given subkey difference (formally called 'probability of a differential') and not in the probability of the exact transition sequence from the zero input difference to the zero output difference (formally called 'probability of a differential characteristic'), we can enjoy several differential characteristics at the same time. Indeed, when we consider the encryption of two identical plaintexts under the subkey difference $\Omega(K_1, K_2, K_3) = (e_{31}, e_{7+i}, e_{31})$, the probability of having a zero difference after three rounds is at least $\sum_{b=1}^{4} p_b^2 2^{-j}$, where p_b is the probability of the transition $8 \to (2^i + 2^{i+1} + \ldots + 2^{i+b-1})$ in S_7.

[2] If a differential of the form $8 \xrightarrow{p} 0$ is satisfied, then an even stronger 1-round iterative differential characteristic of GOST can be constructed, as is described in Section 3.4. We note that the existence of such a transition implies that the S-boxes are not bijective, but the official document describing GOST [16] permits using such S-boxes.

Note that out of the 16 possible output differences of S_7, 11 are of the prescribed form (for some b, i). Hence, for a random S-box, with an overwhelming probability at least one of these differentials is possible.

Among the differential characteristics we consider, the lowest probability is obtained in the case $p = 2^{-3}, b = 4$, in which the probability of the entire truncated differential is about $\left(p^2 \cdot 2^{-b}\right)^3 = 2^{-30}$. (Note that in a 4-bit S-box, the lowest possible non-zero probability of a differential is 2^{-3}). The highest probability is obtained in the case $p = 1, b = 1$, in which the probability of the entire truncated differential is at least $\left(p^2 \cdot 2^{-b}\right)^3 = 2^{-3}$.

In practice (as we have verified by running experiments on many randomly chosen S-boxes), in most cases for at least one of the four key differences, the overall probability of the truncated differential is at least 2^{-24}. Thus, by repeating the attack procedure for at most 5 key differences, we will be able to use the differential in the secret S-box setting (also obtaining some information on the S-box on the way).

3.4 Other Variants of the Differential

Besides the variants described above, many other variants of the differential can be considered. For example, instead of inserting the local collision at the first three rounds, one may insert it at any three other consecutive rounds. The active S-box S_7 can be replaced with any other S-box, and the input difference 8 can be replaced with any other input difference. The key difference in the third round can differ from the difference in the first round, as long as it is contained in the same S-box. Some of these changes affect the probability of the differential (e.g., when the subkey difference at the first and the third rounds is e_i for $i \neq 31$, we have to 'pay' probability of $(1/2)^2$ to bypass the key addition operations at the first and the third rounds).

The following additional variant could be useful in the case of non-invertible S-boxes (which are allowed by the design of GOST). If a differential of the form $(2^i + 2^{i+1} + \ldots + 2^{i+b-1}) \xrightarrow[S_j]{p} 0$ (where $i \in \{0, 1, 2, 3\}$ and $b \in \{1, \ldots, 4-i\}$) is satisfied, then we get an 1-round iterative differential characteristic $(0, 0) \xrightarrow[\Omega_{K_1 = e_{4j+i}}]{p \cdot 2^{-b} \geq 2^{-7}} (0, 0)$. Using this characteristic, one can easily construct a related-key differential for the full cipher of the form $(0, 0) \xrightarrow[\Omega_K = (e_{4j+i}, 0, 0, 0, 0, 0, 0, 0)]{p^3 \cdot 2^{-3b} \geq 2^{-21}} (D, 0)$, where D has a few active bits. Alternatively, one may reach a ciphertext difference with more active bits by inserting the key difference e_{4j+i} in the subkey K_2. Using this variant of the differential, the adaptation of the attack described in Section 4 to the setting of non-invertible S-boxes is very simple. Hence, from now on we focus on the case of invertible S-boxes.

We use several of these variants in our attack (see Sections 4.3 and 4.4); other variants may be useful for future attacks on GOST and on other related cryptosystems. As a concrete application, we note that a variant of the differential applies to GOST2 – a variant of GOST with a modified key schedule that was

proposed in [13] and studied in [1,14]. While the primary goal of the modified key schedule proposed in GOST2 was to thwart attacks based on the key schedule, a slight change of our differential holds for the modified key schedule as well. Specifically, using the same key difference, the only significant change is that the last occurrence of the 'local collision' is in rounds 29–31 instead of 30–32. This slightly increases the complexity of our attack (as one has to assume some probabilistic differential transition at round 29 in order to control the avalanche in round 32), but the attack works and requires less than 2^{30} encryptions.

4 The New Related-Key Attack on GOST with Secret S-boxes

In this section we present our new attack on GOST with secret S-boxes. The attack uses several variants of the related-key differential presented in Section 3 to gradually recover the bits of the subkey K_1 (used at the last round) and the secret S-boxes. Once all the S-boxes and the full subkey K_1 are recovered (up to a few candidates), the other subkeys can be recovered in a similar way with a lower complexity, by attacking a reduced-round variant of GOST and using the knowledge of the S-boxes. For the sake of simplicity, we present the attack in the case where all S-boxes of GOST are permutations (see Section 3.4 regarding non-invertible S-boxes), and use adaptively chosen plaintext queries in order to reduce the number of related-keys used in the attack. We describe the modifications needed for using only chosen plaintexts (which were fully verified experimentally) in Section 4.6.

This section is organized as follows. In Section 4.1 we describe the strategy we use to recover the S-boxes. In Sections 4.2, 4.3, 4.4, and 4.5 we present the attack (divided into four main steps for the sake of convenience), and in Section 4.6 we report on the experimental verification of the attack. The code we use in the attack is enclosed to the paper and will be made publicly available.

4.1 The Strategy Used for S-box Recovery

In order to recover the secret S-boxes, we examine the last round of encryption (i.e., round 32). As we shall see in Section 4.2, the related-key differential allows us to find pairs (v_i, v_i') of inputs/outputs of the round function of round 32, for which we know the inputs and the XOR difference between the outputs. We claim that given a secret S-box $S : \{0,1\}^n \to \{0,1\}^n$, the number of such random pairs (v_i, v_i') needed to recover S, up to an XOR of all outputs of S with the same constant (which we cannot distinguish since at the output of S, we know only XOR differences), is $O(n2^n)$. Indeed, we can use the following simple algorithm to recover S.

First,[3] we assume w.l.o.g. that $S(0) = 0$ (since we recover S only up to an XOR with a constant). Secondly, we sort the pairs (v_i, v_i') according to v_i, and

[3] We alert the reader that this algorithm is different (and much simpler) than the algorithm presented in [18]. The reason for the difference is that in our case we

look for pairs of the form $(v_i, v'_i) = (0, x)$. The assumption $S(0) = 0$ implies $S(x) = S(v_i) \oplus S(v'_i)$. Then, we look for pairs of the form $(v_j, v'_j) = (x, y)$, which yield $S(y) = S(v_j) \oplus S(v'_j)$, where $S(v_j) = S(x)$ was already found at the previous stage. We continue in this fashion until all values $S(\cdot)$ are recovered.

When does this process recover the full S-box? Let G be a graph whose vertex set is $\{0,1\}^n$, where v_i and v'_i are connected by an edge if the pair (v_i, v'_i) exists in our data set. The process recovers S if and only if the graph is connected. It is well-known that a graph on 2^n vertices that has $m = 2^{n-1}(n + c)$ edges which are chosen uniformly at random, is connected with probability that tends to $e^{-e^{-c}}$ as $n \to \infty$ (see, e.g., [19, Theorem 4.1]). Hence, the process is expected to recover the full S-box with a high probability, once significantly more than $n2^{n-1}$ pairs (which is equal to 32 for $n = 4$) are given.

In our case, the value of n is relatively small and the pairs are not random (as they stem from plaintext pairs that satisfy a related-key differential). Hence, the number of required pairs may be somewhat larger than in the general asymptotic result. Our experiments (presented in Section 4.6) show that on average, with 256 pairs, the S-box is recovered with a fairly high probability.

The S-box recovery algorithm is described in Algorithm 1, where at the j'th iteration, we find the value of $S(v)$ for each vertex v whose distance from the vertex 0 in the graph G is j. In addition, the algorithm outputs 'Failure' if it encounters two equal pairs of inputs which lead to different output differences. Thus, the algorithm can be used also for filtering out wrong subkey guesses which lead to such a contradiction.

This simple S-box recovery algorithm is not tailor-made for our attack, and can be used in other differential-based S-box recovery attacks as well. For example, it can be used in the S-box recovery step of the attack of Bar-On et al. on GOST with palindromic key schedule [2, Sec. 5], instead of the more complex and more data-consuming algorithm of Dunkelman and Huang [18] which recovers the S-box from its difference distribution table. Indeed, once the differential part of the attack provides us with right pairs for which the actual input values are known and only the knowledge of the outputs is differential (as is commonly the case, e.g., in attacks on Feistel networks), there is no need to pass through the difference distribution table and our algorithm is sufficient. In the specific case of the attack of [2], this does not affect the overall complexity since the S-box recovery part is not the heaviest part of the attack. However, this might have effect in other cases.

4.2 First Stage of the Attack – Recovering Two S-boxes

In this subsection we present the first stage of the attack. At this stage, we use the related-key differential presented in Section 3, along with a partial guess of the subkey K_1 used at round 32, to obtain 256 pairs of inputs to each of the S-boxes S_4, S_5, for which we know the output differences. This will allow us

know the inputs to the S-box and the output differences, while the algorithm of [18] assumes only knowledge of the input and output differences.

Algorithm 1 S-box Recovery.

Input: A table T of m triples (v_i, v'_i, d_i) where for each i, (v_i, v'_i) is a pair of input values to an S-box $S : \{0,1\}^4 \to \{0,1\}^4$ and $d_i = S(v_i) \oplus S(v'_i)$ is the corresponding output difference.
Sort T according to the first value.
Set $S(0) = 0$.
for all $v \in \{1, \ldots, 15\}$ **do**
  Set $S(v) = -1$.
end for
for all i such that $v_i = 0$ **do**
  Set $S(v'_i) = d_i$.
end for
for all $j \in \{1, \ldots, 15\}$ **do**
  for all $i \in \{0, \ldots, m-1\}$ **do**
    if $S(v_i) \geq 0 \wedge S(v'_i) = -1$ **then**
      Set $S(v'_i) = v_i \oplus d_i$.
    end if
    if $S(v_i) = -1 \wedge S(v'_i) \geq 0$ **then**
      Set $S(v_i) = v'_i \oplus d_i$.
    end if
    if $S(v_i) \geq 0 \wedge S(v'_i) \geq 0 \wedge S(v_i) \oplus S(v'_i) \neq d_i$ **then**
      Abort the algorithm and output 'Failure'.
    end if
  end for
end for
Output

to significantly reduce the number of possible candidates for the guessed subkey bits, and for each remaining guess, to recover these two S-boxes (up to an output XOR with a constant) using Algorithm 1.

The success probability of the attack and its complexity significantly depend on the differential properties of the S-boxes on which we make no assumptions. For the sake of convenience, in this subsection and in the following subsections we give a rough estimate of the success probability and of the complexity at the end of each step, and present the exact figures obtained experimentally in Section 4.6. When the differential we consider is clear from the context, we call a pair that satisfies it a right pair.

Step 1: Finding the first right pair. As described in Section 3.3, if we consider GOST encryptions of the same plaintext P under five related keys: $K, \{K \oplus \Omega^i_{K1}\}_{i \in \{0,1,2,3\}}$, where $\Omega^i_{K1} = (e_{31}, e_{7+i}, e_{31}, 0, 0, 0, 0, 0)$, then for most choices of the secret S-box, the following RK truncated differential holds for one of the

key differences:

$$(0,0) \xrightarrow[\Omega^i_{K1}]{\bar{p} \geq 2^{-24}} (??U00V??_x, Z???W000_x),$$

where $U, W \in \{0, 8\}, V, Z \in \{0, \ldots, 7\}$, and ? is an unknown value. To exploit the differential, we generate plaintexts P_j one by one, and ask for their encryption using the five related keys until one of the following occurs:

1. We find a plaintext P and $i \in \{0, 1, 2, 3\}$ such that $E_K(P) \oplus E_{K \oplus \Omega^i_{K1}}(P) = (??U00V??_x \parallel Z???W000)_x$. In this case we continue to the next step, in which many more right pairs will be found.
2. After the generation of 2^{27} ciphertexts, no pair that satisfies the RK truncated differential was found. We refer to such a case as a failure and abort the attack.

Assuming that $\bar{p} = 2^{-24}$, after trying 2^{24} plaintexts P_j we obtain 2^{26} pairs that satisfy the zero input difference and the key difference of one of the related-key differentials we try in parallel. Hence, with a probability of $1 - (1 - 2^{-24})^{2^{26}} \approx 1 - e^{-4} \approx 0.98$, we obtain at least one right pair. (Note that as the input difference of the differential is zero, the right pair is composed of the same plaintext, encrypted under two different keys.)

Step 2: Finding many more right pairs at a reduced cost. Once we find a single right pair, we can find many right pairs at a significantly lower cost, using the concept of *neutral bits* that was introduced by Biham and Chen [4] and used in the collision attacks on the hash functions SHA-0 and SHA-1 [4, 5]. We observe that there are many plaintext bits that have almost no effect on the first three rounds of the differential. Specifically, a change in bits 12–22 (which are included in the right half of the plaintext) and/or in bits 32–38, 52–63 (which are included in the left half of the plaintext) affect the differential at most with a very small probability. (This effect occurs only in case of a very long carry chains). Therefore, given a single right pair, we can generate 2^{30} more pairs that satisfy the first 3 rounds of the differential with a probability close to 1, by changing the value of some of the 30 neutral bits listed above. Each of these 2^{30} additional pairs satisfies the differential with probability of at least 2^{-16} (instead of 2^{-24}).

Therefore, after checking 2^{25} of the 2^{30} additional pairs, we are expected to find at least $2^{25} \cdot 2^{-16} = 2^9$ additional right pairs. Note that as the probability that a random plaintext pair satisfies the truncated ciphertext difference is 2^{-28}, with a high probability *all* pairs which remain at this stage are indeed right pairs.

Step 3: Partially guessing the subkey K_1 and recovering the S-boxes S_4, S_5. At this point, we have 256 plaintext-ciphertext pairs $(P_j, C_j), (P_j, C'_j)$ such that $E_K(P_j) = C_j, E_{K'}(P_i) = C'_j$, $K \oplus K' = \Omega^i_{K1} = (e_{31}, e_{7+i}, e_{31}, 0, 0, 0, 0, 0)$ for some $i \in \{0, 1, 2, 3\}$, and $C_j \oplus C'_j = (??U00V??_x \parallel Z???W000)_x$, where $U, W \in \{0, 8\}, V, Z \in \{0, \ldots, 7\}$, and ? is an unknown value.

We guess the 24 least significant bits of K_1 and partially decrypt the ciphertext pairs through the last round to obtain the input values to the S-boxes S_4, S_5. Assuming that the pair is a right pair, the corresponding output difference must be equal to bits 59–63,32–34 of $C_j \oplus C'_j$ (which correspond to a left rotation by 11 bits of bits 16–23 in the output of the 32'th round function), since the difference in the corresponding bits in the differential is zero (as $D_1 = 00000XY0_x$). This allows using Algorithm 1 to recover the S-boxes S_4, S_5 (up to XOR of the output with a constant), under the assumption that the subkey guess is correct.[4]

This step can be performed efficiently, such that its complexity will be less than 2^{27} encryptions, as we explain at the end of the next step.

Step 4: Eliminating wrong subkey guesses. As mentioned in Section 4.1, Algorithm 1 not only recovers the S-boxes, but also allows us to filter out wrong subkey guesses, as those guesses lead to a contradiction between the values of S.

In order to further reduce the number of possible subkey guesses, we use three additional subkey filtering steps. These steps are based on checking whether there is an addition carry that affects the input difference to one of the S-boxes S_4, S_5, S_6, and each of them can be applied to some of the right pairs.

1. Assume that the ciphertexts (C_i, C'_i) in some right pair satisfy $C_i \oplus C'_i = (??U00V??_x \parallel Z??0W000_x)$, where $U, W \in \{0, 8\}, V, Z \in \{0, \ldots, 7\}$, and ? is an unknown value. (This means that in addition to being a right pair, we require that the difference in bits 16–19 is zero.) If bits 59–62 of the difference $C_i \oplus C'_i$ are not all equal to zero, then the inputs to the S-box S_4 that correspond to C_i and C'_i cannot be equal (as bits 59–62 correspond to a left shift by 11 bits of the output of S_4). Since we assume that C_i and C'_i are equal in bits 16–19, this may happen only if in exactly one of the addition operations $C_i \boxplus K_1, C'_1 \boxplus K_1$ there is a carry into bit 16. In other words, we have either
$$(K_1 \pmod{2^{16}}) \boxplus (C_i \pmod{2^{16}}) \geq 2^{16}$$
and
$$(K_1 \pmod{2^{16}})) \boxplus (C'_i \pmod{2^{16}}) < 2^{16},$$
or vice versa. This yields an inequality of the form $a_i < K_1 \pmod{2^{16}} < b_i$.
2. By the same reasoning, if $C_i \oplus C'_i = (??U00V??_x \parallel Z?0?W000_x)$ (meaning that in addition to being a right pair, we require that the difference in bits 20–23 is zero), and bits 63 and 32–34 of the difference $C_i \oplus C'_i$ are not all equal to zero, then we have either
$$(K_1 \pmod{2^{20}}) \boxplus (C_i \pmod{2^{20}}) \geq 2^{20}$$
and
$$(K_1 \pmod{2^{20}})) \boxplus (C'_i \pmod{2^{20}}) < 2^{20},$$

[4] We note that while we can use the same strategy to obtain 256 pairs of known input values with known output differences for S_6 as well, it turns out that due to addition carries, many of these pairs are equal and so we do not obtain enough information for recovering this S-box. Instead, we recover it at a later stage.

or vice versa.

3. By the same reasoning, if $C_i \oplus C'_i = (??U00V??_x \parallel Z0??W000)_x$ (meaning that in addition to being a right pair, we require that the difference in bits 24–27 is zero), and bits 35–38 of the difference $C_i \oplus C'_i$ are not all equal to zero, then we have either

$$(K_1 \pmod{2^{24}}) \boxplus (C_i \pmod{2^{24}}) \geq 2^{24}$$

and

$$(K_1 \pmod{2^{24}})) \boxplus (C'_i \pmod{2^{24}}) < 2^{24},$$

or vice versa.

For sake of efficiency, we perform the key filtering of Step 4 before the S-box recovery process of Step 3, and divide the key guessing into several steps. That is, first we guess the 16 least significant bits of K_1 and perform the first key filtering step. Then, for the remaining values of bits 0–15 of K_1, we guess bits 16–19 of K_1 and perform the second key filtering step. Then, for the remaining values of bits 0–19 of K_1, we guess bits 20–23 of K_1 and perform the third key filtering step. For the remaining subkey values, we perform the S-box recovery procedure of Step 3 along with the additional key filtering it provides.

According to our experiments (using 100 different keys and S-boxes), about $2^{14.2}$ possible values of bits 0–23 of K_1 pass this filtering. Among these values, about $2^{7.8}$ possible values pass the additional filtering of Algorithm 1, and for each of them, we recover the S-boxes S_4 and S_5, up to XOR of the output with a constant. (Among the 100 experiments, the highest number of surviving keys was $2274 \approx 2^{11.2}$). The time complexity of this step is significantly smaller than 2^{27} encryptions.

4.3 The Second Stage of the Attack – Recovering Two Additional S-boxes

In this subsection we present the second stage of the attack. At this stage, we use a variant of the related-key differential presented in Section 3 to further reduce the number of possible values of the subkey K_1 and to recover the S-boxes S_1, S_2 (up to an output XOR with a constant).

The differential we use at this stage. In the choice of the differential, we can exploit the fact that the S-box S_4 was already recovered at the first stage.[5] As follows from the analysis presented in Section 3.3, for any input difference of S_4 of the form $2^i + 2^{i+1} + \ldots + 2^{i+\ell-1}$ (where $i \in \{0, 1, 2, 3\}$ and $\ell \in \{1, \ldots, 4-i\}$) and any output difference of the form $2^j + 2^{j+1} + \ldots + 2^{j+b-1}$ (where $j \in \{0, 1, 2, 3\}$

[5] Although theoretically S_4 depends on the 24 least significant bits of K_1, our experiments show that in most of the cases the same S-box is suggested by all the remaining keys. We thus use the S-box S_4 of the first remaining key.

and $b \in \{1, \ldots, 4-j\}$), the following 3-round iterative differential characteristic holds:

$$(0,0) \xrightarrow[\Omega^i_{K_1}=e_{16+i}]{2^{-\ell}\cdot \mathrm{DDT}_{S_4}[(i..i+\ell-1)][(j..j+b-1)]} (0, e_{27+j,27+j+1,\ldots,27+j+b-1})$$

$$\xrightarrow[\Omega^j_{K_2}=e_{27+j}]{2^{-b}} (e_{27+j,27+j+1,\ldots,27+j+b-1}, 0) \xrightarrow[\Omega^i_{K_3}=e_{16+i}]{2^{-\ell}\cdot \mathrm{DDT}_{S_4}[(i..i+\ell-1)][(j..j+b-1)]} (0,0),$$

where $\mathrm{DDT}_{S_4}[(i..i+\ell-1)][(j..j+b-1)]$ denotes the probability of the transition

$$(2^i + 2^{i+1} + \ldots + 2^{i+\ell-1}) \underset{S_4}{\longrightarrow} (2^j + 2^{j+1} + \ldots + 2^{j+b-1}).$$

Here, the characteristic of the first and the third round holds since in the modular addition operation, the difference 2^i is transformed to $2^i + 2^{i+1} + \ldots + 2^{i+\ell-1}$ with probability $2^{-\ell}$. (Note that such addition carries are not considered in Section 3.3, as there the difference is in bit e_{31} for which modular addition does not have carry).

The probability of the 3-round iterative differential

$$(0,0) \xrightarrow{\Omega_{K2}=(e_{16+i}, e_{27+j}, e_{16+i})} (0,0) \tag{1}$$

is much higher than the probability of each separate differential characteristic, since it enjoys the contributions of all differential characteristics that correspond to these values of i, j and all possible values of ℓ, b.

In order to choose the key difference of the differential, we compute a lower bound on the probability of the differential (1) for all i, j:

$$p_{i,j} = \sum_{b=1}^{4-j} 2^{-b} \left(\sum_{\ell=1}^{4-i} 2^{-\ell} \mathrm{DDT}_{S_4}[(i..i+\ell-1)][(j..j+b-1)] \right)^2 .$$

Then, we choose i, j such that $p_{i,j}$ is maximal and ask for the encryption of the same plaintext under keys with difference $\Omega_{K2} = (e_{16+i}, e_{27+j}, e_{16+i})$.

We performed an experiment with 100 randomly chosen S-boxes. The largest value $p_{i,j}$ was about 2^{-6} on average, and for 99 out of the 100 S-boxes it was larger than $2^{-7.3}$. We therefore assume that $p_{i,j} \geq 2^{-7.3}$. Using the key difference

$$\Omega_{K2} = (e_{16+i}, e_{27+j}, e_{16+i}, 0, 0, 0, 0, 0),$$

we get the 29-round related-key differential

$$(0,0) \xrightarrow[\Omega_K]{p^3_{i,j} \geq 2^{-22}} (0,0).$$

As in the differential presented in Section 3, we do not use probabilistic differential transitions in the last three rounds, in which the subkey differences are $e_{16+i}, e_{27+j}, e_{16+i}$ (respectively), and instead, we use a truncated differential based on following the possible differences.

At round 30, the input difference is zero and the subkey difference is e_{16+i}. Thus, after the key addition the difference is $00??0000_x$ with a high probability. This truncated difference is preserved by the S-box layer, and then is mapped by the left rotation to $?Y00000X_x$ (where $X \in \{0,\ldots,7\}, Y \in \{0,8\}$).

At round 31, the truncated difference after the key addition (in which the subkey difference is e_{27+j}) is $?Q00000?_x$ (where $Q \in \{0,8\}$). After the S-box layer, the difference is $??00000?_x$, which is mapped to $0000Z??W_x$ (where $W \in \{0,8\}, Z \in \{0,\ldots,7\}$) by the left rotation. This difference is copied to the right half of the ciphertext.

At round 32, the difference after the key addition (in which the subkey difference is e_{16+i}) is $00?????T_x$ (where $T \in \{0,8\}$). After the S-box layer, the difference is $00??????_x$ which is mapped by the left rotation to $?????U0V_x$ (where $V \in \{0,\ldots,7\}, U \in \{0,8\}$). This is the difference in the left half of the ciphertext.

Hence, we have a related key truncated differential on the entire cipher:

$$(0,0) \xrightarrow[\Omega_{K2}]{p^3_{i,j} \geq 2^{-22}} (?????U0V_x, 0000Z??W_x),$$

where $V, Z \in \{0,\ldots,7\}, U, Y \in \{0,8\}$, and ? is an unknown 4-bit value. We use this characteristic to recover two additional S-boxes, S_1, S_2, and to eliminate more wrong keys.

The following steps are similar to the corresponding steps of Stage 1. As there are many small differences, we provide a detailed description.

Step 1: Finding the first right pair. We generate plaintexts one by one, and ask for their encryption using the two related keys, $K, K' = K \oplus \Omega_{K2}$, until one of the following occurs:

1. We find a plaintext P such that the difference $E_K(P) \oplus E_{K \oplus \Omega_{K2}}(P)$ is of the form $(?????U0V_x \parallel 0000Z??W_x)$. In this case we continue to the next step, in which many more right pairs are found.
2. After the generation of 2^{25} ciphertexts, no pair that satisfies the RK truncated differential was found. We refer to such a case as a failure and abort the attack.

Assuming that $p^3_{i,j} = 2^{-22}$, after trying 2^{24} plaintexts P_j we obtain 2^{24} pairs that satisfy the zero input difference and the key difference of the related-key differentials. Hence, with a probability of $1-(1-2^{-22})^{2^{24}} \approx 1-e^{-4} \approx 0.98$, we obtain at least one right pair.

Step 2: Finding many more right pairs at a reduced cost. Again, once we find a single right pair, we can find many right pairs at a lower cost, using neutral bits. We observe that a change in bits 4–12 and/or in bits 44–58, 63 of the plaintext affects the differential with a very small probability. (This effect occurs only in case of a very long carry chain). Therefore, given a single right pair, we can generate 2^{25} more pairs that satisfy the first 3 rounds of the differential with a probability close to 1, by changing the value of some of the 25 neutral bits listed

above. Each of these 2^{25} additional pairs satisfies the differential with probability of about $p_{i,j}^2 \geq 2^{-14.6}$ (instead of $p_{i,j}^3$).

Therefore, after checking $2^{22.6}$ additional pairs, we are expected to find $2^{22.6} \cdot 2^{-14.6} = 2^8$ additional right pairs. Note that as the probability that a random plaintext pair satisfies the truncated ciphertext difference is 2^{-28}, with a high probability *all* pairs which remain at this stage are indeed right pairs.

Step 3: Recovering the S-boxes S_1, S_2. At this point, we have 256 plaintext-ciphertext pairs $(P_j, C_j), (P_j, C'_j)$ such that $E_K(P_j) = C_j, E_{K'}(P_i) = C'_j$, $K \oplus K' = \Omega_{K2} = (e_{16+i}, e_{27+j}, e_{16+i}, 0, 0, 0, 0, 0)$ for the chosen $i, j \in \{0, 1, 2, 3\}$, and $C_j \oplus C'_j = (?????U0V_x \parallel 0000Z??W_x)$, where $U, W \in \{0, 8\}, V, Z \in \{0, \ldots, 7\}$, and ? is an unknown value.

For each remaining value of the 12 least significant bits of K_1, we partially decrypt the ciphertext pairs through the last round to obtain the input values to the S-boxes S_1, S_2. Assuming that the pair is a right pair, the corresponding output difference must be equal to bits 47–54 of $C_j \oplus C'_j$ (which are a left rotation by 11 bits of bits 4–11, and are of the left half of the ciphertext), since the difference in the corresponding bits in the differential is zero. This allows using Algorithm 1 to recover S-boxes S_1, S_2 (up to XOR of the output with a constant), under the assumption that the subkey guess is correct.

Step 4: Eliminating wrong subkey guesses. As mentioned in Section 4.1, Algorithm 1 not only recovers the S-boxes, but also allows us to filter out wrong subkey guesses, as those guesses lead to a contradiction between the values of S.

In order to further reduce the number of possible subkey guesses, we use an additional subkey filtering step, which we apply for each of the 256 pairs.

Assume that the ciphertexts (C_i, C'_i) in some right pair satisfy $C_i \oplus C'_i = (?????U0V_x \parallel 00000??W_x)$, where $U, W \in \{0, 8\}, V \in \{0, \ldots, 7\}$, and ? is an unknown value. (This means that in addition to being a right pair, we require that the difference in bits 12–14 is zero.) If bits 55–58 of the difference $C_i \oplus C'_i$ are not all equal to zero, then the inputs to the S-box S_4 that correspond to C_i and C'_i cannot be equal (as bits 55–58 correspond to a left shift by 11 bits of the output of S_4). Since we assume that C_i and C'_i are equal in bits 12–15, this may happen only if in exactly one of the addition operations $C_i \boxplus K_1, C'_1 \boxplus K_1$ there is a carry into bit 12. In other words, we have either

$$(K_1 \pmod{2^{12}}) \boxplus (C_i \pmod{2^{12}}) \geq 2^{12}$$

and

$$(K_1 \pmod{2^{12}})) \boxplus (C'_i \pmod{2^{12}}) < 2^{12},$$

or vice versa. This yields an inequality of the form $a_i < K_1 \pmod{2^{12}} < b_i$.

For the sake of efficiency, we first perform the key filtering of Step 4, and then we perform the S-box recovery process of Step 3, along with the additional key filtering it provides. According to our experiments (using 100 randomly selected keys and S-boxes), about 1.2 keys remain out of 2^{24} possible values of the 24 least significant bits of K_1. For each of them we recover the S-boxes S_1, S_2, up to XOR of the output with a constant.

4.4 The Third Stage of the Attack – Recovering One Additional S-box

In this subsection we present the third stage of the attack. At this stage, we use another variant of the related-key differential presented in Section 3 to further reduce the number of possible values of the subkey K_1 and to recover the S-box S_7 (up to an output XOR with a constant). As this stage is very similar to the second stage, we present it briefly.

The differential we use at this stage. In the choice of the differential, we exploit the fact that the S-box S_2 was already recovered at the second stage.[6]

We choose the key difference $\Omega_{K3} = (e_{8+i}, e_{19+j}, e_{8+i}, 0, 0, 0, 0, 0)$, where (i, j) is chosen such that

$$p_{i,j} = \sum_{b=1}^{4-j} 2^{-b} \left(\sum_{\ell=1}^{4-i} 2^{-\ell} \mathrm{DDT}_{S_2}[(i..i+\ell-1)][(j..j+b-1)] \right)^2$$

is maximal. For this key difference, we obtain the 32-round related-key truncated differential

$$(0,0) \xrightarrow[\Omega_{K3}]{p_{i,j}^3 \geq 2^{-22}} (0V?????U_x, ?W0000Z?_x),$$

where $V, Z \in \{0, \ldots, 7\}, U, Y \in \{0, 8\}$, and ? is an unknown 4-bit value.

Steps 1,2: Finding 256 right pairs. To find one right pair, we generate plaintexts one by one, and ask for their encryption using the two related keys, $K, K' = K \oplus \Omega_{K3}$, until either we find a right pair with respect to the differential, or we try 2^{26} pairs and don't find a right one (in which case we abort the attack and declare 'Failure'). By the same analysis as in the second stage, with probability of 98% we obtain a right pair after trying at most 2^{24} plaintexts.

To find many additional right pairs at a reduced cost, we again use neutral bits. We observe that a change in bits 0–4, 28–31 and/or in bits 36–50, 63 of the plaintext affects the differential at most with a very small probability. Therefore, given a single right pair, we can generate 2^{25} more pairs that satisfy the first 3 rounds of the differential with a probability close to 1, by changing the value of some of these 25 neutral bits. Each of these 2^{25} additional pairs satisfies the differential with probability of about $p_{i,j}^2 \geq 2^{-14.6}$. Therefore, after checking $2^{22.6}$ additional pairs, we are expected to find $2^{22.6} \cdot 2^{-14.6} = 2^8$ additional right pairs. As above, with a high probability *all* pairs which remain at this stage are indeed right pairs.

[6] Although S_2 depends on the 12 least significant bits of K_1, since only about 1.2 keys remain out of 2^{24} possible values of the 24 least significant bits of K_1, we assume that the S-box S_2 suggested by all remaining keys is the same. This assumption was verified experimentally. We thus use the S-box S_2 suggested by the first remaining key.

Steps 3,4: Guessing the rest of the bits of K_1, and recovering S-box S_7. At this point, we have 256 plaintext-ciphertext pairs $(P_j, C_j), (P_j, C'_j)$ such that $E_K(P_j) = C_j, E_{K'}(P_i) = C'_j$, $K \oplus K' = \Omega_{K3} = (e_{8+i}, e_{19+j}, e_{8+i}, 0, 0, 0, 0, 0)$ for the chosen $i, j \in \{0, 1, 2, 3\}$, and $C_j \oplus C'_j = (0V?????U_x \,\|?W0000Z?_x)$, where $U, W \in \{0, 8\}, V, Z \in \{0, \ldots, 7\}$, and ? is an unknown value.

We guess the 8 most significant bits of K_1, and for each remaining value of the 24 least significant bits of K_1 we get a candidate for the entire subkey K_1. For each candidate, we partially decrypt the ciphertext pairs through the last round to obtain the input values to the S-box S_7. Assuming that the pair is a right pair, the corresponding output difference must be equal to bits 39–42 of $C_j \oplus C'_j$ (which are a left rotation by 11 bits of bits 28–31, and are of the left half of the ciphertext), since the difference in the corresponding bits in the differential is zero. This allows using Algorithm 1 to recover S-box S_7 (up to XOR of the output with a constant), under the assumption that the subkey guess is correct. Algorithm 1 also allows us to filter out wrong subkey guesses, as those guesses lead to a contradiction between the values of S_7.

According to our experiments (using 100 different keys and S-boxes), $2^{4.1}$ suggestions for the full subkey K_1 remain, and for each of them, we obtain a unique suggestion for the S-boxes S_1, S_2, S_4, S_5, and S_7, up to XOR of the output with a constant.

4.5 The Fourth Stage of the Attack – Recovering the Rest of the S-boxes and Eliminating More Wrong Candidates of K_1

In this subsection we present the fourth stage of the attack. At this stage, we reuse ciphertext pairs obtained at the previous stages to fully recover the S-boxes and the subkey K_1.

Step 1: Recovering the rest of the S-boxes (S_0, S_3, S_6), up to XOR of the output with a constant. While neither of the three differentials used in the attack does not provide enough data for recovering S-boxes S_0, S_3, S_6, we can recover them by combining ciphertext pairs obtained from two differentials.

1. At the first and second stages together, we obtain 512 plaintext-ciphertext pairs $(P_j, C_j), (P_j, C'_j)$ such that bits 12–15 (which form the input to S_3) of $C_j \oplus C'_j$ are of the form $W \in \{0, 8\}$ (at the first stage) or $Z \in \{0, \ldots, 7\}$ (at the second stage). For each remaining candidate of K_1, we partially decrypt these ciphertext pairs through the last round to obtain the input values to the S-box S_3. Assuming that the pair is a right pair, the corresponding output difference must be equal to bits 55–58 of $C_j \oplus C'_j$ (which are a left rotation by 11 bits of bits 12–15, and are of the left half of the ciphertext), since the difference in the corresponding bits in both differentials is zero. This allows using Algorithm 1 to recover the S-box S_3 (up to XOR of the output with a constant), under the assumption that the subkey guess is correct, and also to filter out wrong subkey guesses, as those guesses lead to a contradiction between the values of S_3.

2. Similarly, at the first and the third stages together, we obtain 512 pairs $(P_j, C_j), (P_j, C'_j)$ such that the bits 24–27 (which form the input to S_6) of $C_j \oplus C'_j$ can obtain any value (at the first stage) or are of the form $W \in \{0, 8\}$ (at the third stage). This allows using Algorithm 1 to recover the S-box S_6 (up to XOR of the output with a constant), under the assumption that the subkey guess is correct, and also to filter out wrong subkey guesses, as those guesses lead to a contradiction between the values of S_6.
3. Similarly, at the second and third stages together, we obtain 512 pairs $(P_j, C_j), (P_j, C'_j)$ such that the bits 0–3 (which form the input to S_0) of $C_j \oplus C'_j$ are of the form $W \in \{0, 8\}$ (at the second stage) or can obtain any value (at the third stage). This allows using Algorithm 1 to recover the S-box S_0 (up to XOR of the output with a constant), under the assumption that the subkey guess is correct, and also to filter out wrong subkey guesses.

According to our experiments, in most of the cases, at this stage a unique value of bits 0–27 of K_1 remains. (Specifically, among 100 experiments, only in a single experiment two values remained).

Step 2: Fully recovering the S-boxes, and recovering the rest of K_1 up to a few remaining candidates. Due to the differential nature of the attack, analysis of the last round recovers the S-boxes only up to XOR of the output with a constant, and also cannot recover bits 28–31 of K_1 (as these bits affect no addition carries to other S-boxes). In order to recover the missing key/S-box material, we analyze round 31. Note that at this stage, for each guess of bits 28–31 of K_1, we are able to decrypt the ciphertexts through the last round, to obtain inputs to the round function of round 31, which are correct up to XOR with the same 32-bit constant A. We recover A and filter out wrong guesses of bits 28–31 of K_1, in the following steps.

1. In the plaintext-ciphertext pairs $(P_j, C_j), (P_j, C'_j)$ obtained at the first stage, the difference at the input to the round function of round 31 is of the form $00000XY0_x$, where $X \in \{0, \ldots, 7\}$,$Y \in \{0, 8\}$. Hence, by guessing the 12 least significant bits of A and of K_2, we can obtain the input values to the S-boxes S_1, S_2 in round 31. Assuming that the pair is a right pair, the corresponding output difference must be equal to bits 15–22 of $C_j \oplus C'_j$ (which are a left rotation by 11 bits of bits 4–11), since the difference in the corresponding bits in the differential is zero. As the S-boxes S_1, S_2 are known (up to XOR of the output with a constant, which does not affect output differences), this provides a very strong filtering condition on the guessed values.
   For the sake of efficiency, we apply this filtering in a two-stage process. First we guess bits 0–7 of A and K_2 and check the filtering condition in S_1, and then we guess bits 8–11 of A and K_2 and check the filtering condition in S_2.
2. In a similar manner, in the plaintext-ciphertext pairs $(P_j, C_j), (P_j, C'_j)$ obtained at the third stage, there is a non-zero difference at the inputs of the S-boxes S_4, S_5. This allows guessing bits 12–23 of A and K_2 and obtain an

additional strong filtering condition on the guessed values by checking output differences of S_4, S_5 (whose inputs are known for each guess of the bits of A, K_2).

3. Finally, in the plaintext-ciphertext pairs $(P_j, C_j), (P_j, C'_j)$ obtained at the second stage, there is a non-zero difference at the inputs of the S-boxes S_6, S_7. This allows guessing bits 24–31 of A and K_2 and obtaining an additional strong filtering condition on the guessed values by checking output differences of S_6, S_7 (whose inputs are known for each guess of the bits of A, K_2).

Step 3: Discarding the remaining wrong candidates. We remain with a few values of the most significant bits of A and of K_2 which cannot be recovered by examining merely rounds 31,32, due to the differential nature of the attack. In order to recover them, we analyze round 30. Note that at this stage, we can decrypt the ciphertexts through rounds 32,31 to obtain inputs to the round function of round 30, up to a few possible values which stem from the remaining possible values for A and K_2. Since these values are different from each other on the most significant bits, we need to perform the filtering using differences between inputs to S-box S_7 in round 30.

In a similar manner to *Step 2*, we divide the elimination of the subkey K_3 used at round 30 into three steps, using the three differentials. First, in the third differential we use, at round 30, the S-box S_2 has a non-zero input difference. Hence, we can guess bits 0–11 of K_3, compute the inputs to this S-box, and check whether its output difference is equal to bits 51–54 of the difference at the input to round 30 (which correspond to a left rotation by 11 bits of bits 8–11). By the same reasoning as above, this equality must hold for any right pair with respect to the third differential. As the S-box S_2 is known, this provides us with a strong filtering condition on bits 0–11 of K_3. Then, in the second differential, at round 30, the S-box S_4 has non-zero input difference, which provides a filtering condition on bits 12–19 and the remaining values of bits 0–11. Finally, in the first differential, at round 30, the S-box S_7 has non-zero input difference, which provides a filtering condition on bits 20–31 and the remaining values of bits 0–19.

This completes the recovery of bits 0–27 of the subkey K_1, and the S-boxes 0–6. In addition, for each possible value of bits 28–31 of the subkey K_1, we recover S_7 and the subkey K_2, up to a swap between their most significant bits (which we cannot recognize). Our experiments show that in about 74% of the cases (i.e., in 65 out of the 88 successful experiments), 8 possible vales of bits 28–31 of K_1 remain, and in the rest of the cases, all of the 16 possible values of these four bits remain. As all the S-boxes are known at this stage, the few remaining subkey candidates can be easily discarded by attacking a reduced-round version of GOST.

This stage does not require additional data, and is significantly faster than the previous steps.

	1st stage	2nd stage	3rd stage	4th stage	Overall
Success rate	97/100	94/97	92/94	88/92	88%
Data complexity	$2^{22.2}$	$2^{22.4}$	$2^{23.2}$	0	$2^{24.2}$
Time complexity	$2^{22.2}$	$2^{22.4}$	$2^{23.2}$	negligible	$2^{24.2}$
Memory complexity	$2^{9.5}$	2^{9}	2^{9}	0	$3 \cdot 2^9 = 2^{10.6}$

Table 2: The success rate and the data, time, and memory complexities of the attack (average over 100 experiments), using 7 related keys and 256 right pairs for each characteristic.

4.6 Experimental Verification of the Attack

In this section we describe the experimental verification of the full attack, using 100 different randomly chosen keys and randomly chosen S-boxes. (All S-boxes were chosen to be permutations, following the discussion in Section 3.4).

General information. The code for the experiments is written in C++, uses a Microsoft Visual C++ (MSVC) compiler, and the operating system we use is Windows. The 100 experiments together took 1121 seconds on a single PC, where the longest experiment took 91 seconds and the median experiment took 7 seconds.

Results. Table 2 describes the success rate and the average complexity of the experiments, for each of the four stages of the attack. The maximal time complexity of the full attack was 2^{27} encryptions, while the minimum was $2^{18.7}$ encryptions. The bulk of the memory is used to store the 2^9 ciphertexts (i.e., 2^8 pairs) at each stage and the remaining keys. Our experiments show that on average, after the first stage there are about $2^{7.8}$ remaining keys. (The highest number of remaining keys after the first stage was about $2^{11.2}$). Therefore, the memory required for the first stage is about $2^9 + 2^{7.8} \approx 2^{9.5}$ 64-bit blocks on average (with a maximum of about $2^9 + 2^{11.2} \approx 2^{11.4}$ 64-bit blocks). After the second and the third stages only a few key candidates remain, and therefore, a few memory cells are sufficient for storing them.

The number of related keys required in the attack. To minimize the number of related keys, we used in the experiments adaptive chosen plaintexts, as the key differences at the second and the third stages are chosen according to the previous stages. Our attack uses 7 related keys: four key differences for the first characteristic, and one for each of the two additional characteristics.

One can minimize the number of related keys used in the first stage, by trying the four options one by one (instead of trying them in parallel). This method uses about 4.2 related keys on average. On the other hand, the data and the

CP/ACP	ACP	ACP	CP
Number of RK	4.2	7	13
Data and time complexity	$2^{26.3}$	$2^{24.2}$	$2^{25.1}$

Table 3: A comparison between chosen plaintext (CP) and adaptively chosen plaintext (ACP) modes, in terms of the number of related keys used and the average complexity.

Number of right pairs	128	192	256	384	512
Success rate	84%	88%	88%	91%	83%
Data and time complexity	$2^{23.9}$	$2^{24.2}$	$2^{24.2}$	$2^{24.5}$	2^{25}

Table 4: The effect of the number of right pairs on the success rate and on the average data and time complexity of the full attack.

time complexity of the first stage are somewhat increased: the average is $2^{26.3}$ encryptions and the maximum is $2^{29.7}$ encryptions.

We performed also experiments of the attack in the chosen plaintext model, by using 13 related keys (i.e., trying the four options of the key difference in round 2 for each characteristic). The success rate of the full attack was 89%, and the time and the data complexity of the full attack was about $2^{25.1}$ encryptions on average, with a maximum of $2^{27.3}$ encryptions. The memory complexity was the same as in the adaptive chosen plaintext model. These resuls are summarized in Table 3.

The number of right pairs. We also examined the effect of the number of right pairs we use for each characteristic on the success rate and on the data and time complexity. Table 4 reports the results for some values of the number of right pairs. While it may seem that an increase in the amount of right pairs should increase the success rate, this is not always the case, since once the data complexity is increased, more wrong pairs pass the filtering and undermine the attack. In particular, increasing the number of right pairs from 256 to 512 decreases the success rate of the attack. Switching the algorithm to accept the majority of the suggestions (e.g., by discarding inconsistent pairs) would result in a higher success rate, at the expense of significantly slowing down the attack.

5 Possible Application to GOST-based Hash Functions

In this section we discuss possible applications of our techniques to GOST-based hash functions. First, we present an extremely efficient collision attack on a hash function based on GOST in the Davies-Meyer mode, and then we

present observations that may be useful in future attacks on the actual GOST hash function [17].[7]

5.1 Collision Attack on a Davies-Meyer Construction using GOST

Davies-Meyer construction instantiated by GOST and its security. The Davies-Meyer construction is a way to transform a block cipher into a compression function. Let $E : \{0,1\}^n \cdot \{0,1\}^k \to \{0,1\}^n$ be a block cipher, one can transform it into a compression function $f : \{0,1\}^n \cdot \{0,1\}^k \to \{0,1\}^n$, which accepts an n-bit chaining value and a k-bit message block to produce a new n-bit chaining value using the transformation: $f(cv, m) = E_m(cv) \oplus cv$, where cv is the chaining value and m is the message block. Such a hash function can be used in the Merkle-Damgård mode of iteration to produce a hash function using a standard padding scheme to make the message M a multiple of k-bit blocks, and selecting an IV which is set as H_0, the first chaining value. Then, take the padded message $M' = (M_1 || \ldots || M_t)$ composed of t blocks, and set $H_0 = IV$, and iteratively apply $H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1}$ for $i = 1, \ldots, t$ until H_t, the digest is computed.

It is clear from the construction that regardless of the block cipher used, a collision in the constructions can be found in time $2^{n/2}$. Thus, the proposed construction is not secure, as the 64-bit block length of GOST allows finding a collision generically with time complexity of about 2^{32}.

Collision attack using Mendel et al.'s technique. A possible way to find a collision faster by exploiting the structure of GOST is to use the technique presented by Mendel et al. [25] in their attack on the GOST hash function. This technique assumes that for some i, the chaining value H_{i-1} is of the form (x, x), for some 32-bit value x, and aims at finding two messages M_i, M_i' such that

$$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1} = E_{M_i'}(H_{i-1}) \oplus H_{i-1} = H_i'.$$

By the structure of GOST, the 256-bit word M_i, which can be chosen by the adversary, is divided into eight 32-bit words, which are used as the subkeys of the first eight rounds of GOST. Using this property, the adversary can easily find two different values M_i, M_i' for which the intermediate value after eight rounds of GOST is (x, x). (For this, one can choose the first six subkeys arbitrarily and find the unique value (on average) of the last two subkeys that leads to (x, x) by examining 1-round GOST.) For both values, the adversary obtains a fixed point of 8-round GOST. As rounds 9–16 and 17–24 of GOST are identical to rounds 1–8, the intermediate value after 24 rounds is equal to (x, x) as well. Finally, at the last 8 rounds of GOST, the subkeys are used in a reversed order. As GOST is a Feistel construction, this means that when the intermediate value after 24 rounds is (x, x), the last 8 rounds 'undo' the previous 8 rounds, and the resulting ciphertext is a swapped version of the intermediate value after 16

[7] We remind the reader that the GOST hash function uses 4 parallel applications of the GOST block cipher, has a 256-bit chaining value and a 256-bit message block. See more details in Section 5.2.

rounds, which in our case is (x, x). Hence, $E_{M_i}(H_{i-1}) = E_{M'_i}(H_{i-1}) = (x, x)$, which yields a collision in H_i.

This attack strongly uses the assumption that $H_{i-1} = (x, x)$. In the attack of Mendel et al. on the actual GOST hash function, they check 2^{32} chaining values H_j until they obtain a chaining value of the form (x, x) and only then they apply the attack. It is unclear whether this attack strategy can be applied to find a collision with complexity of less than 2^{32} when the initial value does not have the specific form (x, x).

Efficient collision attack using our technique. We show that with our technique, we can find a collision in time of less than 2^{10} hash function evaluations, for any value of H_{i-1}.

We examine the encryption process $E_{M_i}(H_{i-1})$, where the block cipher E is GOST and the key M_i is the i'th message block that can be chosen by the adversary. We find two values $M_i = (M_{i,1}, \ldots, M_{i,8})$ and $M'_i = (M'_{i,1}, \ldots, M'_{i,8})$ such that in the encryption processes $E_{M_i}(H_{i-1})$ and $E_{M'_i}(H_{i-1})$:

1. The basic 3-round differential of GOST presented in Section 3 is satisfied in rounds 1–3,
2. We have $M_{i,j} = M'_{i,j}$ for $j = 4, 5, 6, 7, 8$, and
3. The intermediate value after 8 rounds in the encryption process $E_{M_i}(H_{i-1})$ is H_{i-1}.

Once such two values M_i, M'_i are found, we track the intermediate differences between the encryption processes $E_{M_i}(H_{i-1})$ and $E_{M'_i}(H_{i-1})$. We denote the value of the intermediate states at the end of round ℓ in the encryption processes $E_{M_i}(H_{i-1})$ and $E_{M'_i}(H_{i-1})$ by X_ℓ, X'_ℓ (respectively), and the difference between them by $\Delta X_\ell = X_\ell \oplus X'_\ell$.

First, we claim that M_i, M'_i that satisfy the three above conditions can be found in time which is significantly faster than 2^9 hash function evaluations. Indeed, as was explained in Section 3, by looking at the S-box S_7 of GOST we can choose the difference between the words $M_{i,1}, M_{i,2}, M_{i,3}$ and the corresponding words of M'_i such that the differential will hold with probability of at least 2^{-8}. (This holds for most of the possible choices of the S-box S_7.) Then, we can try 2^{10} pairs (M_i, M'_i) with the prescribed difference and check whether the intermediate difference ΔX_3 is zero. With a high probability, a pair that satisfies the differential will be found, thus achieving (1). In order to achieve (3), we can fix the value of $M_{i,4}, M_{i,5}, M_{i,6}$ and find the unique value of the words $M_{i,7}, M_{i,8}$ (on average) such that $X_8 = H_{i-1}$, by assuming that (3) holds and separately examining rounds 7 and 8 of GOST, for which we know the input and the output values. Words $M'_{i,j}$ for $4 \leq j \leq 8$ are taken to be equal to the corresponding words of M_i, in order to satisfy (2).

Due to (1), (2) and (3), we have $X'_8 = H_{i-1}$ as well. This means that H_{i-1} is a fixed point of the first 8 rounds of GOST, for both keys M_i and M'_i. Since rounds 9–16 and 17–24 of GOST are identical to rounds 1–8, it follows that $X_{24} = X'_{24} = H_{i-1}$. The zero difference between the encryption processes is preserved until the input of round 30 (as there is no subkey difference and no

state difference), and at the last three rounds, the subkeys generated by the keys M_i, M'_i (which are $M_{i,3}, M_{i,2}, M_{i,1}$ and $M'_{i,3}, M'_{i,2}, M'_{i,1}$, respectively) satisfy the subkey difference of the 3-round related-key differential. Hence, the ciphertext difference is equal to 0 (which means that we get a collision) with probability of at least 2^{-8}.

Finally, we observe that once we obtain one pair (M_i, M'_i) that satisfies the differential in rounds 1–3, we can easily generate 2^9 additional pairs that satisfy the differential by leaving $M_{i,1}, M_{i,2}, M_{i,3}$ unchanged, slightly altering $M_{i,4}, M_{i,5}, M_{i,6}$, recomputing $M_{i,7}, M_{i,8}$, and setting M'_i such that (1),(2) are satisfied. With a high probability, one of these pairs satisfies the differential in rounds 30–32, and thus, provides a collision. Thus, we obtain a collision in the hash function, in time complexity of less than 2^{10} hash function evaluations.

5.2 Observations on the GOST Hash Function

The GOST hash function. The GOST hash function, defined in the standards GOST R 34.11-94 [17] and GOST 34.311-95, was the Russian Federation hash function standard for almost 20 years, until it was replaced by the hash function Streebog in 2012. It is a 256-bit hash function based on a parallel application of four instances of the GOST block cipher to related inputs, followed by a mixing transformation. The exact description of the hash function is rather complex. We briefly describe the details required for our observations, and refer the reader to [17] for the complete specification.

Similarly to the construction described above, the message M is padded to M' whose length is a multiple of 256 bits, then M' is divided to 256-bit blocks $M' = (M_1||\ldots||M_t)$, and then a serial application of $f : \{0,1\}^{256} \times \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ is used to compute $H_i = f(M_i, H_{i-1})$ for $i = 1, \ldots, t$. We omit the way in which the digest is generated from H_t, as we concentrate on the compression function f.

The structure of f is the following. First, the 256-bit word M_i is used to generate four 256-bit words $K_{i,1}, \ldots, K_{i,4}$, using XORs of iterated applications of a linear transformation A that mixes 64-bit chunks of M_i and a permutation P that changes the order of 8-bit chunks of the resulting words. Then, H_{i-1} is partitioned into 64-bit blocks as $H_{i-1} = (H_{i-1,1}||\ldots||H_{i-1,4})$ and the GOST block cipher is applied four times in parallel, to obtain the 256-bit value $S_i = (GOST_{K_{i,1}}(H_{i-1,1})||\ldots||GOST_{K_{i,4}}(H_{i-1,4}))$. Finally, an LFSR-based mixing function χ which depends on S_i but also on M_i and H_{i-1} is used to produce $H_i = \chi(H_{i-1}, M_i, S_i)$.

The attack of Mendel et al. At Crypto'2008, Mendel et al. [25] presented an attack on the GOST hash function, which allows finding a collision with time complexity of 2^{105} hash function evaluations and finding a preimage with time complexity of 2^{192} hash function evaluations. While the attack is far from being practical, it breaks the collision and preimage resistance of the GOST hash function.

The basic observation behind the attack is that when $H_{i-1,1}$ is of the form (x, x), one can easily generate many 256-bit keys $K_{i,1}$ such that $H_{i-1,1}$ is a fixed point of $GOST_{K_{i,1}}(\cdot)$. Specifically, one can generate up to about 2^{192} such keys, with a cost of a few operations for generating each key (as one can arbitrarily choose the first six 32-bit words of $K_{i,1}$ and find the unique value of the last two words which yields a fixed point by analyzing rounds 7,8 separately).

As $K_{i-1,1}$ is obtained from M_i by a simple linear transformation, the adversary can compute 2^{192} values of M_i which lead to the same 64 initial bits of S_i, with a cost of a few operations for generating each such value M_i. If the value H_i would depend only on S_i and H_{i-1} like in the Davies-Meyer construction, one could obtain a collision in H_i after observing 2^{96} such values M_i, by the birthday paradox. (In fact, if this was the case, one could use the attack of Dinur et al. on GOST [12] to obtain 2^{64} values of M_i which lead to the same 128 initial bits of S_i in time 2^{64}, by combining multi-collisions in two instances of GOST, and then obtain a collision in H_i in time 2^{64} using the birthday paradox). Mendel et al. show that by adding a 64-bit linear restriction, one can leverage the collision in 64 bits of S_i into a collision in 64 bits of H_i despite the existence of the mixing function χ. (This additional restriction is the reason why the strategy of combining multi-collisions in two instances of GOST cannot be applied, due to lack of degrees of freedom). The overall complexity of the collision attack of Mendel et al. is higher than 2^{96} (specifically, it is 2^{105}) due to the need to overcome the finalization that was not described above. The preimage attack of Mendel et al. is based on the same technique.

The possibility of applying our technique to the GOST hash function. As described in Section 5.1, our technique can be used to find efficiently pairs $K_{i,1}, K'_{i,1}$ such that $GOST_{K_{i,1}}(H_{i-1,1}) = GOST_{K'_{i,1}}(H_{i-1,1})$. Each such pair can be found at the cost of less than 2^{10} GOST evaluations, even if $H_{i-1,1}$ does not have the form (x, x). A natural strategy for applying this technique to attack the GOST hash function is to find pairs of values (M_i, M'_i) such that collisions in two instantiations of GOST occur simultaneously, thus yielding a collision in 128 bits of H_i.

The first obstacle on the way of this strategy is that the words $K_{i,j}$ are related through the transformations A, P, and thus, it is not clear a-priori that one can obtain differences that comply with our related-key differential in both $K_{i,1}$ and $K_{i,2}$ simultaneously. However, it turns out that the exact structure of A, P does allow to achieve this, at least to some extent. Specifically, let us call a difference of the form e_{31}, e_7, e_{31} in three subsequent subkey words or a cyclic rotation of it a local collision, and observe that if the subkey difference can be decomposed as the sum of two such local collisions (probably, in different words), then the probability of the related-key differential is only squared, which still allows finding collisions efficiently in many cases. Examination of A, P shows that there are many ways to choose a difference in M_i such that in two of the words $K_{i,j}$ the difference will form a local collision. For example, if the nonzero bits in the difference in M_i are the most significant bits of xi_4, xi_5, xi_8 (using the notation of [17]), then the differences in the words $K_{i,1}$ and $K_{i,2}$ form local

collisions and the difference in the word $K_{i,3}$ is the sum of two local collisions. This may allow finding many pairs of values (M_i, M_i') which lead to a collision in the 128 initial bits of S_i.

The second obstacle in the way of this strategy is the mixing transformation χ. Examination of the structure of χ (or more specifically, of the transformation $\Psi^{-12}(\Delta(M_i))$ which affects it; note that the effect of H_{i-1} can be neglected, as we assume that there is no difference in H_{i-1}) shows that for the input difference described above, the partial collision in S_i does not lead to a collision in any of the four 64-bit parts of H_i, due to the mixing. A possible way to overcome this obstacle is to modify the difference $\Delta(M_i)$ in such a way that the effect of $\Psi^{-12}(\Delta(M_i))$ will be smaller. For example, if the nonzero bits in $\Delta(M_i)$ are the most significant bits of xi_9, xi_{11}, xi_{18}, then a collision in $S_{i,4}$ leads to a collision in $H_{i,4}$. For this value of $\Delta(M_i)$, the difference in $K_{i,4}$ forms the sum of two local collisions, and thus, one may find collisions in $S_{i,4}$ (and thus, also in $H_{i,4}$) with a reduced cost. However, we could not find a way to obtain collisions in two words of H_i simultaneously, due to the effect of χ.

Finally, the third obstacle is that while the attack of Mendel et al. uses huge multi-collisions due to GOST's mode of iteration (that contains a an additive checksum of all chaining values H_i), our attack provides us only with pairs of values of M_i which yield collisions in part of the state H_i. This makes leveraging a partial collision into a full collision significantly more expensive.

To summarize, it seems that the mixing function χ thwarts the natural strategy of using our technique to attack the GOST hash function. However, more sophisticated applications might be possible, especially as the structure of A, P allows obtaining local collisions or their combinations in several words $K_{i,j}$ simultaneously.

6 Summary and Conclusions

In this paper we presented a related-key attack on GOST with secret S-boxes, which is the first known attack on the full GOST with secret S-boxes that works for all keys. We fully verified our attack, and it runs in about 11 seconds on a PC, with a success rate of 88%. The main technique we used in the attack is a new related-key differential of GOST, which is based on 3-round local collisions, in the spirit of the collision attacks on the SHA-0 and SHA-1 hash functions. We showed that our techniques apply to other variants of GOST as well, such as the block cipher GOST2 and a Davies-Meyer hash function based on GOST. The main open question for further research is, whether our techniques can be applied to attack the GOST hash function.

References

1. Ashur, T., Bar-On, A., Dunkelman, O.: Cryptanalysis of GOST2. IACR Trans. Symmetric Cryptol. **2017**(1), 203–214 (2017)

2. Bar-On, A., Biham, E., Dunkelman, O., Keller, N.: Efficient slide attacks. J. Cryptol. **31**(3), 641–670 (2018)
3. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. Cryptol. **7**(4), 229–246 (1994)
4. Biham, E., Chen, R.: Near-collisions of SHA-0. In: Advances in Cryptology — Proceedings of CRYPTO 2004. Lecture Notes in Computer Science, vol. 3152, pp. 290–305. Springer (2004)
5. Biham, E., Chen, R., Joux, A.: Cryptanalysis of SHA-0 and reduced SHA-1. J. Cryptol. **28**(1), 110–160 (2015)
6. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptol **4**(1), 3–72 (1991)
7. Biryukov, A., Nikolic, I.: Complementing Feistel ciphers. In: proceedings of FSE 2013. Lecture Notes in Computer Science, vol. 8424, pp. 3–18. Springer (2013)
8. Biryukov, A., Wagner, D.A.: Slide attacks. In: proceedings of FSE 1999. Lecture Notes in Computer Science, vol. 1636, pp. 245–259. Springer (1999)
9. Chabaud, F., Joux, A.: Differential collisions in SHA-0. In: Advanced in Cryptology — proceedings of CRYPTO 1998. Lecture Notes in Computer Science, vol. 1462, pp. 56–71. Springer (1998)
10. Courtois, N.: An improved differential attack on full GOST – extended version (2012), IACR Cryptology ePrint Archive, 2012/138
11. Courtois, N.T.: An Improved Differential Attack on Full GOST. In: The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday. Lecture Notes in Computer Science, vol. 9100, pp. 282–303. Springer (2016)
12. Dinur, I., Dunkelman, O., Shamir, A.: Improved attacks on full GOST. In: proceedings of FSE 2012. Lecture Notes in Computer Science, vol. 7549, pp. 9–28. Springer (2012)
13. Dmukh, A., Dygin, D., Marshalko, G.: A lightweight-friendly modification of GOST block cipher (2015), IACR Cryptology ePrint Archive, 2015/65
14. Dmukh, A., Trifonov, D., Chookhno, A.: Modification of the key schedule of the 2-GOST block cipher and its implementation on FPGA. J. Comput. Virol. Hacking Tech. **18**(1), 49–59 (2022)
15. Dolmatov, V., E.: RFC 8891, GOST R 34.12-2015: Block cipher "Magma" (2020), https://www.ietf.org/rfc/rfc8891.pdf
16. Dolmatov, V.: RFC 5830, GOST 28147-89: Encryption, decryption, and message authentication code (MAC) algorithms (2010), https://www.rfc-editor.org/rfc/rfc5830.html
17. Dolmatov, V.: RFC 5831, GOST R 34.11-94: Hash function algorithm (2010), https://datatracker.ietf.org/doc/html/rfc5831
18. Dunkelman, O., Huang, S.: Reconstructing an S-box from its Difference Distribution Table. IACR Trans. Symmetric Cryptol. **2019**(2), 193–217 (2019)
19. Frieze, A., Karoński, M.: Introduction to Random Graphs. Cambridge University Press (2015)
20. Isobe, T.: A Single-Key Attack on the Full GOST Block Cipher. J. Cryptol. **26**(1), 172–189 (2013)
21. Kelsey, J., Schneier, B., Wagner, D.A.: Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In: Advances in Cryptology — Proceedings of CRYPTO 1996. Lecture Notes in Computer Science, vol. 1109, pp. 237–251. Springer (1996)

22. Kim, J., Hong, S., Preneel, B., Biham, E., Dunkelman, O., Keller, N.: Related-key boomerang and rectangle attacks: Theory and experimental analysis. IEEE Trans. Inf. Theory **58**(7), 4948–4966 (2012)
23. Knudsen, L.R.: Cryptanalysis of LOKI91. In: proceedings of AUSCRYPT 1992. Lecture Notes in Computer Science, vol. 718. Springer (1993)
24. Ko, Y., Hong, S., Lee, W., Lee, S., Kang, J.: Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST. In: proceedings of FSE 2004. Lecture Notes in Computer Science, vol. 3017, pp. 299–316. Springer (2004)
25. Mendel, F., Pramstaller, N., Rechberger, C., Kontak, M., Szmidt, J.: Cryptanalysis of the GOST Hash Function. In: Wagner, D.A. (ed.) Advances in Cryptology - Proceedings of CRYPTO 2008. Lecture Notes in Computer Science, vol. 5157, pp. 162–178. Springer (2008)
26. Pudovkina, M.A., Khoruzenko, G.I.: An attack on the GOST 28147-89 block cipher with 12 related keys. Mathematical Aspects of Cryptography (Russian) **4**(2), 127–152 (2013)
27. Pudovkina, M.: A Related-Key Attack on Block Ciphers with Weak Recurrent Key Schedules. In: proceedings of FPS 2011. Lecture Notes in Computer Science, vol. 6888, pp. 90–101. Springer (2011)
28. Rudskoy, V.: On zero practical significance of "Key recovery attack on full GOST block cipher with zero time and memory", IACR Cryptology eprint archive, 2010:111
29. Saarinen, M.J.: A chosen key attack against the secret S-boxes of GOST (1998), IACR Cryptology ePrint Archive, 2019/540
30. Schneier, B.: Applied Cryptography, Second Edition. John Wiley & Sons (1996)
31. Seki, H., Kaneko, T.: Differential Cryptanalysis of Reduced Rounds of GOST. In: Proceedings of SAC 2000. Lecture Notes in Computer Science, vol. 2012, pp. 315–323. Springer (2001)
32. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Advances in Cryptology — Proceedings of CRYPTO 2017, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 570–596. Springer (2017)
33. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Advances in Cryptology - Proceedings of CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 17–36. Springer (2005)
34. Zhao, X., Guo, S., Zhang, F., Wang, T., Shi, Z.J., Ma, C., Gu, D.: Algebraic fault analysis on GOST for key recovery and reverse engineering. In: Proceedings of FDTC 2014. pp. 29–39. IEEE Computer Society (2014)