Improved Attacks on Full GOST

Itai Dinur[1], Orr Dunkelman[1,2] and Adi Shamir[1]

[1] Computer Science department, The Weizmann Institute, Rehovot, Israel
[2] Computer Science Department, University of Haifa, Israel

Abstract. GOST is a well known block cipher which was developed in the Soviet Union during the 1970's as an alternative to the US-developed DES. In spite of considerable cryptanalytic effort, until very recently there were no published single key attacks against its full 32-round version which were faster than the 2^{256} time complexity of exhaustive search. In February 2011, Isobe used in a novel way the previously discovered reflection property in order to develop the first such attack, which requires 2^{32} data, 2^{64} memory and 2^{224} time. Shortly afterwards, Courtois and Misztal used a different technique to attack the full GOST using 2^{64} data, 2^{64} memory and 2^{226} time. In this paper we introduce a new fixed point property and a better way to attack 8-round GOST in order to find improved attacks on full GOST: Given 2^{32} data we can reduce the memory complexity from an impractical 2^{64} to a practical 2^{36} without changing the 2^{224} time complexity, and given 2^{64} data we can simultaneously reduce the time complexity to 2^{192} and the memory complexity to 2^{36}.

Keywords: Block cipher, cryptanalysis, GOST, reflection property, fixed point property, 2D meet in the middle attack

1 Introduction

During the 1970's, the US decided to publicly develop the Data Encryption Standard (DES), which was the first standardized block cipher intended for civilian applications. At roughly the same time, the Soviet Union decided to secretly develop GOST [10], which was also supposed to be used in civilian applications but in a more controlled way. The general design of GOST was finally published in 1994, but even today some of the crucial elements (such as the choice of Sboxes) do not appear in the public description, and a different choice can be made for each application.

The overall design of GOST is similar to that of DES: Both of them are Feistel structures over 64-bit blocks, in which we repeatedly process the right half of the block, XOR the result to the left half, and swap the two halves. In the case of GOST, the processing consists of adding (modulo 2^{32}) a 32-bit round key to the right half of the block, and then applying the function f described in Figure 1. This function has an Sbox layer consisting of eight different 4×4 Sboxes, followed by a rotation of the 32-bit result by 11 bits to the left using the little-endian format (i.e. the LSB of the 32-bit word enters the rightmost entry of the first Sbox).

Full GOST has 32 rounds, and its key schedule is extremely simple: the 256-bit key is divided into eight 32-bit words $(K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8)$. Each round of GOST uses one of these words as a round key in the following order: in the first 24 rounds, the keys are used in their cyclic order (i.e. K_1 in rounds 1,9,17, K_2 in rounds 2,10,18, and so forth). In the final 8 rounds (25–32), the round keys are used in reverse order (K_8 in round 25, K_7 in round 26, and so forth).

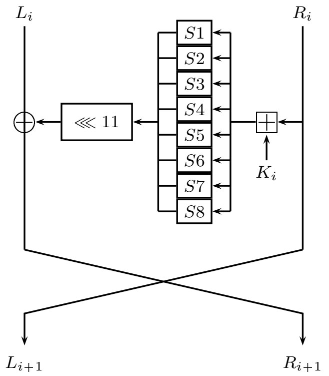

Fig. 1. One round of GOST

A major difference between the design philosophies of DES and GOST was that the publicly available DES was intentionally chosen with marginal parameters (16 rounds, 56 bit keys), whereas the secretive GOST used larger parameters (32 rounds, 256 bit keys) which seemed to offer an extra margin of security. As a result, DES was broken theoretically (by using differential and linear techniques) and practically (by using special purpose hardware) about 20 years ago, whereas all the single key attacks [1, 7, 13] published before 2011 were only applicable to reduced-round versions of the cipher. [1]

The first single key attack on the full 32-round version of GOST was published by Takanori Isobe at FSE'11 [6]. It exploited a surprising reflection property which was first pointed out by Kara [7] in 2008: Whenever the left and right halves of the state after 24 rounds are equal (which happens with probability 2^{-32}), the last 16 rounds become the identity mapping, and thus the effective number of rounds is reduced from 32 to 16. Isobe developed a new key-extraction algorithm for the remaining 16 rounds of GOST which required 2^{192} time and 2^{64} memory, and used it 2^{32} times for different plaintext/ciphertext pairs in order to get the full 256 bit key using a total of 2^{32} data, 2^{64} memory, and 2^{224} time.

[1] Attacks on full GOST in the stronger related-key model are known for about a decade, see [5, 8, 9, 12, 13].

This is much faster than exhaustive search, but neither the time complexity nor the memory complexity are even close to being practical.

Shortly afterwards, Courtois [3] published on ePrint a new attack on the full GOST. It used a very different algebraic approach, but had an inferior complexity of 2^{64} data, 2^{64} memory, and 2^{248} time. Later, Courtois and Misztal [4] described a differential attack which again used 2^{64} data and memory, but reduced the time complexity to 2^{226}.

In this paper we improve several aspects of these previously published attacks. We describe a new *fixed point property*, and show how to use either the previous reflection property or the new fixed point property in order to reduce the general cryptanalytic problem of attacking the full 32-round GOST into an attack on 8-round GOST with two known input-output pairs. We then develop a new way to extract all the 2^{128} possible values of the full 256 bit key given only two known 64-bit input-output pairs of 8-round GOST, which requires 2^{128} time and 2^{36} memory [2] (all the previously published attacks on 8-round GOST have an impractical memory complexity of at least 2^{64}). By combining these improved elements, we can get the best known attacks on GOST for the two previously considered data complexities of 2^{32} and 2^{64}. A comparison between all the previously published single key attacks on the full GOST and our new attacks is given in Table 1.

Reference	Data (KP)††	Memory	Time	Self-Similarity Property	8-Round Attack	Sboxes
[6]	2^{32}	2^{64}	2^{224}	Reflection	-	Bijective
[3]	2^{64}	2^{64}	2^{248}	Other (unnamed)	Algebraic	Russian Banks [11]
[4]	2^{64}	2^{64}	2^{226}	Differential (not based on self-similarity)	-	Russian Banks [11]
This paper	2^{64}	2^{36}	$2^{192\dagger}$	fixed point	2DMITM	any
This paper	2^{64}	2^{19}	$2^{204\dagger}$	fixed point	low-memory	any
This paper	2^{32}	2^{36}	$2^{224\dagger}$	Reflection	2DMITM	any
This paper	2^{32}	2^{19}	$2^{236\dagger}$	Reflection	low-memory	any

† The time complexity can be slightly reduced by exploiting GOST's complementation properties (as described in Appendix A)

†† Known plaintext

Table 1. Single-key Attacks on the Full GOST

[2] We can reduce the memory complexity by another factor of 2^{17} (to 2^{19}) if we are willing to increase the time by a factor of 2^{12} (to 2^{140}). This may seem like an unattractive tradeoff since the 2^{36} memory complexity is already practical, but one can argue that 2^{19} words will fit into the cache whereas 2^{36} will not, which can result in a big performance penalty.

An important observation about Isobe's attack is that it uses in an essential way the assumption that the Sboxes are invertible. Since the GOST standard does not specify the Sboxes, and there is no need to make them invertible in a Feistel structure, Isobe's attack might not be applicable to some valid incarnations of this standard. A similar problem occurs in Courtois' attacks, since he only estimates their time complexity for one particular choice of Sboxes described in [11] which is used in the Russian banking system, and it is possible that for other choices of Sboxes the complexities will be different. The new attacks described in this paper do not suffer from these limitations, since they can be applied with the same complexity to any given set of Sboxes, regardless of whether they are invertible or not and regardless of their differential properties.

2 Overview of Our New Attacks on the Full GOST

The 32 encryption rounds of GOST can be fully described using only two closely related 8-round encryption functions. Let $G_{K_{i_1},...,K_{i_j}}$ be j rounds of GOST under the subkeys $K_{i_1},...,K_{i_j}$ (where $i_1,...,i_j \in \{1,2,...,8\}$), and let (P_L,P_R) be a 64-bit plaintext, such its right half, P_R, enters the first round. Then $GOST(P_L,P_R) = G_{K_8,...,K_1} G_{K_1,...,K_8} G_{K_1,...,K_8} G_{K_1,...,K_8}(P_L,P_R)$

Our new attacks on the full GOST exploit its high degree of self-similarity using a general framework which is shared by other attacks: the algorithm of each attack consists of an outer loop which iterates over the given 32-round plaintext-ciphertext pairs, and uses each one of them to obtain suggestions for two input-output pairs for $G_{K_1,...,K_8}$. For each suggestion of the 8-round input-output pairs, we apply an 8-round attack which gives suggestions for the 256-bit GOST key. We then verify the key suggestions by using some of the other plaintext-ciphertext pairs. The self-similarity properties of GOST ensure that the 8-round attack needs to be applied a relatively small number of times, leading to attacks which are much faster than exhaustive search.

We describe several attacks on the full GOST which belong to this common framework but differ according to the property and the type of 8-round attack we use. The two self-similarity properties are:

1. The *reflection property* which was first described in [7], where it was used to attack 30 rounds of GOST (and 2^{224} weak keys of the full GOST). This property was later exploited in [6] to attack the full GOST for all keys. We describe this property again in Section 3.1 for the sake of completeness.
2. A new *fixed point property* which is described in Section 3.2.

The two properties differ according to the amount of data required to satisfy them, and thus offer different points on a time/data tradeoff curve.

Given two 8-round input-output pairs, we describe in this paper several possible attacks of increasing sophistication:

1. A very basic meet-in-the-middle (MITM) attack [2], which is described in Section 4.1.

2. An improved MITM attack, described in Section 4.2, which uses the idea of equivalent keys (first described by Isobe in [6]).
3. A low-memory attack, described in Section 5.
4. A *2-dimensional meet-in-the-middle* (2DMITM) attack, described in Section 6.

In order to attack the full GOST, we select one of the two self-similarity properties of Sections 3.1 and 3.2 to use in the outer loop of the attack according to the amount of available data. We then select one of the two 8-round attacks of Sections 5 and 6 according to the amount of available memory. The total time complexity of our attacks is calculated by multiplying the complexity of the 8-round attack by the expected number of times we have to try the self-similarity property. Altogether, we obtain four new attacks on the full GOST. In three out of the four cases, we obtain better combinations of complexities than in all the previously published attacks. In the remaining case, we use the reflection property and the low-memory 8-round attack to significantly reduce the memory requirements of Isobe's attack [6], at the expense of a small time complexity penalty. We note that the computation required by each one of our attacks can be easily parallelized, and thus using x CPUs reduces the expected running time of the attack by a factor of x.

As described in Appendix A, the time complexity of all these attacks can be slightly reduced by exploiting GOST's complementation properties. However, in some of these improved attacks we have to use chosen rather than known plaintexts, which reduces their attractiveness.

3 Obtaining 8-Round Input-Output Pairs for GOST

In this section, we describe the two self-similarity properties of GOST which we exploit in order to obtain two 8-round input-output pairs: the previously known reflection property and the new fixed point property.

The fixed point property suggests two correct 8-round input-output pairs with probability of about 2^{-64} and requires about 2^{64} known plaintext-ciphertext pairs to succeed with high probability. The reflection property suggests two correct 8-round input-output pairs with a much smaller probability of about 2^{-96}, but requires only 2^{32} known plaintext-ciphertext pairs to succeed with high probability. This implies that when about 2^{64} known plaintext-ciphertext pairs are available, it is preferable to use the fixed point property, and when about 2^{32} known plaintext-ciphertext pairs are available, we should use the reflection property.

3.1 The Reflection Property

Assume that the encryption of a plaintext P after 24 rounds of GOST gives a 64-bit value Y, such that the 32-bit right and left halves of Y are equal (i.e. $Y_R = Y_L$). Thus, exchanging the two halves of Y at the end of round 24 does not change the intermediate encryption value. In rounds 25–32, the round keys K_1–K_8 are applied in the reverse order, and Y undergoes the same operations as in rounds 17–24, but in the reverse order. As a result, the encryption of P after 32 rounds, which is the ciphertext C, is equal to its encryption after 16 rounds (see Figure 2). By guessing the state of the encryption of P after 8 rounds, denoted by the 64-bit value X, we obtain two 8-round input-output pairs (P, X) and (X, C). For an arbitrary key, the probability that a random plaintext gives such a symmetric value Y after 24 rounds is about 2^{-32}, implying that we have to try about 2^{32} known plaintexts (in addition to guessing X) in order to obtain the two pairs. Note that the reflection property actually gives us another "half pair" $(\widehat{C}, Y)$, where the 64-bit word $\widehat{C}$ is obtained from C by exchanging the right and left 32-bit halves of C, and the 32-bit right and left halves of Y are equal.[3] However, it is not clear how to exploit this additional knowledge in order to significantly improve the running time of our attacks on the full GOST which are based on the reflection property.

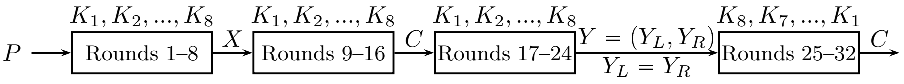

Fig. 2. The Reflection Property of GOST

3.2 The Fixed Point Property

Assume that when we encrypt a 64-bit plaintext P, we obtain P again after 8 encryption rounds. Since rounds 9–16 and 17–24 are identical to rounds 1–8, we obtain P after 16 and 24 rounds as well. In rounds 25–32, the round keys K_1–K_8 are applied in the reverse order, and we obtain some arbitrary ciphertext C (see Figure 3). The knowledge of P and C immediately gives us the 8-round input-output pairs (P, P) and $(\widehat{C}, \widehat{P})$ (in which the right and left 32-bit halves of P and C are exchanged).

For an arbitrary key, the probability that a random plaintext is a fixed point is about 2^{-64}, implying that we need about 2^{64} known plaintexts to have a single fixed point, from which we obtain the two input-output pairs needed in

[3] In our attacks, we use 8-round input-output pairs whose encryption starts with K_1 and thus need to apply the Feistel structure in the reverse order (starting from round 32) for input-output pairs obtained for rounds 25–32. Since in Feistel structures the right and left halves of the block are exchanged at the end (rather than at the beginning) of the round function, we exchange the right and left sides of the input and the output of the input-output pairs obtained for rounds 25–32. We call $(\widehat{C}, Y)$ a "half pair" since we have to guess only 32 additional bits in order to find it, once (P,C) is known.

our attack. If we have only $c \cdot 2^{64}$ known plaintexts for some fraction c, we expect this fixed point to occur among the given plaintexts with probability c, and thus the time complexity, the data complexity, and the success probability are all reduced by the same linear factor c. Consequently, it makes sense to try the fixed point based attack even when we are given only a small fraction of the entire code book of GOST. Such a graceful degradation when we are given fewer plaintexts should be contrasted with other attacks such as slide attacks, in which we have to wait for some random birthday phenomenon to occur among the given data points. Since the existence of birthdays has a much sharper threshold, the probability of finding an appropriate pair of points goes down quadratically rather than linearly in c, and thus they are much more likely to fail in such situations.

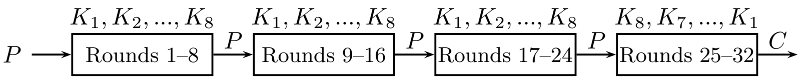

Fig. 3. The fixed point property of GOST

4 Simple Meet-in-the-middle Attacks on 8 Rounds of GOST

Meet-in-the-middle (MITM) attacks can be efficiently applied to block ciphers in which some intermediate encryption variables (bits, or combinations of bits) depend only on a subset of key bits from the encryption side and on another subset of key bits from the decryption side: the attacker guesses the relevant key bits from the encryption and decryption sides independently, and tries only keys in which the values suggested by the computed intermediate variables match up. While the full 32-round GOST seems to resist such attacks, GOST with 8 encryption rounds uses completely unrelated round keys. Thus, the full 64-bit value after 4 encryption rounds depends only on round keys K_1–K_4 from the encryption side and on round keys K_5–K_8 from the decryption side.

4.1 The Basic Meet-in-the-middle Attack

We describe how to mount a simple meet-in-the-middle attack on 8 rounds of GOST given two 8-round input-output pairs and several additional 32-round plaintext-ciphertext pairs:

1. For each of the 2^{128} possible values of K_1–K_4, encrypt both inputs and obtain two 64-bit intermediate encryption values after 4 rounds of GOST (i.e., 2^{128} intermediate values of 128 bits each). Store the intermediate values in a list,

sorted according to these 128 bits, along with the corresponding value of K_1–K_4.

2. For each of the 2^{128} possible values of K_5–K_8, decrypt both outputs, obtain two 64-bit intermediate values and search the sorted list for these two values.
3. For each match, obtain the corresponding value of K_1–K_4 from the sorted list and derive a full 256-bit key by concatenating the value of value of K_1–K_4 with the value of K_5–K_8 of the previous step. Using the full key, perform a trial encryption of several plaintexts (at least two) and return the full key, i.e., the one that remains after successfully testing the given 32-round pairs.

We expect to try about $2^{128+128-128} = 2^{128}$ full keys in step 3 of the attack, out of which only the correct key is expected to pass the exhaustive search of step 3. Including the 2^{128} 8-round encryptions which are performed in each of the first two steps of the attack, the total time complexity of the attack is slightly more than 2^{128} GOST encryptions. The memory complexity of the attack is about 2^{128} words of 256 bits. Note that it is possible obtain a time-memory tradeoff: we partition the 2^{128} possible values of K_1–K_4 into 2^x sets of size 2^{128-x} (for $0 \leq x \leq 128$), and run the second and third steps of the attack independently for each set. Thus, the memory complexity decreases by a factor 2^x to 2^{128-x}, and the time complexity increases by a factor of 2^x to 2^{128+x}.

4.2 An Improved Meet-in-the-middle Attack Using Equivalent Keys

In this section, we use a more general variant of Isobe's equivalent keys idea [6] to significantly improve the memory complexity of the attack. Both our and Isobe's MITM attacks are based on a 4-round attack that uses one input-output pair for 4 encryption rounds to find all the 2^{64} possible values of subkeys K_1–K_4 that yield this pair. However, our MITM attack is more general since we can attack all the possible incarnations of the GOST standard, whereas Isobe can only attack those which use bijective Sboxes. [4] Moreover, our MITM attack can use any two input-output pairs for 8-round GOST, regardless of how they are obtained. We can thus use the same algorithm to exploit both the reflection and the fixed point properties. On the other hand, Isobe's attack works on a single input-output pair obtained for the first 16 rounds of GOST, by guessing the intermediate values obtained after 4 and 12 rounds. Isobe's attack can thus efficiently exploit the 16-round input-output pair obtained from the reflection property, but cannot be directly applied to the two input-output pairs produced by the fixed point property.

We now describe Isobe's 4-round attack: Denote the 4-round input (divided into two 32-bit words) by (X_L, X_R) and the output by (Y_L, Y_R). Denote the middle values (after using K_2) by (Z_L, Z_R) (see Figure 4). We have the following equations on 32-bit words:

$$Z_L = X_L \oplus f(X_R \boxplus K_1)$$

[4] The Feistel structure of GOST does not require bijective Sboxes and the published standard does not discuss this issue, but all the known choices of Sboxes happen to be bijective.

$$Z_R = Y_R \oplus f(Y_L \boxplus K_4)$$

$$Y_L \oplus Z_L = f(Z_R \boxplus K_3)$$

$$X_R \oplus Z_R = f(Z_L \boxplus K_2)$$

Isobe's attack assumes bijective Sboxes (making f invertible), and finds the equivalent keys as follows:[5] for each value of K_1, K_2, compute Z_L from the first equation and Z_R from the fourth equation. From the second equation we have: $K_4 = f^{-1}(Z_R \oplus Y_R) \boxminus Y_L$ and from the third equation: $K_3 = f^{-1}(Z_L \oplus Y_L) \boxminus Y_R$.

Our 8-round attack is a variant of Isobe's MITM attack, given two 8-round input-output pairs (I, O) and (I^*, O^*):

1. For each possible value of the 64-bit word $Y = (Y_L, Y_R)$ obtained after 4 encryption rounds of the first pair:
   (a) Apply the 4-round attack on (I, Y) to obtain 2^{64} candidates for K_1–K_4.
   (b) Partially encrypt I^* using the 2^{64} candidates and store $Y^* = (Y_L^*, Y_R^*)$ in a list with K_1–K_4.
   (c) Apply the 4-round attack on (Y,O) to obtain 2^{64} candidates for K_5–K_8.
   (d) Partially decrypt O^* using each one of the 2^{64} candidates and obtain $Y^* = (Y_L^*, Y_R^*)$.
   (e) Search the list obtained in step (b) for Y^*, and test the full 256-bit keys for which there is a match.

The expected time complexity of steps (a–d) is about 2^{64} (regardless of the algorithm that is used to find the equivalent keys). The time complexity of step (e) is also about 2^{64} since we expect to try about $2^{64+64-64} = 2^{64}$ full keys. Steps (a–e) are performed 2^{64} times, hence the total time complexity of the attack is about 2^{128} GOST encryptions, which is similar to the first attack. However, the memory complexity is significantly reduced from 2^{128} to slightly more than 2^{64} words of 64 bits.

5 A New Attack on 8 Rounds of GOST with Lower Memory Complexity

Simple meet-in-the-middle attacks, such as the ones described in Sections 4.1 and 4.2 are much faster than exhaustive search for the entire 256-bit key. However, they do not fully exploit the slow diffusion of the key bits from the encryption and decryption sides. As a result, these MITM attacks use a large amount of memory to store the many intermediate encryption values obtained for all the possible values of large sets of key bits. In this section, we describe an improved attack which exploits the slow diffusion properties of 4 rounds of GOST in order to reduce the memory complexity from the impractical value of 2^{64} to the

[5] In case f is not bijective, then for a random (X_L, X_R) and (Y_L, Y_R) there exist an average of 2^{64} equivalent keys which can be found using a simple preprocessing MITM algorithm that requires about 2^{64} time and memory.

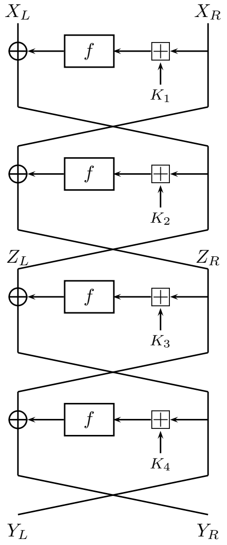

Fig. 4. Four Rounds of GOST

very practical value of 2^{19} words of memory, with a very small time complexity penalty.

Given the two input-output pairs for 8 rounds of GOST, we have a 128-bit constraint on a 256-bit key. Thus, we expect about $2^{256-128} = 2^{128}$ keys to agree with the pair. To efficiently enumerate these keys, we guess the values of the last 4 round keys (K_5, K_6, K_7 and K_8). For each value of the last 4 round keys, we partially decrypt the given outputs and obtain two input-output pairs for the first 4 rounds of GOST. Our attack retrieves all the values of the first 4 round keys that yield these 4-round input-output pairs. We expect that only one such value for K_1–K_4 exists for two arbitrary 4-round input-output pairs (note that there are likely to be input-output pairs for which the encryptions of the inputs does not match the outputs for any of the keys, and input-output pairs for which the encryptions of the inputs matches the outputs for several values of K_1–K_4).

5.1 Overview of the "Guess and Determine" Attack on 4-Round GOST

The attack is a typical "Guess and Determine" attack which traverses a tree of partial guesses for the round keys K_1–K_4 and intermediate encryption values. The tree is composed of layers of nodes $\ell_0, \ell_1, ..., \ell_k$, where each layer contains nodes which specify the potential values for a certain subset of key and intermediate encryption values denoted by $S_0, S_1, ..., S_k$, respectively. The subset of bits whose values are specified by a certain layer ℓ_i is contained in the subset of bits whose values are specified by the next layer ℓ_{i+1} (i.e. $S_i \subset S_{i+1}$). The first layer of the tree ℓ_0 specifies values for the empty subset of bits (i.e. $S_0 = \emptyset$) and contains only the "empty guess". In our attack, the nodes of the last layer of the tree (ℓ_k) contain guesses for the full key. Each node in layer ℓ_i is expanded to nodes in the next layer ℓ_{i+1} which specify all the possible values for $S_{i+1} \setminus S_i$ and the same value for S_i as the parent node. In addition, the expanded nodes of a given layer need to satisfy a predicate whose value depends on the input-output pairs and the value of the bits determined by the previous layers. Nodes which do not satisfy the predicate are not included in the next layer and are discarded. The predicates that we use in our attack check the consistency of intermediate encryption values. In other words, in each layer we expand each node by guessing the values of a small number of additional key bits and state bits that are needed to calculate some intermediate encryption bits both from the encryption and the decryption sides. We then calculate the bits by evaluating the Feistel structure from both sides on a small number of bits, compare the values obtained, and discard guesses in which the values do not match.

We traverse the partial guess tree starting from the root using DFS (which requires only a small amount of memory). Once we reach a leaf node of the last layer (a node that specifies a value for all the key bits), we check whether the key is the correct key by a sufficiently large number of trial encryptions. If we reach a leaf node which does not belong to the last layer (all of its potential children do not satisfy the predicate), we simply discard it and continue the traversal. The total number of operations performed during the traversal is proportional to the total number of nodes in the tree. However, the operations performed when expanding a single node work only on a few bits (rather than on full words). At the same time, when expanding a full path of nodes in the tree from the root to the last layer, we work on the full-size Feistel structure to obtain a guess for the full key. Hence, we estimate the time complexity of expanding a full path by a single Feistel structure evaluation on a full 64-bit input. Using this estimation, we can upper bound the time complexity of the tree traversal (in terms of Feistel structure evaluations) as the width of the tree, or the size of the layer which contains the highest number of nodes. Note that when counting the number of nodes in a layer for the time complexity analysis, we must also include the nodes expanded in the previous layer which were discarded since they do not satisfy the predicate.

5.2 Notations

Assume that we have two input-output pairs for 4 encryption rounds of GOST under the subkeys K_1, K_2, K_3, K_4. Similarly to Section 4.2, denote the input, output and middle values (after using K_2) for the first pair by (X_L, X_R), (Y_L, Y_R) and (Z_L, Z_R), respectively. For the second pair, denote these values by (X_L^*, X_R^*), (Y_L^*, Y_R^*) and (Z_L^*, Z_R^*) respectively.

Since our attack analyzes 4-bit words (which are outputs of single Sboxes), we introduce additional notations: Define the functions $f^0, f^1, ..., f^7$ where each f^i takes a 4-bit word as an input, and outputs a 4-bit word by applying Sbox i to the input. Denote by W^i the i'th bit of the 32-bit word W, and by $W^{i,j}$ the

$(j-i+1)$-bit word composed of consecutive bits of W starting from bit i and ending in bit j. We treat W as a cyclic word, for example $W^{24,3}$ contains 12 bits which are bits 24 to 31 and 0 to 3 of W. Let C_i for be the 32-bit carry word produced by the addition of the round key K_i to the corresponding state word (note that $C_i^0 = 0$ and we can ignore the last carry produced at bit 31 which has no effect on the encryption).

5.3 The Basic Procedure of the 4-Round Attack

We now describe the particular procedure that we use in the 4-round attack to expand a node in the tree. We calculate the 4-bit intermediate encryption words $Z_L^{4i+3,4i+6}$ and $Z_L^{*4i+3,4i+6}$ (where $i \in \{0, 1, ..., 7\}$):

1. Guess the 4 bits of $K_1^{4i-8,4i-5}$, which are added to the 4 data bits of $X_R^{4i-8,4i-5}$ and $X_R^{*4i-8,4i-5}$ that enter the corresponding Sbox.
2. If the 4 bits guessed in step 1 are not LSBs (least significant bits) of the full 32-bit key (i.e. $4i-8 \neq 0(mod\ 32)$), guess the carry bit entering the addition of the 4 key bits and data bits for each input-output pair (denoted by C_1^{4i-8} and C_1^{*4i-8}).
3. Add the value of $K_1^{4i-8,4i-5}$ to $X_R^{4i-8,4i-5}$ and to the (possible) carry bit C_1^{4i-8} (and perform the same operations for the other pair), and obtain the 4-bit inputs to the Sbox for each input-output pair.
4. For each input-output pair, apply the Sbox to the result of the previous step and XOR the result with the corresponding input bits of $X_L^{4i+3,4i+6}$ and $X_L^{*4i+3,4i+6}$ (obtained after the 11-bit rotation to the left).

For each input-output pair, this simple procedure gives us 4 bits of state after the Sbox layer of one round of encryption with a total of 4 or 6 bits guessed (depending on whether the guessed key bits are LSBs or not). Note that a similar procedure can be used to obtain 4 bits of state after one decryption round. In order to calculate 4 bits of Z_R after two encryption rounds for each input-output pair, we perform the following steps:

1. Run the algorithm above and obtain $Z_L^{4i+3,4i+6}$ and $Z_L^{*4i+3,4i+6}$.
2. Guess Z_L^{4i+7} and Z_L^{*4i+7}.
3. Given $Z_L^{4i+4,4i+7}$, $Z_L^{*4i+4,4i+7}$ (that enter an Sbox before the addition of K_2), use a variant of the algorithm above to obtain $Z_R^{4i+15,4i+18}$ and $Z_R^{*4i+15,4i+18}$.

Note that we need to guess the state bit of Z_L in step 2 once for each of the two input-output pairs available. The procedure described can be repeated several times in order to determine 4 bits after any number of encryption (or decryption) rounds, where the number of required guesses of key bits, carry bits and state bits increases with the number of rounds.

5.4 Optimization Methods Used in the 4-Round Attack

Since the time complexity of the attack is determined by the widest layer of the tree, we use several optimizations in order to obtain effective filtering conditions while guessing the smallest possible number of bits in each layer. These optimizations ensure that the expected number of children per node at a given layer is small, and thus reduce the expected width of the next layer. As a result, we can recover the possible keys that generate the given input-output pairs more efficiently. The optimizations that we use are described below:

1. We optimize the basic process of expanding a node described above by using a more direct approach which gives the same result. For example, we calculate the values of the 4 intermediate encryption bits, $Z_R^{31,2}$, from the encryption side of the Feistel structure. The consistency predicate on these bits can now be viewed as an equation on 4 bits of K_4 ($K_4^{20,23}$) from the decryption side. In the basic approach we solve this equation by exhaustive search on the 16 possible values of $K_4^{20,23}$. Instead, we precompute and store the solutions to the equation for all its 2^{20} possible values, and use this small precomputed table to directly derive the values of $K_4^{20,23}$ (i.e. we expand only the nodes that satisfy the consistency predicate in advance). This direct approach is more efficient than the basic approach since the size of the layers of the guess tree (including the size of the widest layer) is reduced in exchange for a small amount of precomputation and memory.
2. Given the case of two input-output pairs, we can use differential methods in order to simultaneously reduce the number of unknown bits and constraint bits in our equations. As a result, the size of our precomputed tables can be reduced. For example, for each input-output pair, we have a 4-bit equation, whose left hand side is $Z_R^{31,2}$ ($Z_R^{*31,2}$ for the second pair). The value of Z_R^{31} is unknown, and $Z_R^{0,2}$ is obtained by subtracting a known 3-bit value from 3 bits of K_3 ($K_3^{0,2}$). Altogether, when we consider both input-output pairs, we have 8 constraint bits and 5 unknown bits on the left side of the equation. By subtracting the two 4-bit equations, $K_3^{0,2}$ is eliminated, and we get a single 4-bit equation whose left hand side is $(Z_R \boxminus Z_R^*)^{31,2}$. This equation has only 1 unknown bit ($(Z_R \boxminus Z_R^*)^{31}$), and 3 known bits ($(Z_R \boxminus Z_R^*)^{0,2}$). Note that both the number of unknown bits and constraints in the equations is reduced by 4, and thus we do not remove possible solutions or add solutions which are not possible using the non-differential method.
3. To minimize the number of required carry and state bit guesses, we work on consecutive chunks of bits from right to left. For example, we initially derive key bits $K_4^{20,23}$. The corresponding data bits that are added to the key in round 4 are known from the two input-output pairs, and we only have to guess the two carry bits into bit 20 of the two addition operations (C_4^{20} and C_4^{*20}). This allows us to derive C_4^{24} and C_4^{*24}, which are required to derive $K_4^{24,27}$. Effectively, this approach enables us to guess the carries and state bits only initially, when working on the first chunk of bits. Afterwards, the carries and state bits are already known since we know all the relevant less significant bits needed to calculate them.

4. Initially, we guess values that are required to calculate the four LSBs of several addition operations. Since there is no carry into the LSBs, this enables us to reduce the number of required guesses of carry bits into addition operations of partial words. For example, we start by guessing $K_1^{0,3}$, which allows us to derive $Z_L^{11,14}$ without guessing any carry bits.

5.5 Details and Analysis of the 4-Round Attack

Consider the equations of Section 4.2 for the first pair, and similar equations for the second pair. From each one of these four 32-bit equations, we derive eight equations which equate 4-bit words, and are indexed by $i \in \{0, 1, ..., 7\}$:
(E_1^i): $Z_L^{4i+11,4i+14} = X_L^{4i+11,4i+14} \oplus f^i(X_R^{4i,4i+3} \boxplus K_1^{4i,4i+3} \boxplus C_1^{4i})$
(E_2^i): $Z_R^{4i+11,4i+14} = Y_R^{4i+11,4i+14} \oplus f^i(Y_L^{4i,4i+3} \boxplus K_4^{4i,4i+3} \boxplus C_4^{4i})$
(E_3^i): $Y_L^{4i+11,4i+14} \oplus Z_L^{4i+11,4i+14} = f^i(Z_R^{4i,4i+3} \boxplus K_3^{4i,4i+3} \boxplus C_3^{4i})$
(E_4^i): $X_R^{4i+11,4i+14} \oplus Z_R^{4i+11,4i+14} = f^i(Z_L^{4i,4i+3} \boxplus K_2^{4i,4i+3} \boxplus C_2^{4i})$
In addition to the carry words defined in Section 5.2, we define CS_2 and CS_3 as the 32-bit words $(Z_L \boxminus Z_L^*) \oplus Z_L \oplus Z_L^*$ and $(Z_R \boxminus Z_R^*) \oplus Z_R \oplus Z_R^*$ respectively.

In the rest of this section we describe the algorithm for deriving the 32 bits of K_1 and the 32 bits of K_4. Afterwards, deriving the values of K_2 and K_3 is immediate using the third and forth equations of Section 4.2 (Z_L and Z_R are known from the first and second equations).

Our tree contains 9 layers $(\ell_0, \ell_1, ..., \ell_8)$, where the procedure for expanding the nodes of layer $i \in \{0, 1, ..., 7\}$ uses equations $E_1^i, E_1^{i+2}, E_2^{i+5}, E_3^i$ and E_4^{i+5} (the index additions are performed numerically modulo 8). The steps of the procedure for expanding the nodes of each layer are basically the same and differ only according to the indices of the equations that are used (which determine the 4-bit chunks that we work on). Thus, we call the procedure for expanding layer $i \in \{0, 1, ..., 7\}$ an *iteration*.

Table 2 gives the iteration inputs and outputs calculated in each step of the iteration algorithm for $i \in \{0, 1, ..., 7\}$. Note that the carry and state bits and expressions which are outputs of the iteration i, serve as inputs to iteration $i+1$.

The steps of iteration $i \in \{0, 1, ..., 7\}$ are given below.[6] Note that steps 6–10 are analogous to steps 1–5, but are performed from the decryption side.

1. Given the inputs $K_1^{4i,4i+3}, C_1^{4i}, C_1^{*4i}$, use equation E_1^i to calculate $Z_L^{4i+11,4i+14}$ for both pairs.
2. Given $Z_L^{4i+11,4i+14}$ (from step (1)), use equation E_3^i to calculate $Z_R^{4i,4i+3} \boxplus K_3^{4i,4i+3} \boxplus C_3^{4i}$ for both pairs.
3. Subtract the expressions calculated in step (2), $Z_R^{4i,4i+3} \boxplus K_3^{4i,4i+3} \boxplus C_3^{4i}$ and $Z_R^{*4i,4i+3} \boxplus K_3^{4i,4i+3} \boxplus C_3^{*4i}$, to eliminate $K_3^{4i,4i+3}$, and obtain the value of $(Z_R^{4i,4i+3} \boxminus Z_R^{*4i,4i+3}) \boxplus (C_3^{4i} \boxminus C_3^{*4i} \boxminus CS_3^{4i})$.

[6] For the sake of simplicity, we do not mention the carry and state output bits in the description of the steps, and just list them in Table 2.

Step	Key input	Carry input	State input	Key output	Carry output	State output
(1)	$K_1^{4i,4i+3}$	C_1^{4i}, C_1^{*4i}	-	-	C_1^{4i+4}, C_1^{*4i+4}	-
(3)	-	-	-	-	$C_3^{4i+4} \boxminus$ $C_3^{*4i+4} \boxminus$ CS_3^{4i+4}	$(Z_R \boxminus Z_R^*)^{4i+3}$
(4)	-	$C_3^{4i} \boxminus$ $C_3^{*4i} \boxminus$ CS_3^{4i}	$(Z_R \boxminus Z_R^*)^{4i+31}$	-	-	-
(5)	-	C_4^{4i+20}, C_4^{*4i+20}	-	$K_4^{4i+20,4i+23}$	-	-
(6)	-	C_4^{4i+20}, C_4^{*4i+20}	-	-	C_4^{4i+24}, C_4^{*4i+24}	-
(8)	-	-	-	-	$C_2^{4i+24} \boxminus$ $C_2^{*4i+24} \boxminus$ CS_2^{4i+24}	$(Z_L \boxminus Z_L^*)^{4i+23}$
(9)	-	$C_2^{4i+20} \boxminus$ $C_2^{*4i+20} \boxminus$ CS_2^{4i+20}	$(Z_L \boxminus Z_L^*)^{4i+19}$	-	-	-
(10)	-	C_1^{4i+8}, C_1^{*4i+8}	-	$K_1^{4i+8,4i+11}$	C_1^{4i+12}, C_1^{*4i+12}	-

Steps (2) and (7) do not use any iteration input or calculate any iteration output.
Table 2. Iteration inputs used and iteration outputs calculated in each step of the iteration algorithm for $i \in \{0, 1, ..., 7\}$

4. Subtract the input $C_3^{4i} \boxminus C_3^{*4i} \boxminus CS_3^{4i}$ from the 3 LSBs of the expression calculated in step (3), and concatenate the 3-bit result with the input $(Z_R \boxminus Z_R^*)^{4i+31}$ to obtain $(Z_R \boxminus Z_R^*)^{4i+31,4i+2}$.
5. Given $(Z_R \boxminus Z_R^*)^{4i+31,4i+2}$ (from step (4)) and the carries $C_4^{4i+20}, C_4^{*4i+20}$, solve the equation obtained by subtracting right hand side of E_2^{i+5} to obtain $K_4^{4i+20,4i+23}$.
6. Given $C_4^{4i+20}, C_4^{*4i+20}$, and $K_4^{4i+20,4i+23}$ (derived in step (5)), use equation E_2^{i+5} to calculate $Z_R^{4i+31,4i+2}$ for both pairs.
7. Given $Z_R^{4i+31,4i+2}$ (from step (6)), use equation E_4^{i+5} to calculate $Z_L^{4i+20,4i+23} \boxplus K_2^{4i+20,4i+23} \boxplus C_2^{4i+20}$ for both pairs.
8. Subtract the expressions calculated in step (7), $Z_L^{4i+20,4i+23} \boxplus K_2^{4i+20,4i+23} \boxplus C_2^{4i+20}$ and $Z_L^{*4i+20,4i+23} \boxplus K_2^{4i+20,4i+23} \boxplus C_2^{*4i+20}$ to eliminate $K_2^{4i+20,4i+23}$, and obtain the value of $(Z_L^{4i+20,4i+23} \boxminus Z_L^{*4i+20,4i+23}) \boxplus (C_2^{4i+20} \boxminus C_2^{*4i+20} \boxminus CS_2^{4i+20})$.
9. Subtract the input $C_2^{4i+20} \boxminus C_2^{*4i+20} \boxminus CS_2^{4i+20}$ from the 3 LSBs of the expression calculated in step (8), and concatenate the 3-bit result with the input $(Z_L \boxminus Z_L^*)^{4i+19}$ to obtain $(Z_L \boxminus Z_L^*)^{4i+19,4i+22}$.

10. Given $(Z_L \boxminus Z_L^*)^{4i+19,4i+22}$ (from step (9)) and the inputs C_1^{4i+8}, C_1^{*4i+8}, solve the equation obtained by subtracting right hand side of E_1^{i+2} to obtain $K_1^{4i+8,4i+11}$.

All the steps of this iteration algorithm involve simple operations on 4-bit words (addition, subtraction, XOR and application of a 4×4 Sbox, or its inverse). The exceptional steps are (5) and (10), where we have to solve the equations obtained by subtracting the right hand sides of E_2^{i+5} and E_1^{i+2}, respectively. Each equation adds a 4-bit constraint on 4 unknown bits of the key, and thus we expect a single solution on average. However, it is possible that these equations will have more than one solution (and then we have to try each one), or no solutions at all (and then we can discard the guess at this stage). The solutions to each equation can be derived by using the basic approach of exhaustive search over the 2^4 possible values of the 4 key bits. However, we speed up the process for each equation by precomputing and storing the solutions for each of the 2^4 possible values of the equation and for each of the 2^{16} values of the 16 relevant input or output bits that participate in the equation. A table for a single equation has $2^{4+16} = 2^{20}$ entries, where each entry has an average of a single 4-bit solution (2^{22} bits, or 2^{16} words of 64 bits in total per table), and requires a negligible precomputation time compared to the complexity of the full attack on GOST.

The algorithm for deriving K_1 and K_4 expands the guess tree by running iterations $i \in \{0, 1, ..., 7\}$ in their natural order, guessing unknown iteration inputs when they are required. We now analyze its expected time complexity by calculating the width of the layers of the tree according to the expected number of guesses required at each stage of the algorithm: In general, iteration i requires the following input bits (as specified in Table 2): 4 bits of K_1 in step (1), 6 single carry bits in steps (1),(5),(6) and (8) (note that steps (5) and (6) require the same carry bits), 4 carry expression bits in steps (4) and (9) (note that the value of each carry expression is either -2,-1,0 or 1) and 2 state bit expressions in steps (4) and (9). Altogether, iteration i requires $4 + 6 + 4 + 2 = 16$ input bits. However, in iteration 0 (which is the first iteration performed), the carry inputs required in step (1) and the carry expression required in step (4) are known to be zero. Thus, iteration 0 requires only 12 unknown iteration input bits which we have to guess, thus the expected size of the second layer is 2^{12}. Note that the inverse Sbox computed in steps (2) and (7) is expected to provide a single output value per input (i.e. step (2) and (7) are not expected to increase the width of the guess tree). In addition, the equations solved in steps (5) and (10), are expected to have a single solution, as explained above.

In iteration 1 (where we derive layer 2 of the tree), iteration inputs which are carry and state bits are already known from the output of iteration 0. Moreover, after step (10) of the first iteration, we know the values of C_1^8 and C_1^{*8}. This gives us a 2-bit filtering condition on $K_1^{4,7}$ (we only try values of $K_1^{4,7}$ which are consistent with the carries). In this sense, the carries guessed in step (10) of the first iteration are "consumed" by the second iteration. Thus, after the first two iterations, we obtain $K_4^{20,27}$ and $K_1^{8,15}$ from guessing 8 bits of the first key, $K_1^{0,7}$. In addition, we have an expected number of $2^{8-2} = 2^6$ additional guesses (counting the carry and state bit guesses of steps (2)–(10) of iteration 0, without the 2-bit guess of step (10)). Thus, the expected size of layer 2 is $2^{8+6} = 2^{14}$, which is larger than the 2^{12} expected size of layer 1, but not by a large factor.

In iteration 2, we derive $K_4^{28,31}$ and $K_1^{16,19}$ from $K_1^{8,11}$. Since $K_1^{8,11}$ is already known at this stage, we do not need to guess it again. Thus, the size of layer 3 remains the same as in layer 2, namely 2^{14} possible solutions. This pattern continues until the end of iteration 5, where our partial guess nodes include the values of $K_1^{0,31}$ and $K_4^{20,11}$ (as shown in Table 3). In iterations 6 and 7, we derive the remaining bits of K_4 ($K_4^{12,19}$) and the bits of K_1 ($K_1^{0,7}$) which were already guessed, and give us additional 4-bit filtering conditions on the guesses in each of these iterations. Thus, layer 7 of the tree in expected to contain $2^{14-4} = 2^{10}$ nodes. Iteration 7 is the final iteration, in which besides the 4-bit filtering condition on $K_1^{4,8}$, we also obtain the remaining 6 iteration inputs guessed in iteration 0. We thus receive additional filtering conditions of 6 bits and expect the final layer to contain $2^{10-4-6} = 1$ node (a single value for K_1 and K_4, as expected when we compare the total number of key and input-output constraint bits).

Iteration	0	1	2	3	4	5	6	7
K_1 bits derived	0–3 8–11	4-7 12–15	$\underline{8\text{–}11}$ 16–19	$\underline{12\text{–}15}$ 20–23	$\underline{16\text{–}19}$ 24–27	$\underline{20\text{–}23}$ 28–31	$\underline{24\text{–}27}$ $\underline{0\text{–}3}$	$\underline{28\text{–}31}$ $\underline{4\text{–}7}$
K_4 bits derived	20–23	24–27	28–31	0–3	4–7	8–11	12–15	16–19

The key bits which are already known from previous iterations are underlined.

Table 3. The key bits derived in each iteration

The expected number of nodes in the widest layer of the partial guess tree is 2^{14}, and it is obtained at iterations 1 to 5 (which define layers 2 to 6 in the tree). Thus, the time complexity of the algorithm is about 2^{14} Feistel structure evaluations for each one of the two input-output pairs, and 2^{15} evaluations altogether. Since we work on a 4-round Feistel structure which contains a fraction of 2^{-3} of the 32 rounds of the full GOST, we estimate that the expected time complexity of this attack is equivalent to about $2^{15-3} = 2^{12}$ GOST evaluations. We apply this 4-round attack for each one of the 2^{128} possible values of the last 4 round keys (K_5, K_6, K_7 and K_8), and thus the time complexity of the 8-round attack is about $2^{128+12} = 2^{140}$ GOST evaluations.

In terms of memory, we store precomputed tables for steps (5) and (10) in each iteration. The equations solved in these two steps are of the same structure for each one of the 8 iterations and differ only according to the Sbox used. Thus, we need 8 such tables (one for each Sbox), which require $8 \cdot 2^{22} = 2^{25}$ bits of memory. The additional memory required to store other intermediate variables and to store our state in the DFS traversal is negligible compared to the space consumed by the precomputed tables. Hence, the attack has a completely practical memory complexity of 2^{25} bits, which is equivalent to 2^{19} 64-bit words.

6 A New 2-Dimensional Meet-in-the-middle Attack on 8 Rounds of GOST

In this section, we present a new attack on 8 rounds of GOST given two input-output pairs, which combines the ideas of the "Guess and Determine" attack and the MITM attacks. Unlike the attack of the previous section, we do not guess the last 4 round keys in advance. Instead, we divide the 8-round Feistel structure horizontally by splitting it into a *top part*, which uses round keys K_1–K_4, and a *bottom part*, which uses round keys K_5–K_8.

The main additional property of 4-round GOST that we exploit in this attack is the slow diffusion of the input bits into the state. This allows us to divide the 4-round attack of Section 5 on the top part into two partial 4-round attacks. Each partial 4-round attack uses part of the input and a set containing only 82 state bits (out of the 128 bits of Y and Y^*) in order to obtain suggestions for the values of a set containing slightly more than half of the key bits of K_1–K_4, and some state material. The slow diffusion of the input bits into the state ensure that the input bits and state bits (used in each partial 4-round attack) contain almost all the information required to uniquely recover the corresponding key bits, and thus each partial 4-round attack does not give many suggestions for the partial key. We note that the full 4-round attack uses all of the 128 bits of Y and Y^* to recover all of the 128 bits of K_1–K_4. This implies that the union of the two sets of state bits used in the two partial 4-round attacks contains the 128 bits of Y and Y^*, and the union of the two sets of recovered key bits contains the 128 bits of K_1–K_4.

Since the value of Y and Y^* is unknown, we execute each partial 4-round attack 2^{82} times (once for each value of the 82 state bits it requires). We can now execute a MITM attack in which we compare the values of the common key and state material derived from the two partial 4-round attacks, and obtain suggestions for K_1–K_4 with corresponding 128-bit values of Y and Y^*. We then use the same type of attack on the bottom part of the 8-round Feistel structure, obtain suggestions for K_5–K_8 and use a final MITM attack which joins the two parts by comparing the 128-bit values of Y and Y^* to obtain suggestions for the full key.

Schematically, we split the top and bottom parts of the block cipher vertically into two (potentially overlapping) cells, such that on each cell we execute an independent partial attack to obtain suggestions for a part of the key. We then join all the suggestions to obtain suggestions for the full key using three MITM attacks. This can be visualized using a 2×2 matrix (as shown in Figure 5), where the horizontal line separates the four first and last rounds of the 8-round block cipher, and the dashed vertical line separates the left and right cells in each one of the top and bottom parts.

After the MITM attacks on the top and bottom parts of the Feistel structure, we obtain 2^{128} suggestions for K_1–K_4 and 2^{128} suggestions for K_5–K_8, each with corresponding 128-bit values of Y and Y^*. Note that so far we did not filter out any possible keys, and thus the final MITM attack, which compares the 128-bit values of Y and Y^* to obtain about 2^{128} suggestions for the full key, is

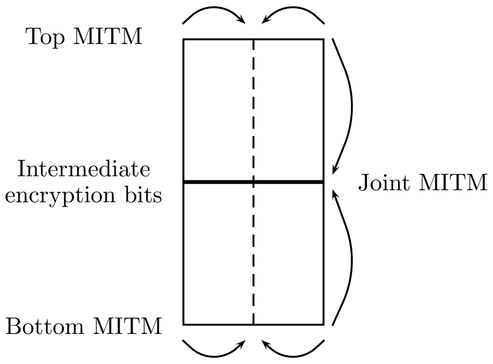

Fig. 5. The general framework of the 2-dimensional meet-in-the-middle attack

essentially the basic MITM attack of Section 4.1, which would normally require 2^{128} memory.

To reduce the memory consumption, we guess many of the 128 bits of Y and Y^* in advance (in the outer loop of the 8-round attack). For each possible value of those bits, we execute the 2DMITM (2-dimensional MITM) attack described above, but obtain fewer suggestions for the key which we have to store. This increases the number of times that we execute the partial 4-round attacks and potentially the overall time complexity of the full 8-round attack. However, this is not the case, as the partial 4-round attacks were relatively efficient (the time complexity of each one was at most 2^{18}) and were originally executed only 2^{82} times. Thus, the partial 4-round attacks were not the bottleneck of the time complexity of the attack.[7]

6.1 Details of the 8-Round Attack

Formally, we define the following sets which contain bits of Y and Y^*: S_1 is the set of bits that we guess in the outer loop of the 8-round attack. S_2 and S_3 are the sets of remaining bits, which are not guessed in advance (i.e. not in S_1), that are required for the execution of the partial 4-round attacks on the left and right cells, respectively, of the top part of the 8-round Feistel structure. Namely, $S_1 \bigcap S_2 = \emptyset$, and $S_1 \bigcup S_2$ is the minimal set that contains all the bits of Y and Y^* which are required by the partial 4-round attack on the left cell of the top part. Similarly, $S_1 \bigcap S_3 = \emptyset$, and $S_1 \bigcup S_3$ is the minimal set that contains all the bits of Y and Y^* which are required by the partial 4-round attack on the right cell of the top part. For the bottom MITM attack, we define S_4 and

[7] Note again that we expect about 2^{128} keys to fulfill the filtering conditions of the two input-output pairs. Thus, the time required for the attack cannot be reduced below 2^{128} (without exploiting additional filtering conditions).

S_5 in a similar way to S_2 and S_3, respectively, but for the bottom part of the 8-round Feistel structure. Note that since the 4-round attacks on both the top and bottom parts require all the 128 intermediate bits, $S_2 \bigcup S_3 = S_4 \bigcup S_5$.

The details of the 4-round attacks are given in the next section. We now refer to them as black boxes, and give the algorithm of the full 8-round attack:

1. For each value of the bits of the set S_1:
   (a) Perform the 4-round attack on the top part of the Feistel structure, and obtain a list with values of K_1–K_4, sorted according to the value of the bits of $S_2 \bigcup S_3$.
   (b) Perform the 4-round attack on the bottom part of the Feistel structure. For each value of $S_4 \bigcup S_5 = S_2 \bigcup S_3$ (given along with the value of K_5–K_8), search the list obtained in the previous step of matches. For each match test the full key K_1–K_8 with the given plaintext-ciphertext pairs.

6.2 Details of the 4-Round Attacks

We concentrate first on the top part of the 8-round Feistel structure: each one of the two partial 4-round attacks on the top part sequentially executes a subset of the iterations defined in Section 5, and is called an iteration *batch*. The first (left) iteration batch executes iterations 0–3, and the second (right) executes iterations 4–7.

After performing iteration batches 0–3 and 4–7 independently, we get suggestions for the values of some key bits, and for some carry and state bits. We then discard inconsistent suggestions by comparing the values of the common bits that are derived by both of the iteration batches. We partition these bits into three groups:

1. G_1 contains the 16 key bits which are derived by both of the iteration batches 0–3 and 4–7 (as specified in Table 4).
2. G_2 contains the carry and state iteration input bits that we guess in iteration 0, not including step (10) (the bits that we guess in step (10) are already used as filtering conditions in iteration 1). Using Table 2, we get $|G_2| = 6$ (using the fact that the carry bits are known to be zero). Note that the bits of G_2 are also contained in the set of iteration output bits of iteration 7 (of batch 4–7), and can thus be used to discard inconsistent suggestions made by batches 0–3 and 4–7.
3. G_3 contains the carry and state iteration input bits that we guess in iteration 4 (the first iteration of batch 4–7), not including the bits that we guess in step (10). Using Table 2, we get that $|G_3| = 10$ (unlike iteration 0, in iteration 4 no carry bits and expressions are known in advance). Note that the bits of G_3 are also contained in the set of iteration output bits of iteration 3 (of batch 0–3), and can thus be used to discard inconsistent suggestions made by batches 0–3 and 4–7.

Assume that the values of all the bits of S_1 are known. We now give the algorithm of the MITM attack performed on the top part of the 8-round Feistel structure:

Iteration	0	1	2	3	4	5	6	7
K_1 bits derived	(0–3) 8–11	(4–7) 12–15	$\underline{8\text{–}11}$ (16–19)	$\underline{12\text{–}15}$ (20–23)	(16–19) 24–27	(20–23) 28–31	$\underline{24\text{–}27}$ (0–3)	$\underline{28\text{–}31}$ (4–7)
K_4 bits derived	20–23	24–27	28–31	0–3	4–7	8–11	12–15	16–19
Bits of Y and Y^* required	$R[31,2]$ $L[11,14]$ $L[20,23]$	$R[3,6]$ $L[15,18]$ $L[24,27]$	$R[7,10]$ $L[19,22]$ $L[28,31]$	$R[11,14]$ $L[23,26]$ $L[0,3]$	$R[15,18]$ $L[27,30]$ $L[4,7]$	$R[19,22]$ $L[31,2]$ $L[8,11]$	$R[23,26]$ $L[3,6]$ $L[12,15]$	$R[27,30]$ $L[7,10]$ $L[15,19]$

Key bits which are known from previous iterations of the batch are underlined. Key bits of G_1 (derived by both of the iteration batches) appear is parenthesis. The bits of Y and Y^* are denoted as follows: $R[i,j]$ denotes $Y_R^{i,j}$ and $Y_R^{*i,j}$, $L[i,j]$ denotes $Y_L^{i,j}$ and $Y_L^{*i,j}$.

Table 4. The key bits derived and the intermediate encryption bits required in each iteration of iteration batches 0–3 and 4–7

1. For each value of the bits of S_2, perform the batch of iterations 0–3. Save all the nodes of the final layer in a list. These nodes contain the values of slightly more than half of the bits of K_1–K_4 (including the values of the bits of G_1), and also the values of the bits of G_3. In addition to the information obtained by each node, also save the value of the initial guess of the bits of G_2, and the value of the bits of S_2 per node. Sort the list according to the values of G_1,G_2 and G_3.
2. For each value of the bits of S_3, perform the batch of iterations 4–7. For each node in the final layer obtain the value of the bits of G_1,G_2 and G_3 and search the list obtained in the first step for their value. For each match, save the value of the full K_1–K_4 in a sorted list according to the value of the bits of $S_2 \bigcup S_3$.

The iteration batches of the MITM attack on the bottom part of the Feistel structure are performed from the decryption side and are completely analogous to the iteration batches on the top part (i.e. in iteration 0, we start by guessing $K_8^{0,3}$, and derive $K_5^{20,23}$ and $K_8^{8,11}$). We also define analogous sets to G_1,G_2 and G_3 for the bottom part.

Before analyzing the complexities of the top and bottom MITM attacks, we determine the sets S_1–S_5. We refer to Table 4, which gives the indices of the intermediate encryption bits required by iterations 0–7 of the top part of the 8-round Feistel structure. In order to calculate the indices of these bits, recall from Section 5.5 that iteration $i \in \{0,1,...,7\}$ uses equations $E_1^i, E_1^{i+2}, E_2^{i+5}, E_3^i$ and E_4^{i+5}, out of which only E_2^{i+5} and E_3^i require bits of Y and Y^*: E_2^{i+5} requires $Y_R^{4i+31,4i+2}$ and $Y_L^{4i+20,4i+23}$, and E_3^i requires $Y_L^{4i+11,4i+14}$ (note that iteration i also requires the same indices for Y^*). Altogether, iterations 0–3 require the 82 intermediate bits $Y_R^{31,14}$, $Y_L^{11,3}$, $Y_R^{*31,14}$ and $Y_L^{*11,3}$, and iterations 4–7 require the 82 intermediate bits of $Y_R^{15,30}$, $Y_L^{27,19}$, $Y_R^{*15,30}$ and $Y_L^{*27,19}$. After calculating the indices of the intermediate encryption bits that the iteration batches of the top part require, we can easily derive the analogous indices that the iteration batches of the bottom part require, taking into account that the right and left 32-bit halves of Y and Y^* are exchanged at the end of round 4. Thus, we need to exchange the right and left halves of the bits calculated for the top part: for the bottom part, iteration batch 0–3 requires the 82 intermediate encryption bit values of $Y_L^{31,14}$,$Y_R^{11,3}$,$Y_L^{*31,14}$ and $Y_R^{*11,3}$ and the iteration batch 4–7 requires the 82 bits of $Y_L^{15,30}$,$Y_R^{27,19}$,$Y_L^{*15,30}$ and $Y_R^{*27,19}$.

The sets S_1–S_5 that we choose are given in table 5. Note that since the right and left 32-bit halves of Y and Y^* are exchanged at the end of round 4, we choose S_1 so that it contains the same bit indices from both halves of Y and Y^*. As a result, the sets used during the iteration batches are of the same size ($|S_2| = |S_3| = |S_4| = |S_5| = 18$). This implies that the iteration batches of both the top and the bottom parts are performed the same number of times (2^{18}) for a given value of the 92 bits of S_1.

S_1	$Y_L^{10,19}$, $Y_L^{23,3}$, $Y_R^{10,19}$, $Y_R^{23,3}$, $Y_L^{*10,19}$, $Y_L^{*23,3}$, $Y_R^{*10,19}$, $Y_R^{*23,3}$
S_2	$Y_L^{20,22}$, $Y_R^{4,9}$, $Y_L^{*20,22}$, $Y_R^{*4,9}$
S_3	$Y_L^{4,9}$, $Y_R^{20,22}$, $Y_L^{*4,9}$, $Y_R^{*20,22}$
S_4	$Y_R^{20,22}$, $Y_L^{4,9}$, $Y_R^{*20,22}$, $Y_L^{*4,9}$
S_5	$Y_R^{4,9}$, $Y_L^{20,22}$, $Y_R^{*4,9}$, $Y_L^{*20,22}$

Table 5. The sets S_1–S_5

We now analyze the complexity of the MITM attack on the top part of the Feistel structure: as calculated in Section 5.5, when starting the iteration batch from iteration 0, the expected maximal size of the tree is 2^{14}. It is obtained after iteration 1, and is maintained until the end of iteration 5 (even though we do not perform 5 consecutive iterations in this attack). The time complexity of the first step of the attack is thus about $2^{|S_2|+14} = 2^{14+18} = 2^{32}$, and this is also the size of the sorted list at the end of the first step. The maximal size of the tree of the iteration batch 4–7 is $2^{14+4} = 2^{18}$ (as described above, we have to guess 4 more carry bits compared to iterations 0–3). Thus, the time complexity of expanding the tree in the second step is $2^{|S_3|+18} = 2^{36}$. The time and memory complexities of the remainder of step 2 (in which we match the iteration batches) are $2^{|S_2|+|S_3|+14+18-(|G_1|+|G_2|+|G_3|)} = 2^{|S_2|+|S_3|+14+18-(16+6+10)} = 2^{|S_2|+|S_3|} = 2^{36}$. Note that it is not surprising that the time and memory complexities of the matching part of the attack reduce to $2^{|S_2|+|S_3|}$, since given the full 128-bit intermediate value, we expect that only one key survives the filtering conditions. Altogether, the memory complexity of the top MITM attack is about 2^{36} 64-bit words. The time complexity is dominated by step 2 and is equivalent to about 2^{36} 4-round Feistel structure evaluations, which is equivalent to about 2^{33} evaluations of the full GOST cryptosystem. For the bottom MITM attack, we obtain the same time and memory complexities, since the sizes of S_4 and S_5 are equal to the sizes of S_2 and S_3, and the sets corresponding to G_1, G_2 and G_3 are completely symmetrical.

6.3 Analysis of the 8-Round Attack on GOST

We analyze the attack of Section 6.1: The time complexities of each of the MITM attacks on the bottom and top parts in steps (a) and (b) are equivalent to about 2^{36} 4-round Feistel structure evaluations, as calculated above. The number of expected matches for which we run the full cipher in step (b) is $2^{36+36-36} = 2^{36}$. Hence, the time complexity of these steps is equivalent to a bit more than 2^{36} full GOST evaluations. Since $|S_1| = 92$, the total time complexity of the attack is equivalent to about $2^{92+36} = 2^{128}$ GOST evaluations. The total memory complexity of the attack is about 2^{36} 64-bit words, and is dominated by the sorted list calculated in step (a).

7 Conclusions and Open Problem

In this paper we introduced several new techniques such as the fixed point property and two dimensional meet in the middle attacks, and used them to greatly improve the best known attacks on the full 32-round GOST. In particular, we reduced the memory complexity of the attacks from an impractical 2^{64} to a practical 2^{36} (and to an even more practical 2^{19} complexity, which can fit into the cache of modern microprocessors, with a small penalty in the running time). The lowest time complexity of our attacks is 2^{192}, which is 2^{32} times better than previously published attacks but still very far from being practical. Consequently, we are concerned about the weaknesses which were demonstrated in the design of GOST (especially in its simplistic key schedule), but do not advocate that its current users should stop using it right away.

The main open problems left in this paper are whether it is possible to find faster attacks, and how to better exploit other amounts of available data (in addition to the 2^{32} and 2^{64} complexities considered in this paper, which are the natural thresholds for our techniques).

References

1. Eli Biham, Orr Dunkelman, and Nathan Keller. Improved Slide Attacks. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 153–166. Springer, 2007.
2. David Chaum and Jan-Hendfik Evertse. Cryptanalysis of DES with a Reduced Number Of Rounds: Sequences of Linear Factors in Block Ciphers. In *Advances in Cryptology, CRYPTO 85*, pages 192–211. Springer-Verlag, 1986.
3. Nicolas T. Courtois. Security Evaluation of GOST 28147-89 in View of International Standardisation. Cryptology ePrint Archive, Report 2011/211, 2011. http://eprint.iacr.org/.

4. Nicolas T. Courtois and Michał Misztal. Differential Cryptanalysis of GOST. Cryptology ePrint Archive, Report 2011/312, 2011. http://eprint.iacr.org/.
5. Ewan Fleischmann, Michael Gorski, Jan-Hendrik Huehne, and Stefan Lucks. Key Recovery Attack on full GOST Block Cipher with Negligible Time and Memory. Presented at Western European Workshop on Research in Cryptology (WEWoRC), 2009.
6. Takanori Isobe. A Single-Key Attack on the Full GOST Block Cipher. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.
7. Orhun Kara. Reflection Cryptanalysis of Some Ciphers. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2008.
8. John Kelsey, Bruce Schneier, and David Wagner. Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 1996.
9. Youngdai Ko, Seokhie Hong, Wonil Lee, Sangjin Lee, and Ju-Sung Kang. Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 299–316. Springer, 2004.
10. National Bureau of Standards. Federal Information Processing Standard-Cryptographic Protection - Cryptographic Algorithm. GOST 28147-89, 1989.
11. OpenSSL. A Reference Implementation of GOST. http://www.openssl.org/source/.
12. Vladimir Rudskoy. On Zero Practical Significance of Key Recovery Attack on Full GOST Block Cipher with Zero Time and Memory. Cryptology ePrint Archive, Report 2010/111, 2010. http://eprint.iacr.org/.
13. Haruki Seki and Toshinobu Kaneko. Differential Cryptanalysis of Reduced Rounds of GOST. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2000.

A Appendix: Exploiting GOST's Complementation Property

The full GOST block cipher has a well-known complementation property. If the plaintext $P = (P_L, P_R)$ is encrypted under $K = (K_1, K_2, \ldots, K_8)$ to the ciphertext $C = (C_1, C_2)$, then the encryption of $P^* = (P_L \oplus e_{31}, P_R \oplus e_{31})$ under $K = (K_1 \oplus e_{31}, K_2 \oplus e_{31}, \ldots, K_8 \oplus e_{31})$ is $C^* = (C_1 \oplus e_{31}, C_2 \oplus e_{31})$ (where e_{31} is the 32-bit vector whose entries are all zero, except the MSB, which is one.).

At the same time, in our attacks on reduced-round GOST, we notice the existence of two less known complementation properties: for $G_{K_1,K_2,K_3,K_4}(P_L, P_R) = (T_L, T_R)$, $G_{K_1 \oplus e_{31},K_2,K_3 \oplus e_{31},K_4}(P_L, P_R \oplus e_{31}) = (T_L, T_R \oplus e_{31})$ and $G_{K_1,K_2 \oplus e_{31},K_3,K_4 \oplus e_{31}}(P_L \oplus e_{31}, P_R) = (T_L \oplus e_{31}, T_R)$.

One can use these three complementation properties in all of our attacks (even though each one of them leads to a different improvement factor). For example, consider the meet-in-the-middle attack suggested in Section 4.2. In this attack, we obtain two 8-round input-output pairs (I, O) and (I^*, O^*). The attack starts by guessing Y (the partial encryption of I after four rounds). The naive way to implement the search loop is to try any possible value of Y, and then any value of K_3, K_4 to obtain the candidate values of K_1, K_2. However, for each guess of Y, I, K_3, K_4, consider the 2^{64} candidates for K_1, K_2. If we consider the list of candidates for $Y \oplus (e_{31}, e_{31}), I \oplus (e_{31}, e_{31}), K_3 \oplus e_{31}, K_4 \oplus e_{31}$, it is the same as the previous one (up to the MSBs of K_1 and K_2). The same is true for the other two complementation properties.

In other words, instead of computing the three additional lists (for each of the three complementation properties) we can perform this step only once. As there are four 4-round steps (we need to deal with (I, Y), (Y, O), (I^*, Y^*) and (Y^*, O^*)), we can save three out of the 16 4-round steps (i.e., for each I, $I \oplus (0, e_{31})$, $I \oplus (e_{31}, 0)$ and $I \oplus (e_{31}, e_{31})$ with all the corresponding Y's we compute the list only once).

We note that in the attacks based on the fix point point property, the first input-output pair is actually (I, I), hence, one can use the complementation property again (once for $(I, I \oplus (e_{31}, e_{31}))$ and once for $(I \oplus (0, e_{31}), I \oplus (e_{31}, 0))$. Additionally, as O^* is I (up to a swap), one can again save two out of the four rounds computations. In total, this improvement results in an overall saving of 7/16 in the 8-round attack.

In the unoptimized fixed-point attack there are 2^{192} steps of full-GOST trial encryptions, and 2^{192} executions of the 8-round attack, which result in a total time complexity equivalent to $(32+16) \cdot 2^{192} = 48 \cdot 2^{192}$ rounds of GOST. Using this improvement, the total running time is reduced to $(32+9) \cdot 2^{192} = 41 \cdot 2^{192}$ rounds of GOST, a speed up of about 14.6% in the total running time.

In the reflection-based attacks one can optimize the trial encryptions: instead of performing 2^{224} full-GOST trial encryptions, it is possible to exploit the additional "half pair" and obtain an additional 32-bit filtering condition by running 8 rounds of GOST. As a result, the trial encryptions require less than 2^{224} full-GOST evaluations, while the 8-round attacks take more than that. Thus, unlike the fixed-point-based attacks, in the reflection-based attacks the 8-round attacks form the bottleneck, and reducing their complexity gives a more significant savings. We note that the complex attack procedure of Section 6 can also be improved by changing the order of the loop. To do so, one needs to reorder the guess of X, and Y accordingly. Therefore, using a chosen plaintext model for the reflection-based attacks (to obtain 2^{32} appropriate plaintext-ciphertext pairs), it is possible to perform the analysis for three out of the four 4-round phases only once. This reduces the running time to 7/16 of the original time complexity. The total running time of the improved attack is thus reduced to $2^{222.8}$ applications of the 8-round attack.