BLACK-BOX COLLISION ATTACKS ON THE COMPRESSION FUNCTION OF THE GOST HASH FUNCTION

Nicolas T. Courtois and Theodosis Mourouzis
Computer Science, University College London, Gower Street, WC1E 6BT, London, U.K.

Keywords: Hash functions, Block ciphers, GOST, Compression function, Cryptanalysis, Generic attacks, Collisions.

Abstract: The GOST hash function and more precisely GOST 34.11-94 is a cryptographic hash function and the official government standard of the Russian Federation. It is a key component in the national Russian digital signature standard. The GOST hash function is a 256-bit iterated hash function with an additional checksum computed over all input message blocks. Inside the GOST compression function, we find the standard GOST block cipher, which is an instantiation of the official Russian national encryption standard GOST 28147-89.
In this paper we focus mostly on the problem of finding collisions on the GOST compression function. At Crypto 2008 a collision attack on the GOST compression function requiring 2^{96} evaluations of this function was found. In this paper, we present a new collision attack on the GOST compression function which is fundamentally different and more general than the attack published at Crypto 2008. Our new attack is a black-box attack which does not need any particular weakness to exist in the GOST block cipher, and works also if we replace GOST by another cipher with the same block and key size. Our attack is also slightly faster and we also show that the complexity of the previous attack can be slightly improved as well.
Since GOST has an additional checksum computed over all blocks, it is not obvious how a collision attack on the GOST compression function can be extended to a collision attack on the hash function. In 2008 Gauravaram and Kelsey develop a technique to achieve this, in the case in which the checksum is linear or additive, using the Camion-Patarin-Wagner generalized birthday algorithm. Thus at Crypto 2008 the authors were also able to break the collision resistance of the complete GOST Hash function.
Our attack is more generic and shows that the GOST compression function can be broken whatever is the underlying block cipher, but remains an attack on the compression function. It remains an open problem how and if this new attack can be extended to a collision attack on the full GOST hash function.

1 INTRODUCTION

The GOST 34.11-94 hash function is defined in the national Russian hash standard (GOST, 1994) and a key component on the national Russian digital signature standard. It is a 256-bit hash function which has an iterative structure like most of the common hash functions such as MD5 and SHA-1 with a particularity that it involves an extra checksum computed over all input message blocks. The value of this checksum is an input to the final compression function.

The GOST hash function is based on the GOST 28147-89 block cipher. The specification of the GOST block cipher can be found in (I.A. Zabotin, 1989). GOST 28147-89 is a well-known block cipher which started as an official government standard of the former Soviet Union, and remains the official encryption standard of the Russian Federation. GOST has very large 256-bit keys and is intended to provide a military level of security. Very few serious standardized block ciphers were ever broken, and until 2010 the consensus in the cryptographic community was that GOST could or should be very secure, which was summarized in 2010 in these words: "despite considerable cryptanalytic efforts spent in the past 20 years, GOST is still not broken", see (A. Poschmann, 2010). In addition GOST is an exceptionally competitive block cipher in terms of the cost of implementation, see (A. Poschmann, 2010). Accordingly, in 2010 GOST was submitted to ISO, to become a worldwide industrial encryption standard.

Then suddenly the GOST cipher was shown to be broken by several new attacks, see (Isobe, 2011; Courtois, b; Courtois, a). It is however not necessary at all to be able to break the block cipher, in order to find attacks on the hash function. Here the attacker has much more freedom to control and manipulate the key. He can exploit weak keys, related keys,

T. Courtois N. and Mourouzis T..
BLACK-BOX COLLISION ATTACKS ON THE COMPRESSION FUNCTION OF THE GOST HASH FUNCTION.
DOI: 10.5220/0003525103250332
In *Proceedings of the International Conference on Security and Cryptography* (SECRYPT-2011), pages 325-332
ISBN: 978-989-8425-71-3

and other ideas to benefit from the additional degrees of freedom available. There is a substantial amount of published work which deal with these aspects of GOST, see (Kara, 2008; Isobe, 2011; Courtois, a) for a more complete bibliography. In this paper we will look only at one aspect of GOST which is the study of symmetric fixed points in GOST (Kara, 2008; Isobe, 2011; Courtois, a; F. Mendel N. Pramstaller, 2008), which are those which are relevant in the attack on the GOST hash function (F. Mendel N. Pramstaller, 2008) presented at Crypto 2008. In our attack we will not use any particular property of GOST, only a higher-level self-similarity property: one inside the compression function, namely the fact that it uses the same block cipher several times. This is a very weak property present in many cryptographic constructions.

1.1 Hash Functions

A hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ is a map that maps a message M of arbitrary but finite length to a fixed-length hash value. Cryptographic hash functions are never injective and have many applications in information security such as digital signatures, message authentication codes (MACs) and other forms of authentication (Schneier, 1996). The three main security requirements for cryptographic hash functions are one-wayness, second pre-image resistance, and collision resistance (Schneier, 1996; J. Talbot, 2006). The properties which seem to be the hardest to achieve for the designers of hash functions, or those on which most successful attacks concentrate are second pre-image resistance, and collision resistance. Each property has its own generic attacks and their complexity determines the security objective to be attained, as shown in Table 1. Any attack below this bound will be considered as a valid shortcut attack on this security property.

Table 1: Number of messages needed to perform an attack.

Resistance type	Number of messages
Collision	$2^{n/2}$
Pre-image	2^n
Second pre-image	2^n

We can remark that the reference attack complexity level for collision attacks on hash functions is much smaller than in the other two security notions, due to the well-known birthday paradox (Schneier, 1996). This security requirement depends only on the size of the output space. In the GOST hash function, the output space for the compression function which hashes messages of a fixed length, and for the full hash function which hashes messages of variable length, are of the same size, both outputs are 256 bits long. Accordingly in both cases the goal of the attacker is to find an attack faster than 2^{128} times the cost of computing the compression function. We stress the fact that for the hash functions the reference unit is also the cost of computing the compression function, and the cost of computing the hash function is variable. These bounds correspond closely to what really is achieved by generic attacks on hash functions. In both cases, any attack to compute collisions faster than 2^{128} computations of the GOST compression function, will be considered as a valid shortcut attack which allows to break the given (hash or compression) function.

Attacks on hash functions can occur at three distinct levels. Some of the attacks on hash function are generic and high-level attacks related to high-level construction used in a specific hash function, such as the Merkle-Damgård construction (Damgård, 1990). Other are more specific attacks exhibit a specific weaknesses of a given compression function. Finally in block-cipher based constructions, one can also go one level deeper and exploit particular weaknesses of the underlying block cipher.

It is well known that if the compression function is collision-resistant, so is the resulting Merkle-Damgård construction (Damgård, 1990). If the compression function is not collision-resistant, as it is the case in this paper, it may be possible or not to extend the attack to the full hash function, but the exact way to do that will depend a lot on the high-level construction.

1.2 Specificity of GOST Hash

At Crypto 2008 an attack on the GOST compression function of complexity of 2^{96} evaluations of the compression function is presented. Then the attack is extended to a collision attack on the full GOST hash function with a complexity of 2^{105} evaluations of the compression function, see (F. Mendel N. Pramstaller, 2008). This extension is a non-trivial step. GOST contains a major innovation compared to many classical hash functions based on the Merkle-Damgård construction (Damgård, 1990). It has an additional checksum computed over all input message blocks which is hashed in the last application of the compression function. However in 2008 Gauravaram and Kelsey demonstrated that if the checksum is linear or additive, one can still do the extension of the attack and this independently of the underlying compression function (P. Gauravaram, 2008). The extension method uses the Camion-Patarin-Wagner generalized birthday attack (P. Camion, 1991; Wagner, 2002). Thus at Crypto 2008 the authors were able also to

break the collision resistance of the complete GOST Hash function. Hence, this sort of extra checksum does not prevent GOST and many other hash functions from being cryptanalysed.

1.3 Our Focus

The attack on the GOST hash function presented at Crypto 2008 is an extension of an attack on the underlying compression function of complexity of 2^{96} evaluations of the compression function (F. Mendel N. Pramstaller, 2008). This attack is based on a specific weakness of the GOST block cipher, where given some plaintexts with a specific structure, the attacker can construct a fixed point for the block cipher by constructing a fixed point in the first eight rounds (Kara, 2008).

In this paper, we present a generic black-box attack on the GOST compression function with improved complexity which also works if we replace the GOST by any other cipher with the same block and key size. Our attack is very different from the attack presented by Mendel *et al*(2008) (F. Mendel N. Pramstaller, 2008). Currently we are not able to propose an extension of this new attack for the full GOST hash function.

Our collision attack on the compression function works by exploiting the linear nature of key derivation and other components of the compression function and by constructing a certain subspace where we are going to be able to exploit the self-similarity of the GOST compression method, and force two GOST encryptions to coincide, without exploiting any weaknesses in the internal structure of the GOST block cipher. Our attack shows that attacks of the same complexity can be performed for any hash function built in the same way, regardless whether the GOST cipher is or not a secure cipher in some sense.

Table 2: Comparison of results regarding collision attacks for the compression function of the GOST hash function.

Source	Attack complexity
Mendel *et al*(2008)	2^{96}
Our Improvement	$2^{95.58}$
This work	$2^{95.58}$

2 GOST HASH FUNCTION

The GOST hash function, defined in the Russian government standard GOST R 34.11-94 and it is a cryptographic hash function which outputs 256-bit hash values. It processes message blocks of 256-bits and if the number of bits of the given message is not a multiple of 256 then an extra padding to the final block is added. A more detailed description of the GOST hash function is found in (GOST, 1994).

Its high level structure does more work than just following the Merkle-Damgård design principles used in common hash functions such as MD5 and SHA-1. In addition to the same iterated structure, it has is the extra checksum, which has to be computed over all input blocks and is an input to the very last hash block. On Figure 1 below we recall the structure of the GOST hash function where f stands for the compression function $GF(2)^{256+256} \rightarrow GF(2)^{256}$.

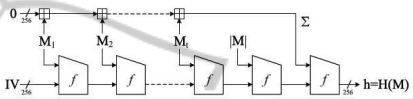

Figure 1: The structure of the GOST hash function.

The hash value $h = H(M)$ is computed recursively using the following formulas:

$$H_0 = IV \tag{1}$$

$$H_i = f(H_{i-1}, M_i) \tag{2}$$

$$H_{t+1} = f(H_t, |M|) \tag{3}$$

$$h = f(H_{t+1}, \Sigma) \tag{4}$$

where Σ is the sum of all the message blocks M_i computed modulo 2^{256}, IV is the fixed initial vector given to the compression function and where $|M|$ is the size in bits of the entire message.

In this paper, we focus on the properties of the compression function and we will work on how to construct a collision for the compression function without using any mathematical or structural properties of the GOST cipher. The compression function f consists of three basic components, the State update transformation, the Key generation and the Output transformation which are shown on Figure 2.

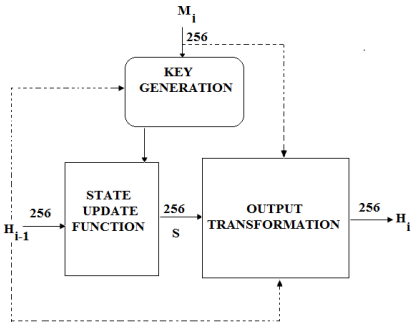

Figure 2: The three different components of the compression function.

From Figure 2 we observe that the intermediate hash value H_{i-1} affects all the three components of the compression function. In the following subsection we go through the full details of these three components which are appropriate for our security analysis of the GOST hash function.

2.1 State Update Transformation

The state update transformation as shown on Figure 3 is the "four encryptions in parallel" component of the underlying compression function. Here the GOST block cipher (E) is used four times in parallel to encrypt the intermediate hash value $H_{i-1} = h_3||h_2||h_1||h_0$ where $h_i \in \{0,1\}^{64}$ for $0 \leq i \leq 3$ and output a 256-bit value $S = s_3||s_2||s_1||s_0$ where $s_i \in \{0,1\}^{64}$ for $0 \leq i \leq 3$ as described below:

$$s_0 = E(k_0, h_0) \quad (5)$$

$$s_1 = E(k_1, h_1) \quad (6)$$

$$s_2 = E(k_2, h_2) \quad (7)$$

$$s_3 = E(k_3, h_3) \quad (8)$$

Here $E(K,P)$ is the encryption of the given plaintext P on 64-bits with a key K on 256-bits using the GOST cipher. The cost of evaluating the whole compression function depends essentially on the cost on these four encryptions, while as we will see later, all the other components are very inexpensive to implement, being very simple linear operations. Figure 3 gives a more detailed presentation of the compression function of the GOST hash function.

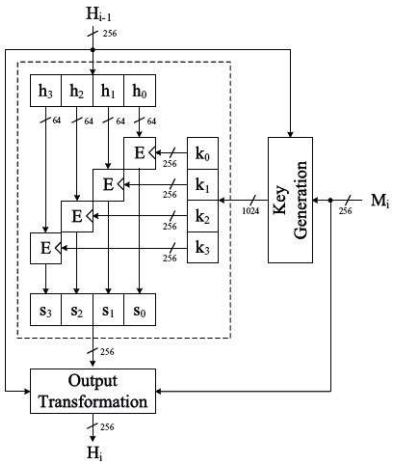

Figure 3: The compression function of GOST.

2.2 Key Generation

The key generation of GOST produces a 1024-bit key K using the intermediate hash value H_{i-1} and the message block M_i. The key is split in four 256-bit components $k_3||k_2||k_1||k_0$ and each component is computed using the following formulas:

$$k_0 = P(H_{i-1} \oplus M_i) \quad (9)$$

$$k_1 = P(A(H_{i-1}) \oplus A^2(M_i)) \quad (10)$$

$$k_2 = P(A^2(H_{i-1}) \oplus Const \oplus A^4(M_i)) \quad (11)$$

$$k_3 = P(A(A^2(H_{i-1}) \oplus Const) \oplus A^6(M_i)) \quad (12)$$

The maps A and P are linear transformations and *Const* is a constant. The fact that these maps are linear is the keystone to our attack since they force linear relations to hold between bits of the keys and bits of the messages and the intermediate hash value.

2.3 Output Transformation

The output transformation is the final transformation which is applied in order to produce the output value H_i of the compression function. The inputs given to the output transformation are the intermediate hash value H_{i-1}, the message block M_i and the output of the state update transformation S. The output value H_i is computed in the following way.

$$H_i = \psi^{61}(H_{i-1} \oplus \psi(M_i \oplus \psi^{12}(S))) \quad (13)$$

where ψ is a linear and invertible transformation from $\{0,1\}^{256}$ to $\{0,1\}^{256}$ given by:

$$\psi(X) = (x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_{12} \oplus x_{15})||x_{15}||x_{14}||...||x_1 \quad (14)$$

where the vector $X \in \{0,1\}^{256}$ is written as $x_{15}||x_{14}||....||x_0$ with each $x_i \in \{0,1\}^{16}$.
Since the map ψ is linear and invertible, equation (13) can be written as:

$$\psi^{-74}(H_i) = \psi^{-13}(H_{i-1}) \oplus \psi^{-12}(M_i) \oplus S \quad (15)$$

By the linearity of ψ we have that this equation induces linear equations on the bits of H_i which involve bits of H_{i-1}, M_i and S. Also the term $\psi^{-74}(H_i)$ is linearly related to H_i so any change in the bits of the latter term affects linearly the bits of the first term. This observation is the basic idea behind our construction of the collision for the compression function.

3 COLLISION ATTACKS ON THE GOST HASH FUNCTION

This section is divided in three main parts. In the first part we recall some basic facts about existing collision attacks on hash functions. Then we explain the collision attacks by Mendel *et al*(2008) from Crypto 2008. Then we present our new black-box collision attack on the GOST hash function.

3.1 Collision Attacks on GOST and other Hash Functions

As explained before, attacks on hash functions are either generic high-level structural attacks, or more specific attacks. In the case of generic or high-level attacks the hash function or a smaller component of it is treated as a black box and the running time of this type of attacks is evaluated in terms of how many time the hash function or another component of the hash function needs to be evaluated. An example of a generic attack but of very high complexity is a brute force attack in the case of finding a second preimage.

Specific attacks depend on what is inside specific components. For a hash function based on a block cipher, one example of a specific attack is a weak-key attack (Schneier, 1996). A weak key is a key with a specific structure which when used in a specific cipher makes the cipher to behave in a particular way. Weak keys do not matter that much in attacks on confidentiality encryption because very keys are usually random and weak keys will very rarely be able to occur in any realistic scenario. However weak keys do matter a lot in attacks on hash functions, where the attacker is able to somewhat choose the keys to be not random, and exploit the special cases where they become weak. Thus many weak-key attacks on block ciphers, hardly in encryption, can potentially become a serious threat when these ciphers are used to build hash functions.

Another important type of attacks on block ciphers which are by extension applicable and applied in hash function cryptanalysis are the fixed-point attacks. This was exploited at Crypto 2008 to break the GOST hash function.

3.2 Fixed Point Attacks in Cryptanalysis of GOST Hash

In the particular case of GOST interesting fixed points are those which are also symmetric plaintexts. This is due to a so called Reflection property (Courtois, b; Kara, 2008). This is what allows Mendel *et al*(2008) at Crypto 2008 to propose a first attack on the GOST hash function found so far. Their attack belongs to the category of the specific attacks exploiting the underlying GOST cipher and specifically the ability of the attacker to efficiently construct specific types of symmetric fixed points for the GOST cipher. Overall it allows to construct a collisions fro the full GOST hash with a total complexity of 2^{105} evaluations of the compression function. In what follows we are going to summarize this attack and look at particular structural and time complexity aspects of this attack. The attack consists of the following three main steps:

First Step. Find a collision on the underlying compression function of the GOST hash function. The objective of this attack is to fix s_0 where $s_0 = E(k_0, h_0)$ is the first instance of GOST encryption and fix some other linear combinations of inputs of the compression function, so that overall we will be able to reduce the entropy of $X = \psi^{-74}(H_i)$ by 64 bits which is to be achieved by fixing 64 rightmost bits of X, which is called x_0 to some constant value. In order to obtain this constant value x_0 a specific construction of fixed points for the fist instance of GOST block cipher is proposed with a third added condition which will be specified in terms of linear constraints on GOST key bits, and a forth condition that in all these fixed points the input is a fixed value,

Finally their construction leads to an enumeration of 2^{96} GOST keys k_0 which are all guaranteed to work for our fixed and chosen fixed point s_0. Then the attacker computes the compression function in each of these cases where the choice of k_0 also implies other conditions and allows him to choose pairs H_{i-1}, M_i which give this k_0, and additional conditions to impose that the input of the first GOST instance is the exact symmetric fixed point, and that the outputs of s_0 and 64 bits of H_i are meaningfully related.

In each of these 2^{96} pairs H_{i-1}, M_i we need to compute the GOST hash function, however since we are able to fix the input of the first GOST instance out of 4 and make it always produce the same output, it is easy to see that the cost of their attack is **LESS** than 2^{96} compression function evaluations as claimed in the paper, but only $3/4 \cdot 2^{96} = 2^{95.58}$ compression function evaluations. The same cost saving will be obtained in our attack.

Then by exploring collisions in the space where the compressed output value $X = \psi^{-74}(H_i)$ lies in a linear space of dimension $256 - 64 = 192$, we can finally construct a collision for the compression function with the complexity of about $2^{192/2} = 2^{96}$ evaluations.

Second Step. Use the collisions on the compression function to construct collisions in the iterative structure. To achieve these they use a standard technique with multicollisions. The result of this construction is a 2^{128} multicollision of about $128 \cdot (2^{96} + 2^{32}) = 2^{103}$ evaluations of the compression function.

Third Step. Construct a collision for the final sum exploiting multicollisions and the Camion-Patarin-Wagner generalized birthday paradox method (P. Camion, 1991; Wagner, 2002) and get a collision attack on the GOST hash function with an overall total complexity of 2^{105} evaluations of the underlying compression function.

3.3 A New Black-box Collision Attack on the Compression Function of the GOST Hash Function

In this section, we present a new collision attack on the GOST compression function with the same complexity of about $2^{95.58}$ evaluations of the compression function. The major novelty here is that the attack which we present is a generic attack and it does not exploit any weaknesses in the design or construction of the GOST cipher. Our attack works also if the GOST cipher is replaced by any other cipher with the same block size and the same key size.

First we establish our notation. We represent the compression function by the following map:

$$C : (M_i, H_{i-1}) \rightarrow H_i, \tag{16}$$

where $M_i, H_i, H_{i-1} \in \{0,1\}^{256}$ represent elements in the message space (M), compression function output space (G) and intermediate hash values space (H) respectively.

Then our aim is to find two distinct pairs (M_i, H_{i-1}) which give the same hash value. In our construction we will construct a linear subspace of $M \times H$ which is mapped under the compression function C to another linear output subspace of G with a small enough dimension. As in the previous attack, the dimension of this output space will be small enough so that the birthday attack allow us to find collisions. Unlike in the previous attack by Mendel *et al*(2008) the input space is also a linear space.

We do this by considering the subspace of $M \times H$ described by the following relations:

$$k_0 = k_1 \tag{17}$$

$$h_0 = h_1 \tag{18}$$

where $k_0, k_1 \in \{0,1\}^{256}$ are the rightmost bits of the key K and $h_0, h_1 \in \{0,1\}^{64}$ the rightmost bits of H_{i-1}. Equating bits of the key is reasonable since in the case of attacks on hash functions the attacker has full control over the key.

The bits of the key are linear functions of the bits of H_{i-1} and M. Thus relation (17) imposes 256 linear equations on the space $M \times H$ by equating each bit component of k_0 with the corresponding bit component of k_1, while relation (18) imposes 64 linear equations over M.

Additionally, if we consider the output transformation applied on the pair (H_{i-1}, M) we have that:

$$\psi^{-74}(H_i) = \psi^{-13}(H_{i-1}) \oplus \psi^{-12}(M_i) \oplus S \tag{19}$$

where ψ is a linear and invertible map, as in the previous attack.

Considering the encryption component of the cipher we have that $s_0 = s_1$ for the encrypted message $S = s_3||s_2||s_1||s_0$ since we have encryption of identical blocks under identical keys. In our overall attack, the cost of each step involves 2^{96} evaluations of the compression function, however will be under the conditions where two instances of the GOST cipher have identical keys and identical data, thus neglecting the cost of linear operations, very small compared to 4 evaluations of the GOST cipher done in each compression function, the actual cost of our construction construction will be only $3/4 \cdot 2^{96}$ evaluations of the compression function.

The fact that $s_0 = s_1$ induces a set of 64 linear equations which are satisfied by the bits of the output of the compression function. Thus this forces the dimension of the image to be 192.

In total we have 384 linear equations on $M \times H$. So the kernel of the linear map A which has as entries the coefficients of these linear equations has dimension at least 128 so there exist at least 2^{128} solutions to the system of these linear equations.

Hence we have constructed the restriction of the map C to a smaller subspace of $M \times H$ which contains at least 2^{128} elements. The resulting image of this subspace under the map C has dimension 192 since the dimension of the full space is 256 and we added another 64 linear relations. Hence by the birthday paradox if we sample $2^{192/2} = 2^{96}$ elements in this proper subspace of $M \times H$ we get a collision with a probability approximately $1/2$.

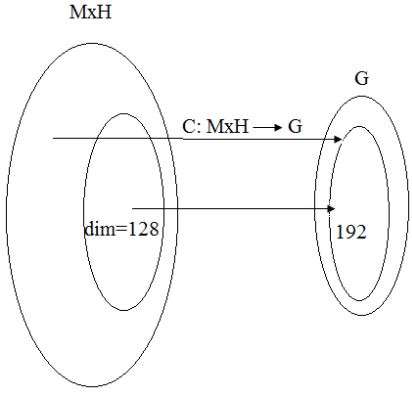

Figure 4: The restriction map that we have constructed from $\{0,1\}^{128}$ to $\{0,1\}^{192}$.

However, equations (17) and (18) suggest that only $3/4$ of the input-message needs to be encrypted since $s_0 = s_1$. This shows that our attack has complexity of $2^{96} \times 3/4$ of the compression function which is approximately $2^{95.58}$.

Note that it should be possible to prove that the map that we have constructed from $\{0,1\}^{128}$ to

$\{0,1\}^{192}$ which is a restriction map of the compression function to the subspace given by the equations above is not injective. It seems that the probability that it is not injective is very small. If we assume GOST is a pseudorandom permutation, then every bit of this map can be seen as a XOR of linearly independent linear combinations of the bits coming from three active blocks of the GOST block cipher, XORed with some linear combinations of input bits. A simpler way to actually check that this map is not injective will be to run our complete attack only once.

The attack can be summarized as follows:

1. Consider the following system of 384 equations over $GF(2)$:
   (a) $k_0 = k_1$
   (b) $h_0 = h_1$
   (c) $s_0 = s_1$ in $\psi^{-74}(H_i) = \psi^{-13}(H_{i-1}) \oplus \psi^{-12}(M_i) \oplus S$
2. Form the matrix A which has as entries the coefficients of the solutions (M,H) found for the system above.
3. Find a basis for the kernel of this matrix which has dimension at least 128. The kernel contains at least 2^{128} elements so there are at least so many solutions to the system of equations above.
4. Sample 2^{96} elements $r_1, r_2, ..., r_{2^{96}}$ which belongs to the space of solutions randomly.
5. Compute the corresponding hash values of these elements $H(r_1), H(r_2), ..., H(r_{2^{96}})$. Storing these outputs in a data structure such as a hash table which allows to search for collisions on the go and in constant time per every new value.
6. Output a collision. Because of the complexity and pseudo-random behaviour of the GOST encryption, the results $H(r_i)$ are expected to behave like random elements of our output space of dimension 192. Then the birthday paradox states that there is a high probability for a collision to be found by random sampling.

4 CONCLUSIONS

The GOST 34.11-94 hash function is an important national Russian government standard and a key component in the national Russian digital signature standard. At Crypto 2008 a collision attack on the GOST compression function requiring 2^{96} evaluations of this function was found. It was based on a specific weakness in the GOST block cipher such as the possibility to efficiently enumerate fixed-points for plaintexts with special properties and for 8 rounds of GOST, and the fact that this leads to a construction of many appropriate fixed points for 32 rounds of GOST. This attack on the compression function was extended to a collision attack with a complexity of 2^{105} evaluations for the whole GOST hash function.

In this paper we presented a new collision attack on the GOST compression function which is fundamentally different and more general than the attack published at Crypto 2008. Our new attack is a generic black-box attack which does not need any particular weakness to exist in the GOST block cipher, and works also if we replace GOST by another cipher with the same block and key size. In this way it is possible to see that from the strict point of view of the GOST compression function, the problem is not that much the GOST cipher itself which is now known to be weak in more than one way (Isobe, 2011; Courtois, b; Courtois, a). We have demonstrated that there is also a definite flaw in the way in which the cipher is used here. This in particular is really due to the self-similarity of the high level structure, with 4 identical encryption blocks, where the attacker can try to exploit the self-similarity to force these blocks to be in the same state. The attack is also helped by the linearity of certain components such as the internal key derivation function. Thus we are able to find collisions on the GOST compression function which do not exploit the internal structure of any particular cipher.

The importance of self-similarity in symmetric cryptanalysis remains under-estimated by designers, and it is a source of plenty of new cryptographic attacks every year. For example in the numerous very recent attacks which allow to break the GOST cipher in encryption (Isobe, 2011; Courtois, b; Courtois, a). In attacks on hash functions the attacker has even more freedom to exploit this self-similarity. The easiest way to avoid this type of attacks in the design of hash functions will be to remove this structural self-similarity, for example by using a block cipher with large blocks, rather than four copies of the same block cipher. However a block cipher with very large blocks would be very costly while GOST is a block cipher which is exceptionally economical in implementation, see (A. Poschmann, 2010). There is another simple way to avoid this type of self-similarity attack: simply to use a different set of S-boxes in each instantiation of the GOST block cipher.

Our new attack is also slightly faster than the previous attack, and the complexity is only $2^{95.6}$ evaluations of the compression function. Moreover we also slightly improve the complexity of the previous attack by remarking that each compression in their attack can also be computed more efficiently.

ACKNOWLEDGEMENTS

We would like to thank the anonymous referees of this paper who helped us a lot to improve it.

REFERENCES

A. Poschmann, S. Ling, H. W. (2010). 256 bit standardized crypto for 650 ge gost revisited. In *CHES 2010 Proceedings*.

Courtois, N. Algebraic complexity reduction and cryptanalysis of gost. Unpublished manuscript, 17 February 2011. 28 pages. MD5=d1e272a75601405d156618176cf98218.

Courtois, N. Security evaluation of gost 28147-89 in view of international standardisation. Unpublished manuscript, 2011. Available: http:// www.nicolas courtois.com/papers/gostreport.pdf (2011/05/01).

Damgård, I. (1990). A Design Principle for Hash Functions. In Brassard, G., editor, *Advances in Cryptology – CRYPTO '89, Proceedings*, volume 435 of *LNCS*, pages 416–427. Springer.

F. Mendel N. Pramstaller, C. Rechberger, M. K. J. S. (2008). Cryptanalysis of the GOST Hash Function. In Wagner, D., editor, *Advances in Cryptology – CRYPTO 2008, Proceedings*, volume 5157 of *LNCS*, pages 162–178. Springer.

GOST, C. R. F. (1994). GOST R 34.11-94, the Russian hash function standard. *Government Standard of the Russian Federation, Government Committee of Russia for Standards, in Russian*. English translation by Michael Roe available as gost34.11.ps inside: http://www.autochthonous.org/crypto/gosthash.tar.gz.

I.A. Zabotin, G.P. Glazkov, V. I. (1989). Gost 28147-89, cryptographic protection for information processing systems. *Government Standard of the USSR, Government Committee of the USSR for Standards, in Russian*. English translation gost28147.ps by Aleksandr Malchik available inside: http://www.autochthonous.org/crypto/gosthash.tar.gz.

Isobe, T. (2011). A single-key attack on the full gost block cipher. In *Fast Software Encryption 2011, Proceedings*, LNCS. Springer.

J. Talbot, D. W. (2006). *Complexity and Cryptography*. Cambridge University Press, Cambridge, 1st edition.

Kara, O. (2008). Reflection cryptanalysis of some ciphers. In *Indocrypt 2008 Proceedings*, volume 5365 of *LNCS*, pages 294–307. Springer.

P. Camion, J. P. (1991). The Knapsack Hash Function proposed at Crypto'89 can be broken. In Davies, D. W., editor, *Advances in Cryptology – EUROCRYPT '91, Proceedings*, volume 547 of *LNCS*, pages 39–53. Springer.

P. Gauravaram, J. K. (2008). Linear-XOR and Additive Checksums Don't Protect Damgård-Merkle Hashes from Generic Attacks. In Malkin, T., editor, *Topics in Cryptology – CT-RSA 2008*, volume 4964 of *LNCS*, pages 36–51. Springer.

Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Willey, New York, 2nd edition.

Wagner, D. (2002). A Generalized Birthday Problem. In Yung, M., editor, *Advances in Cryptology – CRYPTO 2002, Proceedings*, volume 2442 of *LNCS*, pages 288–303. Springer.