Meet in the Middle Attacks on Reduced Round Kuznyechik

Riham AlTawy and Amr M. Youssef

Concordia Institute for Information Systems Engineering,
Concordia University, Montréal, Québec, Canada

Abstract. Kuznyechik is an SPN block cipher that has been recently chosen to be standardized by the Russian federation as a new GOST cipher. The algorithm updates a 128-bit state for nine rounds using a 256-bit key. In this paper, we present meet-in-the-middle attacks on the 4 and 5 round reduced cipher. Our attacks are based on the differential enumeration approach, where we propose a distinguisher for the middle rounds and match a sequence of state differences at its output. However, the application of the exact approach is not successful on Kuznyechik due to its optimal round diffusion. Accordingly, we adopt an equivalent representation for the last round where we can efficiently filter ciphertext pairs and launch the attack in the chosen ciphertext setting. We also utilize partial sequence matching which further reduces the memory and time complexities through relaxing the error probability. The adopted partial sequence matching approach enables successful key recovery by matching parts of the generated sequence instead of the full sequence matching used in the traditional settings of this attack. For the 4 and 5 round reduced cipher, the 256-bit master key is recovered with a time complexity of $2^{139.6}$ and $2^{140.3}$, and a memory complexity of $2^{24.6}$ and $2^{153.3}$, respectively. Both attacks have similar data complexity of 2^{113}.
Keywords: Kuznyechik, Cryptanalysis, Meet-in-the-middle attacks, Differential enumeration, Partial sequence matching, GOST.

1 Introduction

The Russian Federation has recently published a project of a new standard for block cipher encryption algorithm [2]. A draft for this new algorithm was presented by its designers at CTCrypt 2014 [21]. The new algorithm, Kuznyechik, (Grasshopper in Russian), is in the process of being standardized [2] to accompany the current encryption standard GOST 28147-89 [1]. Although the current standard is considered a lightweight cipher [19], and only theoretical attacks on the full round cipher have been presented [17, 13], it operates on 64-bit blocks of data which is not sufficient for the current requirements. Hence, the need arose for a new standard with larger block length which is intended to supersede the current GOST 28147-89 cipher.

The meet-in-the-middle (MitM) attack was first proposed in 1977 by Diffie and Hellman [12] for the analysis of the Data Encryption Standard (DES). Ever since, the attack has been evolving to cryptanalyze block ciphers such as Present and Prince [7], KTANTAN [6], LBlock [3], and mCrypton [15]. Additionally, MitM preimage attacks on hash functions have been presented on HAS-160 [16], Whirlpool [20], and Streebog [4]. The first application of the MitM attack on AES was due to the work of Demirci and Selçuk [9], whose approach opened the door to a new line of research. They constructed a truncated differential four round distinguisher, and showed that if the input to the distinguisher has only one active byte that takes all the possible values, then each output byte can be evaluated as a function of 25 parameters. They also showed that the values of each output byte corresponding to the input byte values form an ordered sequence that can be used as a property to identify the right key guess. The main disadvantage of their technique is the high memory complexity which is required by a precomputation table to store all the sequences resulting from all the possible combinations of the 25 byte parameters. Accordingly, the approach was only valid to attack seven and eight rounds of AES-192 and AES-256, and not the 128-bit version. Afterwards, the number of parameters was reduced to 24 bytes in [10], which lowered the size of the table by a factor of 8.

Later on, Dunkelman *et al.* proposed the idea of multisets and differential enumeration [14] to tackle the high memory requirements of the approach of Demirci and Selçuk [9]. While the concept of multisets provides better encoding of the ordered sequence which reduces the size of the table by a factor of 4, differential enumeration is considered the main advantage of their attack. More precisely, differential enumeration allows the ordered sequence to be generated by the knowledge of 16 byte parameters only instead of 24, which brings the number of entries of the table down from 2^{192} to 2^{128} and makes the attack possible on AES-128. This gain is attributed to the use of a low probability truncated differential distinguisher where the generated sequences at its output can only take a restricted number of values. Accordingly, one must initially search through a large amount of input data pairs to find one pair that satisfies the chosen distinguisher. Indeed, their proposal has reduced the memory complexity of the attack at the expense of its data complexity required to search for the right input data pair.

Finally, Derbez *et al.* [11] improved the attack of Dunkelman *et al.* by borrowing ideas from the rebound attack [18], and proving that not all of the sequences in the table can be verified by input data satisfying the truncated distinguisher. They have presented an efficient enumeration technique and showed that the whole set of sequences can take only 2^{80} values and not 2^{128} as with the case in the attack by Dunkelman *et al.* Accordingly, all the generated sequences require the knowledge of only 10 byte parameters, thus the number of entries of the precomputation table is further reduced to 2^{80}. A direct consequence of their improvement is that the memory complexity is not the bottleneck of the attack anymore but both the time and data complexities are. Nevertheless, their attack is considered the most efficient attack on the 7-round reduced AES-128 and

8-round reduced AES-196/256. They have also used a 5-round distingusher to attack the 9-rounds reduced AES-256.

In this work, we present a MitM attack on Kuznyechik using the idea of efficient differential enumeration. Unlike AES, Kuznyechik employs an optimal diffusion transformation applied to the whole state, where one byte difference results in a full active state with certainty after one round. Consequently, we construct two and three round distinguishers in our attacks to recover 16-bytes of the master key of the reduced 4 and 5 round cipher. The direct application of the attack on Kuznyechik requires a time complexity that exceeds that of the exhaustive search for the 256-bit key, which is also attributed to the optimal round diffusion. Accordingly, we adopt an equivalent representation of the last round which allows us to efficiently select ciphertext pairs that satisfy the lower half of the differential path used in the attack with certainty. Hence, our attacks are considered in the chosen ciphertext setting. This modification lowers the time complexity of the online phase by a factor of 2^{120} because we eliminate the probabilistic propagation of the 16 to 1 transition through the linear transformation from the ciphertext side. We also present partial sequence matching, by which we generate, store, and match parts of the ordered sequence while maintaining negligible probability of error. Indeed, not only we decrease the partially encrypted/decrypted data during online matching and thus the overall time complexity of the attacks is lowered, but this approach also reduces the memory requirements of both attacks.

The rest of the paper is organized as follows. In the next section, the description of the Kuanyetchik block cipher along with the notation used throughout the paper are provided. Afterwards, in section 3, we provide a detailed description of the proposed distinguisher, the adopted attack procedure, our filtering approach, the proposed partial sequences idea, and show how to extend the attack to five rounds. Finally, the paper is concluded in section 4.

2 Specification of Kuznyechik

Kuznyechik [21, 2] is an SPN block cipher that operates on a 128-bit state. The cipher employs a 256-bit key which is used to generate ten 128-bit round keys. As depicted in Figure 1, the encryption procedure updates the 16-byte state by iterating the round function for nine rounds. The round function consists of:

- SubBytes (S): A nonlinear byte bijective mapping.
- Linear Transformation (L): An optimal diffusion operation that operates on a 16-byte input and has a branch number = 17.
- Xor layer (X): Mixes round keys with the encryption state.

Additionally, an initial XOR layer is applied prior to the first round. The full encryption function where the ciphertext C is evaluated from the plaintext P is given by:

$$C = (X[K_{10}] \circ L \circ S) \circ \cdots \circ (X[K_2] \circ L \circ S) \circ X[K_1](P)$$

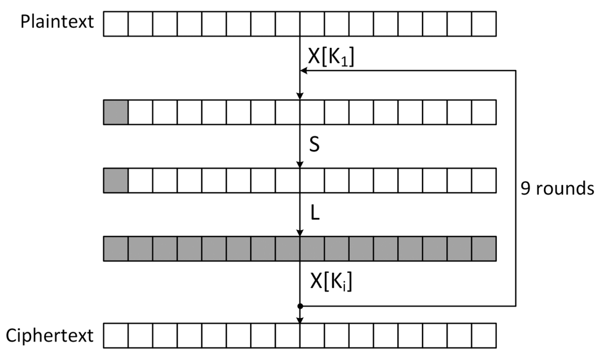

Fig. 1. Encryption procedure

In our attack, we use an equivalent representation of the round function. The representation exploits the fact that both the linear transformation, L, and the Xor operation, X, are linear and thus, their order can be swapped. One has to first Xor the data with an equivalent round key, then apply the linear transformation, L, to the result. We evaluate the equivalent round key at round i by $EK_i = L^{-1}(K_i)$. We also use the following property of the Sbox:

Property 1. For a given Sbox differential $(\delta x, \delta y)$, the average number of solutions to $S(x) \oplus S(x \oplus \delta x) = \delta y$ is 1.

For further details regarding the employed Sboxes and linear transformation, the reader is referred to [21].

Key schedule: The ten 128-bit round keys are derived from the 256-bit master key by undergoing 32 rounds of a Feistel structure function. The first two round keys, K_1 and K_2, are derived directly from the master key, K, as follows: $K_1 \parallel K_2 = K$. As depicted in Figure 2, each pair of subsequent round keys is extracted after eight rounds of execution. During each round, the same round function used in the encryption procedure is applied to the right half of the input to the Feistel round. However, round constants are used with the X operation instead of round keys.

The 128-bit round constants C_i are defined as follows: $C_i = L(i)$, $i = 1, 2, \cdots, 32$. Let $F[C](a, b)$ denote $(LSX[C](a) \oplus b, a)$, where C, a, and b are 128-bit inputs. The rest of the round keys are derived from the first two round keys, K_1 and K_2, as follows:

$$(K_{2i+1}, K_{2i+2}) = F[C_{8(i-1)+8}] \circ \cdots \circ F[C_{8(i-1)+1}](K_{2i-1}, K_{2i}),\ i = 1, 2, 3, 4.$$

It is interesting to note that Kuznyechik bares a lot of resemblance with one of the AES predecessors, Khazad [5]. In particular, both ciphers employ an iterative SPN structure for updating the input block state, where the adopted

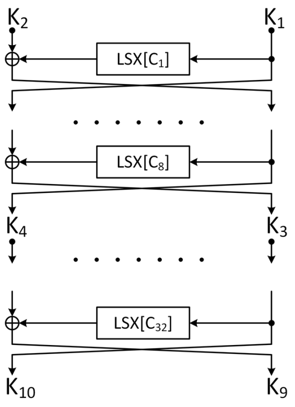

Fig. 2. Key schedule

linear transformation has an optimal diffusion properties. Also, they both use Fiestel network for the round keys generation. While in Kuznyechik, two round keys are generated after eight rounds of execution, only one round of execution separates consecutive round keys in Khazad. They also differ in that Khazad employs involution Sboxes and linear transformation, and it does not use a linear transformation in the last round.

Notation The following notation is used throughout the paper:

- x_i, y_i, z_i: The 16-byte state after the X, S, L operation, respectively, at round i.
- x_i^j: The state at round i whose position within a set or a sequence is given by j.
- $x_i[j]$: The j^{th} byte of the state x_i, where $j = 0, 1, \cdots, 15$, and the bytes are indexed from left to right.
- Δx_i, $\Delta x_i[j]$: The difference at state x_i, and byte $x_i[j]$, respectively.

The memory complexity of our attacks is given in 16-byte states and the time complexity is evaluated in reduced round Kuznyechik encryptions. In the following section, we give the details of our MitM attack on Kuznyechik.

3 A MitM Attack using Differential Enumeration on Kuznyechik

Generally, our attack divides the reduced Kuznyechik block cipher, C_K, into three parts, such that $C_K = C_{k_2} \circ C^m \circ C_{k_1}$, where C^m is the middle part of the cipher which exhibits a distinguishing property. The employed property is evaluated without the knowledge of the key bits used in these middle rounds. Hence, correct round key candidates for k_1 and k_2 are checked if they verify this distinguishing property or not. Our middle distinguisher is a truncated differential characteristic such that, when a set of input states from a δ*-set* [8] is presented as its input, the set of each byte of the output state forms an ordered sequence.

Definition 1. (δ*-set* of Kuznyechick) *is a set of 256 states where one byte at a particular position takes all the* 2^8 *possible values and the rest of the 15 bytes are constants.*

In our MitM attacks, we employ distinguishers where the δ*-set* is presented at their input from the ciphertext side, thus, after partially decrypting it, we acquire the corresponding ordered sequence. We denote the δ*-set* at state x_i resulting from changing the byte at position j by δs^j, $j = 0, 1, \cdots, 15$, where

$$\delta s^j = \{x_i^0, x_i^1, \cdots, x_i^{255}\}.$$

We also denote the set of 255 differences at byte $x_{i-r}[k]$ which form the ordered sequence for an r round distinguisher by os^k, $k = 0, 1, \cdots, 15$, where

$$os^k = \{\Delta^1 x_{i-r}[k], \Delta^2 x_{i-r}[k], \cdots, \Delta^{255} x_{i-r}[k]\},$$

and $\Delta^l x_{i-r}[k] = x_{i-r}^0[k] \oplus x_{i-r}^l[k]$, for $l = 1, 2, \cdots, 255$. The correct ordered sequence os^k is evaluated by partially decrypting the 256 bytes which are different in the δ*-set* for r rounds. However, we compute all the possible sequences so that one does not need to know the key bits involved in this encryption process because we simply compute it using all the possible values of the involved parameters.

3.1 The 4-round Attack

Our proposed 4-round MitM attack employs a two round distinguisher. Figure 3 depicts the differential path used in the attack in which we embed a $1 \to 16 \to 1$ distinguisher that starts at x_4 and ends at x_2. The length of the distinguisher is restricted by the properties of optimal linear transformation used in the Kuznyechik round. Unlike the MixColumn transformation used in AES which works on independent columns leading to full state diffusion after two rounds, the linear transformation L guaranties full diffusion in one round. As depicted in Figure 3, our δ*-set* is the set of states resulting from changing the first byte at state x_4 and is given by:

$$\delta s^0 = \{x_4^0, x_4^1, \cdots, x_4^{255}\}.$$

The corresponding ordered sequence:

$$os^0 = \{\Delta^1 x_2[0], \Delta^2 x_2[0], \cdot, \Delta^{255} x_2[0]\}$$

is evaluated by the knowledge of the values of 18 bytes. More precisely, in addition to δs^0, given the values of 1 byte at $x_4[0]$, 16 bytes at y_3, and 1 byte at $y_2[0]$, the i^{th} element, $\Delta^i x_2[0]$ of the ordered sequence is computed as follows:

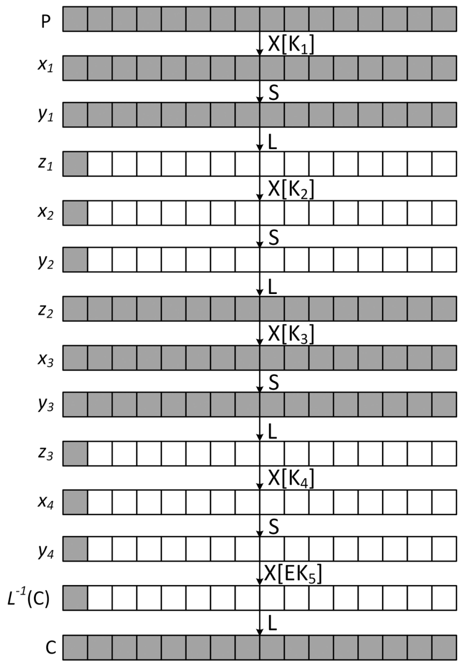

Fig. 3. Differential path used in the 4-round attack.

– Compute $\Delta^i x_4[0] = x_4^0[0] \oplus x_4^i[0]$.
– Linearly propagate $\Delta^i x_4[0]$ backwards and compute the value of $\Delta^i y_3$.
– Using the value of y_3 and $\Delta^i y_3$, pass the substitution layer with certainty and evaluate $\Delta^i x_3$
– Linearly propagate $\Delta^i x_3$ backwards through both $X[K_3]$ and L to evaluate $\Delta^i y_2[0]$.
– Using $\Delta^i y_2[0]$ and $y_2[0]$, compute $\Delta^i x_2[0]$.

However, by employing the rebound based differential enumeration technique [11], we deduce that if x_4^0 of δs^0 belongs to a pair of messages that follows the differential path in Figure 3, then the corresponding ordered sequence os^0 can have only 2^{24} values. Accordingly, a given ordered sequence can be computed by the knowledge of 3 byte parameters only. These parameters are $\Delta x_4[0]$, $y_2[0]$, and $\Delta y_2[0]$, where $\Delta x_4[0]$ and $\Delta y_2[0]$ denote the differences generated by a conforming message pair. In what follows, we give the details of the attack steps and explain how we evaluate the 2^{24} sequences from these 3 parameters.

Attack Procedure: The attack recovers the 128-bit first round key K_1 and one byte of $EK_5 = L^{-1}(K_5)$. The fact that K_1 is half the master key, K, enables us to recover the whole master key by exhaustively searching for the other half. The benefit of the extra knowledge of the recovered byte of EK_5 is limited to making the exhaustive search for the rest of the master key more efficient (by early aborting the round keys generation). More precisely, the key schedule employs a large number of rounds between the generation of sequential round keys, which leads to a very complex relation between them and renders any key bridging approaches useless.

The attack is composed of precomputation and online phases. In the precomputation phase, we iterate on all the values of the parameters required for evaluating the ordered sequence, and for each value, we deduce its corresponding 18 bytes values which are then used to generate the ordered sequence. We store all the sequences in a hash table. The online phase is further divided into data collection and filtration, and key recovery phases. In the data collection phase, we collect many pairs such that one of them satisfies the 4-round differential characteristics given in Figure 3. However, given the fact that our required ciphertext pairs are fully active, we employ an equivalent representation of the last round to enable efficient filtering by which we are certain that the obtained ciphertext pairs satisfy the lower two rounds of the differential characteristic. Finally, in the key recovery phase, for each of the obtained pairs, we compute the ordered sequences by deducing the first round key K_1 and guessing the first byte of EK_5. We then search for a match between the online computed sequence and the ones stored in the precomputed table, which enables the recovery of K_1.

Precomputaion phase: In this phase, we construct a lookup table that contains the 2^{24} ordered sequences of the resulting 255 difference,

$$os^0 = \{\Delta^1 x_2[0], \Delta^2 x_2[0], \cdots, \Delta^{255} x_2[0]\},$$

from the $\delta s^0 = \{x_4^0, x_4^1, \cdots, x_4^{255}\}$. This stage is done by first iterating on the 2^{24} possible values for the 3 bytes $\Delta x_4[0]$, $y_2[0]$, and $\Delta y_2[0]$, and for each one of them, we deduce the possible values of the 18 original parameters using the rebound approach. Then, for each of them, we construct the ordered sequence of 255 differences. The procedure can be summarized as follows:

1. For each of the 2^{24} possible values of $\Delta x_4[0] \parallel y_2[0] \parallel \Delta y_2[0]$, evaluate the values of the 18 bytes required to compute the ordered sequence, which are $y_2[0]$, y_3, and $x_4[0]$, as follows:
   – Using $\Delta x_4[0]$, evaluate Δy_3 by linearly propagating $\Delta x_4[0]$ backwards.
   – Compute Δx_3 by linearly propagating $\Delta y_2[0]$ through the linear transformation.
   – Find x_3, such that $S(x_3) \oplus S(x_3 \oplus \Delta x_3) = \Delta y_3$. According to property 1 in Section 2, not all the 2^{24} differential are possible, but the ones that are possible results in about 2^{16} solutions on average so we get one solution on average.
   – Evaluate $y_3 = S(x_3)$.
2. The additional knowledge of the evaluated value of y_3 provides us with the values of the 18 bytes required to compute the 255 differences $\Delta^l x_2[0]$, $l = 1, 2, \cdots, 255$, of the ordered sequence as described at the beginning of this section.
3. Store all the generated sequences in a hash table.

Online phase: In this phase, we first find enough pairs of messages such that one of them conforms to the truncated differential characteristic in Figure 3. In this step we introduce a modification to the default process of data collection [11, 14]. More precisely, instead of collecting enough random pairs with full active states so that one of them satisfies the two $16 \rightarrow 1$ transitions through the linear transformation in rounds 1 and 4, we start data collection from the ciphertext side and employ an equivalent representation of the last round. During this stage, we commence by composing structures of the inverse linear transformation of ciphertext that have all the 2^8 possible values in one byte while the other bytes are constants. Accordingly, even though their corresponding ciphertext pairs have full active state, these differences guaranty the $16 \rightarrow 1$ transition through the linear transformation. Hence, we have to repeat this filtration stage enough times so that we satisfy only the probabilistic transition in round 1. A direct consequence of our modification is that instead of requiring 2^{240} pairs, the attack is applicable with 2^{120} pairs only, thus both the data and time requirements of the attack are lowered by this difference. The second step uses the found pairs to create a set of sequences and test them against the precomputed table to identify the correct K_1.

Data collection and filtration: In this step, we query the decryption oracle with structures of chosen ciphertexts to get enough pairs such that one conforms to the whole truncated differential path. Each structure is composed of 256 ciphertext, where the first byte after applying the inverse linear transformation on them takes all the 256 values and the remaining fifteen bytes are equal. The process is described as follows:

1. To get one ciphertext structure, randomly pick the value of the rightmost fifteen bytes of $L^{-1}(C)$ and let the first byte take all the possible 256 values. This structure generates about $\frac{2^8 \times (2^8-1)}{2} \approx 2^{15}$ pairs. This step guaranties that all the corresponding (C, C') pairs in the structure conform to the $16 \rightarrow 1$ transition in round 4.
2. Query the decryption oracle for the plaintext pairs (P, P') corresponding to the ciphertext pairs generated in step 1. These pairs are not necessarily going to conform to the $16 \rightarrow 1$ transition in round 1, which happens with probability 2^{-120}.
3. Store the 2^8 plaintexts and their corresponding ciphertexts in a hash table.
4. To get one pair of plaintexts (P, P') that satisfy the $16 \rightarrow 1$ probabilistic transition, we need to try approximately 2^{120} pairs. Since, each structure provides 2^{15} pairs, one requires about 2^{105} structures, and hence the above two steps are repeated 2^{105} times.

All in all, we ask for the decryption of $2^{105} \times 2^8 = 2^{113}$ chosen ciphertexts to get the required 2^{120} pairs.

Key recovery: The previous steps results in 2^{120} candidate pairs (P_i, C_i) and (P'_i, C'_i), for $i = 0, 1, \cdots, 2^{120} - 1$, with a plaintext difference, $\varDelta P_i = P_i \oplus P'_i$, and a predetermined ciphertext difference, $\Delta C_i = C_i \oplus C_i'$. For each pair, we deduce the possible values of K_1 and guess the value of $EK_5[0]$ to compute a candidate ordered sequence and match it against the precomputed table, and thus determine the value of the right K_1. The following process describes the method adopted for the recovery of the first round key, and it is repeated for each plaintext pair (P_i, P_i') and their corresponding ciphertext pair (C_i, C_i').

1. Guess a value for $\Delta x_2[0]$, and linearly propagate it backwards to get the value of Δy_1.
2. Using the fact that $\Delta x_1 = \Delta P_i$, find the value of x_1 which provides a solution for $\Delta y_1 = S(x_1) \oplus S(x_1 \oplus \Delta x_1)$. According to property 1 in section 2, we get one solution on average.
3. Evaluate $K_1 = P_i \oplus x_1$. By repeating the previous two steps for all the possible guesses of $\Delta x_2[0]$, we get 2^8 candidate values for K_1.
4. For each candidate of the 2^8 values of K_1 and for each guess of the 2^8 guesses of $EK_5[0]$, use C_i to get the rest of the 255 ciphertexts C_i^j, for $j = 1, 2, \cdots, 255$, corresponding to the δs^0 generated by C_i as follows:
   - The value of $x_4[0]$ which is the first byte of the first state in δs^0 is evaluated as follows:
   $$x_4[0] = S^{-1}(L^{-1}(C_i)[0] \oplus EK_5[0]).$$
   - The set of different values of $x_4[0]$ in the states of δs^0 has the following structure:
   $$\{x_4[0], x_4[0] \oplus \Delta_1, x_4[0] \oplus \Delta_2, \cdots, x_4[0] \oplus \Delta_{255}\},$$
   and $\Delta_j = j$ for $j = 1, 2, \cdots, 255$. Accordingly, we can evaluate the 255 values of $L^{-1}(C_i^j)[0]$ corresponding to the values of $x_4[0] \oplus \Delta_j$ by
   $$L^{-1}(C_i^j)[0] = S(x_4[0] \oplus \Delta_j) \oplus EK_5[0].$$
   - Get the difference $\Delta^j L^{-1}(C_i)[0] = L^{-1}(C_i^j)[0] \oplus L^{-1}(C_i)[0]$, and propagate it through the linear transformation to get the corresponding difference $\Delta^j C_i$. Finally, the required 255 values of C_i^j are evaluated by $C_i^j = C_i \oplus \Delta^j C_i$.
5. Get the 256 plaintexts $(P_i, P_i^1, \cdots, P_i^{255})$ corresponding to the ciphertexts generated in the previous step using K_1 from the currently stored structure.
6. Partially encrypt the plaintexts $(P_i, P_i^1, \cdots, P_i^{255})$ to get the 256 values of $\Delta^j x_2[0]$, which form the ordered sequence os^0.
7. Check if there is a match between the computed os^0 and the 2^{24} ordered sequences stored in the precomputed table. If there is no match we discard the candidate K_1 with certainty.

The probability of a wrong key producing a valid 255 byte ordered sequence is given by $2^{24+120+16-2040} = 2^{-1880}$, which is negligible and can be relaxed. This fact allows us to present our partial sequence matching idea.

Complexity Analysis: The memory complexity of the attack is attributed to the precomputed table required for the storage of 2^{24} sequences of size 2040 bits each. Thus the memory requirements of the attack is given by $2^{24} \times 2040/128 \approx 2^{28}$ 128-bit states. That memory complexity can be reduced by a factor of 4 using the multiset encoding idea (cf. Appendix A in [11]), where 512-bits are used to store the required information of the 255 bytes in a sequence. The data complexity of the attack is due to the data collection step where we query the decryption oracle with 2^{113} chosen ciphertexts. The time complexity for recovering the first round key is dominated by the time required for partially encrypting the 256 values in a *δ-set* with all the 2^{16} key candidates for all the 2^{120} collected pairs. Accordingly, the time complexity of the attack $\approx 2^{(120+16+8)} \times 2^{-1} = 2^{143}$.

As it is fairly complex to deduce any relation between the recovered K_1 and $EK_5[0]$ that can aid us in the recovery of K_2, which is the second half of the master key, we are left with two options. First, with the knowledge of the recovered K_1, we can add one round at the beginning, and repeat the attack on the following four rounds to recover K_2. Otherwise, our second option is to exhaustively search for K_2. Comparing the complexities of both options, we opt for the second one. Thus, the memory, data, and time complexities required for the recovery of the 256-bit KuznyechiK key are given by 2^{26}, 2^{113} and $2^{143} + 2^{128} \approx 2^{143}$, respectively. In what follows, we present the idea of partial sequence matching by which we reduce both the memory and time complexities of the attack.

Partial sequence matching: Our proposed 4-round attack has a time complexity of 2^{143}, which is affected by the number of partial encryptions/decryptions required to generate the $2^8 - 1$ differences in the ordered sequence from the *δ-set*. Accordingly, instead of partially encrypting the 2^8 values of the *δ-set* to get their corresponding ciphertexts, and then encrypting 2^8 plaintexts to evaluate the ordered sequence, we can reduce the number of encryption/decryption operations to b, where $b < 2^8$ and denotes the number of differences stored in the ordered sequence. In other words, since the probability of error is so small, it can be relaxed such that we match b bytes of the 2^8 of the ordered sequence to identify the right key. More precisely, if we accept the error probability to be 2^{-32}, which is still negligible, the required number of bytes, b, is evaluated by $2^{-32} = 2^{120+16+24-8b}$. Hence, it is enough to match 24 bytes of the ordered sequence to identify a right key with an error probability of 2^{-32}. In the sequel, the memory complexity of the attack is reduced to $2^{24} \times (192/128) \approx 2^{24.6}$ states, and the time complexity is evaluated by $2^{120+16} \times 2^{4.6} \times 2^{-1} \approx 2^{139.6}$. In what follows, we show how this attack can be extended to cover one more round of the cipher.

3.2 Extending the Attack to 5 Rounds

Adding an additional round to our attack can be accomplished by either appending one round from the plaintext or ciphertext side, or by adopting a three round distinguisher. We opted for employing a longer distinguisher because the idea of adding one round from either ends makes the attack inapplicable. Indeed, if we add one round from the plaintext side, we have to deal with all the possible guesses for the 128-bit value of K_1, and then deduce the value of K_2 via our attack. On the other hand, by appending one round from the ciphertext side, we are not able to use our online filtering stage, which raises the time complexity of the attack by a factor of 2^{120}.

As depicted in Figure 4, we propose a $1 \rightarrow 16 \rightarrow 16 \rightarrow 1$ distinguisher that covers rounds two to five. The additional round in the distingusher in-

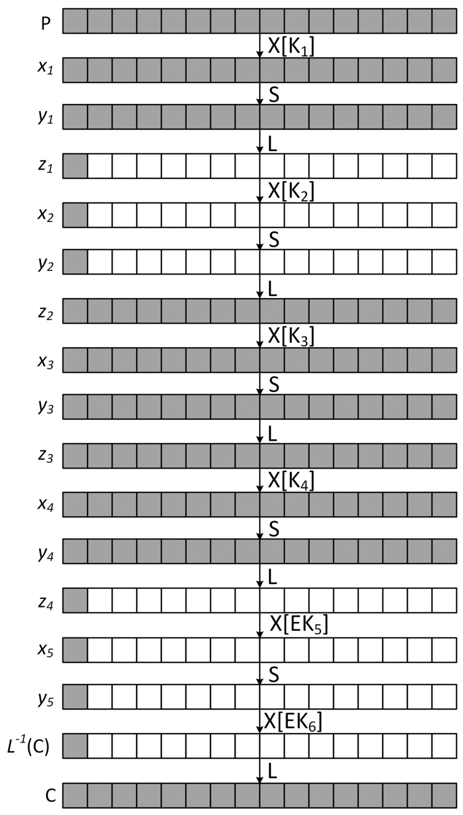

Fig. 4. Differential path used in the 5-round attack.

crease only the memory requirements of the attack because now the sequences are generated by the knowledge of 34 byte parameters. However, using the differential enumeration approach, the ordered sequences can only have 2^{152} values. Accordingly, we need only 19 byte parameters, namely $\Delta x_5[0]$, y_4, $y_2[0]$, and $\Delta y_2[0]$ to deduce a given sequence. In the precomputation phase, we build a table of size $2^{152} \times 2040/128 \approx 2^{156}$ states. The online phase is exactly like that of the 4-round attack, and hence, the time complexity is given by

$2^{120+16+8} \times (2/5) \approx 2^{143}$ and the data complexity is evaluated by 2^{113}. However, employing partial sequence matching, an error probability of 2^{-32} is achieved by matching 40 bytes of the ordered sequence. Accordingly, the memory complexity is given by $2^{152} \times 320/128 \approx 2^{153.3}$, and the time complexity is evaluated by $2^{120+16} \times 2^{5.3} \times 2^{-1} \approx 2^{140.3}$.

4 Conclusion

In this paper, we have presented MitM attacks on the new draft of the Russian encryption standard, also known as Kuznyechik, using the idea of efficient differential enumeration. We have proposed an initial filtration stage which lowers the time complexity of the basic approach by a factor of 2^{120}. Instead of trying random data pairs such that the truncated differential path is satisfied probabilistically, we carefully compose ciphertext pairs so that the lower half of the path is conformed to with certainty. Additionally, we have adopted partial sequence matching, by which, we store and match parts of the ordered sequences while maintaining a negligible probability of error which reduces both the memory and time complexities of the attacks. Our attacks on the 4 and 5 round reduced cipher have a memory complexity of $2^{24.6}$ and $2^{153.3}$, and a time complexity of $2^{139.6}$ and $2^{140.3}$, respectively. Both attacks have similar data complexity of 2^{113}.

It should be noted that several improvements, like key bridging techniques [11], for this class of attacks on AES were possible because of the relatively simple key schedule. This is unlikely to be the case for Kuznyechik, given the large number of rounds used in the generation of the round keys, which despite its conceptual simplicity leads to a very complex relation between successive round keys. While these attacks may not present direct threat to the security of Kuznyechik, they are considered forward steps in the public cryptanalysis of this soon to be the new Russian block cipher standard.

References

1. GOST 28147-89. Information Processing Systems. Cryptographic Protection. Cryptographic Transformation Algorithm. *(In Russian).*
2. The National Standard of the Russian Federation GOST R 34._-20_. Russian Federal Agency on Technical Regulation and Metrology report, 2015. http://www.tc26.ru/en/standard/draft/ENG_GOST_R_bsh.pdf.
3. AlTawy, R., and Youssef, A. Differential sieving for 2-step matching meet-in-the-middle attack with application to LBlock. In *Lightsec* (2014), T. Eisenbarth and E. Öztürk, Eds., Lecture Notes in Computer Science, Springer. *(to appear).*
4. AlTawy, R., and Youssef, A. M. Preimage attacks on reduced-round Stribog. In *AFRICACRYPT* (2014), D. Pointcheval and D. Vergnaud, Eds., vol. 8469 of *Lecture Notes in Computer Science*, Springer, pp. 109–125.
5. Barreto, P., and Rijmen, V. The Khazad Legacy-Level Block Cipher. In First Open NESSIE Workshop, KU-Leuven, 2000. Submission to NESSIE.
6. Bogdanov, A., and Rechberger, C. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In *SAC* (2011),

A. Biryukov, G. Gong, and D. Stinson, Eds., vol. 6544 of *Lecture Notes in Computer Science*, Springer, pp. 229–240.

7. Canteaut, A., Naya-Plasencia, M., and Vayssire, B. Sieve-in-the-middle: Improved MITM attacks. In *CRYPTO* (2013), R. Canetti and J. Garay, Eds., vol. 8042 of *Lecture Notes in Computer Science*, Springer, pp. 222–240.
8. Daemen, J., and Rijmen, V. AES proposal: Rijndael, 1998.
9. Demirci, H., and Selçuk, A. A meet-in-the-middle attack on 8-round AES. In *FSE* (2008), K. Nyberg, Ed., vol. 5086 of *Lecture Notes in Computer Science*, Springer, pp. 116–126.
10. Demirci, H., Taşkn, I., Çoban, M., and Baysal, A. Improved meet-in-the-middle attacks on AES. In *INDOCRYPT* (2009), B. Roy and N. Sendrier, Eds., vol. 5922 of *Lecture Notes in Computer Science*, Springer, pp. 144–156.
11. Derbez, P., Fouque, P.-A., and Jean, J. Improved key recovery attacks on reduced-round AES in the single-key setting. In *EUROCRYPT* (2013), T. Johansson and P. Nguyen, Eds., vol. 7881 of *Lecture Notes in Computer Science*, Springer, pp. 371–387.
12. Diffie, W., and Hellman, M. Exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer 10*, 6 (1977), 74–84.
13. Dinur, I., Dunkelman, O., and Shamir, A. Improved attacks on full GOST. Cryptology ePrint Archive, Report 2011/558, 2011. http://eprint.iacr.org/.
14. Dunkelman, O., Keller, N., and Shamir, A. Improved single-key attacks on 8-round AES-192 and AES-256. In *ASIACRYPT* (2010), M. Abe, Ed., vol. 6477 of *Lecture Notes in Computer Science*, Springer, pp. 158–176.
15. Hao, Y., Bai, D., and Li, L. A meet-in-the-middle attack on round-reduced mCrypton using the differential enumeration technique. In *Network and System Security* (2014), M. Au, B. Carminati, and C.-C. Kuo, Eds., vol. 8792 of *Lecture Notes in Computer Science*, Springer, pp. 166–183.
16. Hong, D., Koo, B., and Sasaki, Y. Improved preimage attack for 68-step HAS-160. In *ICISC* (2009), D. Lee and S. Hong, Eds., vol. 5984 of *Lecture Notes in Computer Science*, Springer, pp. 332–348.
17. Isobe, T. A single-key attack on the full GOST block cipher. In *FSE* (2011), A. Joux, Ed., vol. 6733 of *Lecture Notes in Computer Science*, Springer, pp. 290–305.
18. Mendel, F., Rechberger, C., Schläffer, M., and Thomsen, S. S. The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In *FSE* (2009), O. Dunkelman, Ed., vol. 5665 of *Lecture Notes in Computer Science*, Springer, pp. 260–276.
19. Poschmann, A., Ling, S., and Wang, H. 256 bit standardized crypto for 650 GE GOST revisited. In *CHES* (2010), S. Mangard and F.-X. Standaert, Eds., vol. 6225 of *Lecture Notes in Computer Science*, Springer, pp. 219–233.
20. Sasaki, Y., Wang, L., Wu, S., and Wu, W. Investigating fundamental security requirements on Whirlpool: Improved preimage and collision attacks. In *ASIACRYPT* (2012), X. Wang and K. Sako, Eds., vol. 7658 of *Lecture Notes in Computer Science*, Springer, pp. 562–579.
21. Shishkin, V., Dygin, D., Lavrikov, I., Marshalko, G., Rudskoy, V., and Trifonov, D. Low-Weight and Hi-End: Draft Russian Encryption Standard. In *CTCrypt* (2014), pp. 183–188.