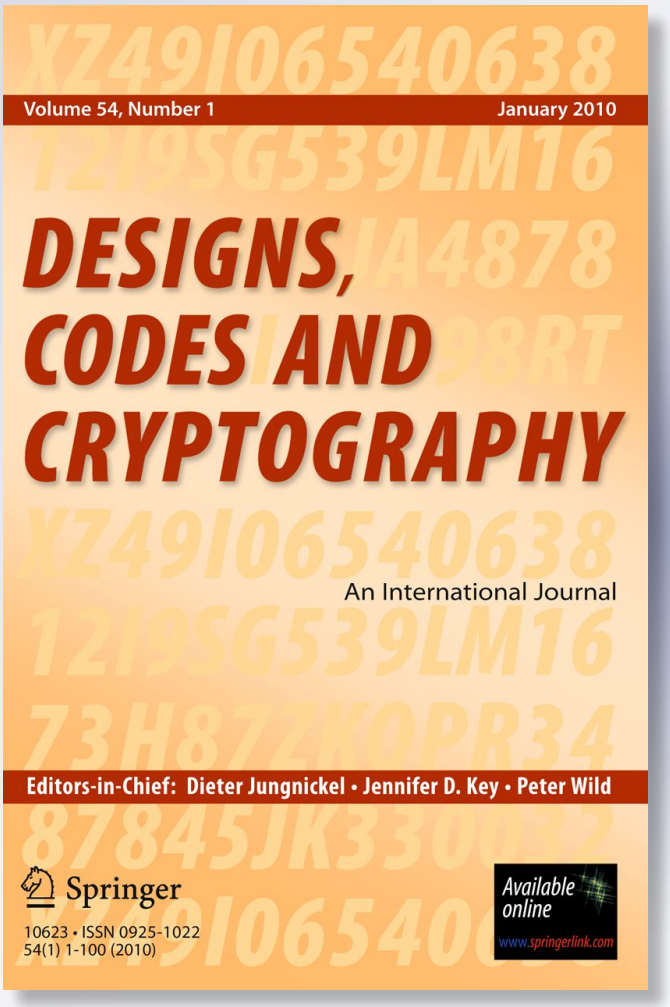
Cryptanalysis of the Stream Cipher LEX

Orr Dunkelman & Nathan Keller

Designs, Codes and Cryptography
An International Journal

ISSN 0925-1022

Des. Codes Cryptogr.
DOI 10.1007/s10623-012-9612-7



Springer

Des. Codes Cryptogr.
DOI 10.1007/s10623-012-9612-7

Cryptanalysis of the Stream Cipher LEX

Orr Dunkelman · Nathan Keller

Received: 29 July 2010 / Revised: 8 January 2012 / Accepted: 13 January 2012

Abstract Biryukov (The Design of a Stream Cipher LEX, Proceedings of Selected Areas in Cryptography, 2006 Springer, pp 67–75, 2007) presented a new methodology of stream cipher design called *leak extraction*. The stream cipher LEX, based on this methodology and on the AES block cipher, was selected to round 3 of the eSTREAM competition. The suggested methodology seemed promising, and LEX, due to its elegance, simplicity, and performance, was expected to be selected to the eSTREAM portfolio. In this article we present a key recovery attack on LEX. The attack requires about 2^{40} bytes of key-stream produced by the same key (possibly under many different IVs), and retrieves the secret key in time of about 2^{100} AES encryptions. Following a preliminary version of our attack, LEX was discarded from the final portfolio of eSTREAM.

Keywords LEX · AES · Stream cipher design

Mathematics Subject Classification (2000) 94A60 · 68P25

Communicated by V. Rijmen.

A preliminary version of the paper, in which the time complexity of the attack is $2^{110.3}$ encryptions, was presented at Asiacrypt 2008 [15]. The improved attack presented in Sect. 5 is novel. Another new result in this article is the improved analysis of the sampling resistance of LEX presented in Sect. 6.

O. Dunkelman (✉)
Computer Science Department, University of Haifa, Haifa 31905, Israel
e-mail: orrd@cs.haifa.ac.il

N. Keller
Department of Mathematics, Bar-Ilan University, Ramat Gan 52900, Israel
e-mail: nathan.keller@weizmann.ac.il

O. Dunkelman · N. Keller
Faculty of Mathematics and Computer Science, Weizmann Institute of Science,
P.O. Box 26, Rehovot 76100, Israel

Published online: 02 March 2012

1 Introduction

The design of stream ciphers, and more generally, pseudo-random number generators (PRNGs), has been the subject of intensive study over the last decades. One of the well-known methods to construct a PRNG is to base it on a keyed pseudo-random permutation. A provably secure construction of this class is given by Goldreich and Levin [20]. An instantiation of this approach (even though an earlier one) is the Blum and Micali [9] construction, based on the hardness of RSA. A more efficiency-oriented construction is the BMGL stream cipher [23], based on the Rijndael block cipher. However, these constructions are relatively slow, and hence are not used in practical applications.

In [4], Biryukov presented a new methodology for constructing PRNGs of this class, called *leak extraction*. In this methodology, the output key stream of the stream cipher is based on parts of the internal state of a block cipher at certain rounds (possibly after passing an additional filtering function). Of course, in such a case, the "leaked" parts of the internal state have to be chosen carefully such that the security of the resulted stream cipher will be comparable to the security of the original block cipher.

As an example of the leak extraction methodology, Biryukov presented in [4] the stream cipher LEX, in which the underlying block cipher is AES. The key stream of LEX is generated by applying AES in the OFB (Output Feedback Block) mode of operation and extracting 32 bits of the intermediate state after the application of each full AES round.

LEX was submitted to the eSTREAM competition (see [5]). Due to its high speed (2.5 times faster than AES), fast key initialization phase (a single AES encryption), and expected security (based on the security of AES), LEX was considered a very promising candidate and selected to the third (and final) round of evaluation.

During the eSTREAM competition, LEX attracted a great deal of attention from cryptanalysts due to its simple structure, but nevertheless, only two prior attacks on the cipher were reported: A slide attack [24] requiring 2^{61} different IVs (each producing 20,000 keystream bytes), and a generic attack [18] requiring $2^{65.7}$ re-synchronizations. Both attacks are applicable only against the original version of LEX presented in [4], but not against the tweaked version submitted to the second round of eSTREAM [6]. In the tweaked version, the number of IVs used with a single key is bounded by 2^{32}, and hence both attacks require too much data and are not applicable to the tweaked version.

In this article we present a new attack on LEX. The attack requires about 2^{40} bytes of key stream produced by the same key, possibly under different IVs. The time complexity of the attack is $2^{100.3}$ AES encryptions.[1] Following a preliminary version of our attack, LEX was discarded from the final portfolio of eSTREAM.[2]

Our attack is composed of four steps:

1. **Identification of a special state:** We focus our attention on pairs of AES states which satisfy a certain difference pattern. While the probability of occurrence of the special pattern is 2^{-64}, the pattern can be observed by a 32-bit condition on the output stream. In order to detect such a state, the adversary considers about $2^{37.8}$ bytes of LEX key stream obtained under the same key (possibly with different IVs),[3] which correspond to $2^{32.5}$

[1] We note that checking whether a key candidate is the right key of LEX requires about one 10-round AES encryption. Thus, using AES encryptions as our measuring unit enables to compare the time complexity of our attack with exhaustive key search whose complexity is about 2^{128} AES encryptions.

[2] After the preliminary version of this article was submitted, Bouillaguet et al. [11] have rediscovered our attack using an automatic tool, thus verifying its complexity.

[3] The exact data complexity of the attack is slightly lower due to some optimizations described later, see Sect. 4.1.

AES encryptions, and thus are expected to contain one pair of AES states that satisfies the special difference pattern. About 2^{32} of the 2^{64} pairs of AES states satisfy the 32-bit condition on the output stream, and the next steps of the attack are repeated for all of them.

2. **Extracting information on the special state:** By using the special difference pattern of the pair of intermediate values, and guessing the values of eight additional state/subkey bytes, the adversary can retrieve the value of 12 additional subkey bytes.
3. **Key ranking:** By using a slightly larger data set which is expected to contain several pairs satisfying the special difference pattern, the adversary can use key ranking techniques to find the most likely values of the 12 subkey bytes suggested in the second step. This allows to filter most of the key candidates while increasing the data complexity by only a small factor.[4]
4. **Guess-and-determine attack on the remaining unknown bytes:** For the remaining key candidates, the adversary can mount a guess-and-determine attack that retrieves the key with time complexity of $2^{100.3}$ encryptions.

The attack uses several observations on the structure of the AES round function and key schedule algorithm.[5] One of them is the following novel observation:

Proposition 1 *Denote the 128-bit subkey used in the r-th round of AES-128 by k_r, and denote the bytes of this subkey by an 4-by-4 array $\{k_r(i, j)\}_{i,j=0}^{3}$. Then for every $0 \leq i \leq 3$ and r,*

$$k_r(i, 1) = k_{r+2}(i, 1) \oplus SB(k_{r+1}(i + 1, 3)) \oplus RCON_{r+2}(i),$$

where SB denotes the SubBytes operation, $RCON_{r+2}$ denotes the round constant used in the generation of the subkey k_{r+2}, and $i + 1$ is replaced by 0 for $i = 3$.

It is possible that the observations on the structure of AES presented in this paper can be used not only in attacks on LEX, but also in attacks on AES itself.[6]

This article is organized as follows: In Sect. 2 we briefly describe the structures of AES and LEX, and present the observations on AES used in our attack. In Sect. 3 we show that a specific difference pattern in the internal state can be partially detected by observing the output stream, and can be used (along with an additional 8-byte guess) to retrieve the actual value of 16 bytes of the internal state (in both encryptions). In Sect. 4 we present a basic key recovery attack based on the above procedure which requires about $2^{110.3}$ trial encryptions. In Sect. 5 we present an enhanced attack using a key ranking technique, which allows to reduce the time complexity to about $2^{100.3}$ trial encryptions. We also discuss the sampling resistance of LEX in Sect. 6. We conclude the article in Sect. 7.

2 Preliminaries

In this section we describe the structures of AES and LEX, and present the observations on AES used in our attack.

[4] This step is performed in an optimized way in order to minimize its memory requirements.

[5] We note that in [4] it was remarked that the relatively simple key schedule of AES may affect the security of LEX, and it was suggested to replace the AES subkeys by 1,280 random bits. Our attack, which relies heavily on some properties of the AES key schedule, would fail if such replacement was performed. However, some of our observations can be used in this case as well.

[6] We note that after the first version of the article was published, these observations were further generalized and used in [16] to improve the best known attack on 8-round AES with 192-bit keys as well as in [10] to attack reduced-round AES variants with a low data complexity.

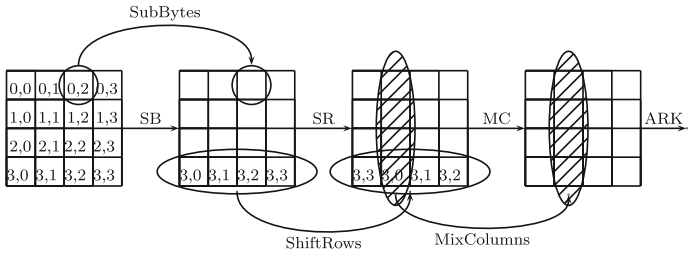


Fig. 1 An AES round

2.1 Description of AES

The advanced encryption standard [12,13] is an SP-network that supports key sizes of 128, 192, and 256 bits. As this article deals with LEX which is based on AES-128, we shall concentrate the description on this variant and refer the reader to [22] for a complete detailed description of AES.

A 128-bit plaintext is treated as a byte matrix of size 4×4, where each byte represents a value in $GF(2^8)$. An AES round applies four operations to the state matrix:

- SubBytes (SB): applying the same 8-bit to 8-bit invertible S-box 16 times in parallel on each byte of the state,
- ShiftRows (SR): cyclic shift of each row (the i'th row is shifted by i bytes to the left),
- MixColumns (MC): multiplication of each column by a constant 4×4 matrix over the field $GF(2^8)$, and
- AddRoundKey (ARK): XORing the state with a 128-bit subkey.

We outline an AES round in Fig. 1. Throughout the paper we allow ourselves the abuse of notation $SB(x)$ to denote the application of the S-box to x (whether it is one S-box when x is an 8-bit value, or 4 times when x is a 32-bit value). In the first round, an additional AddRoundKey operation (using a whitening key) is applied, and in the last round the MixColumns operation is omitted. We note that in LEX these changes to the first and last round are not applied.

AES-128, i.e., AES with 128-bit keys, has 10 rounds. For this variant, 11 subkeys of 128 bits each are derived from the key. The subkey array is denoted by $W[0, \ldots, 43]$, where each word of $W[\cdot]$ consists of 32 bits. The first four words of $W[\cdot]$ are loaded with the user supplied key. The remaining words of $W[\cdot]$ are updated according to the following rule:

- For $i = 4, \ldots, 43$, do
  - If $i \equiv 0 \bmod 4$ then $W[i] = W[i-4] \oplus SB(W[i-1] \lll 8) \oplus RCON[i/4]$,
  - Otherwise $W[i] = W[i-1] \oplus W[i-4]$,

where $RCON[\cdot]$ is an array of predetermined constants, and $\lll$ denotes rotation of the word by 8 bits to the left.

2.2 Description of LEX

For the ease of description, we describe only the tweaked version of LEX submitted to the second round of eSTREAM [6]. The original version of LEX can be found in [4]. We note that our attacks can be easily adapted to the original version as well.

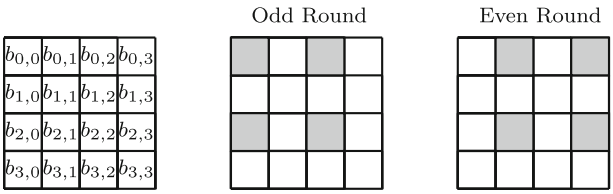


Fig. 2 Odd and even rounds of LEX. The *gray bytes* are the output bytes

In the initialization step, the publicly known IV is encrypted by AES[7] under the secret key K to obtain $S = AES_K(IV)$. Then, S is repeatedly encrypted in the OFB mode of operation under K, where during the execution of each encryption, 32 bits of the internal state are leaked in each round. These state bits compose the key stream of LEX. The state bytes used in the key stream are shown in Fig. 2. After 500 encryptions, another IV is chosen, and the process is repeated. After 2^{32} different IVs, the secret key is replaced. [8]

Encryption in LEX is done by XORing the generated key stream with the plaintext. Decryption is done by XORing the ciphertext with the same key stream.

2.3 Notations used in the article

As in [4], the bytes of each internal state during AES encryption, as well as the bytes of the subkeys, are denoted by a 4-by-4 array $\{b_{i,j}\}_{i,j=0}^{3}$, where $b_{i,j}$ is the j-th byte in the i-th row. For example, the output bytes in the even rounds are $b_{0,1}, b_{0,3}, b_{2,1}, b_{2,3}$.

2.4 Observations on AES used in our attack

Throughout the article we use several observations concerning AES.

Observation 1 *For every non-zero input difference to the SubBytes operation, there are 126 possible output differences with probability* 2^{-7} *each (i.e., only a single input pair with the given difference leads to the specified output difference), and a single output difference with probability* 2^{-6}.

As a result, for a randomly chosen pair of input/output differences of the SubBytes operation, with probability 126/256 there is exactly one unordered pair of values satisfying these differences. With probability 1/256 there are two such pairs, and with probability 129/256, there are no such pairs.

We note that while each ordered pair of input/output differences suggests one pair of actual values on average, it actually never suggests *exactly* one pair. In about half of the cases, two (or more) *ordered* pairs are suggested, and in the rest of the cases, no pairs are suggested. In the cases where two (or more) pairs are suggested, the analysis has to be repeated for each of the pairs. On the other hand, if no pairs are suggested, then the input/output differences pair is discarded as a wrong pair and the analysis is not performed at all. Hence, when factoring both events, it is reasonable to assume that each input/output differences pair suggests one pair of actual values.

[7] Actually, LEX uses a tweaked version of AES where the AddRoundKey before the first round is omitted, and the MixColumns operation of the last round is present. We allow ourselves the slight abuse of notations, for sake of clarity.

[8] We note that in the original version of LEX, the number of different IVs used with a single key was not bounded. Following the slide attack presented in [24], the number of IVs used with each key was restricted. This restriction also prevents the attack suggested later in [18] which requires $2^{65.7}$ re-synchronizations.

Our attack uses this observation in situations where the adversary knows the input and output differences to some SubBytes operation. In such cases, using the observation she can deduce the actual values of the input and the output (for both encryptions). This can be done efficiently by preparing the difference distribution table of the SubBytes operation, along with the actual values of the input pairs satisfying each input/output difference relation (rather than only the number of such pairs). In the actual attack, given the input and output differences of the SubBytes operation, the adversary can retrieve the corresponding actual values using a simple table lookup.

Observation 2 *Since the MixColumns operation is an MDS matrix, if the values (or the differences) in any four out of its eight input/output bytes are known, then the values (or the differences, respectively) in the other four bytes are uniquely determined, and can be computed efficiently.*

The following two observations are concerned with the key schedule of AES. While the first of them is known (see [19]), it appears that the second is new.

Observation 3 *For each* $0 \leq i \leq 3$*, the subkeys of AES satisfy the relations:*

$$k_{r+2}(i,0) \oplus k_{r+2}(i,2) = k_r(i,2).$$
$$k_{r+2}(i,1) \oplus k_{r+2}(i,3) = k_r(i,3).$$

Proof Recall that by the key schedule, for all $0 \leq i \leq 3$ and for all $0 \leq j \leq 2$, we have $k_{r+2}(i,j) \oplus k_{r+2}(i,j+1) = k_{r+1}(i,j+1)$. Hence,

$$\begin{aligned} k_{r+2}(i,0) \oplus k_{r+2}(i,2) &= k_{r+2}(i,0) \oplus k_{r+2}(i,1) \oplus k_{r+2}(i,1) \oplus k_{r+2}(i,2) \\ &= \Big(k_{r+2}(i,0) \oplus k_{r+2}(i,1)\Big) \oplus \Big(k_{r+2}(i,1) \oplus k_{r+2}(i,2)\Big) \\ &= k_{r+1}(i,1) \oplus k_{r+1}(i,2) = k_r(i,2), \end{aligned}$$

and the second claim follows similarly. □

Observation 4 *For each* $0 \leq i \leq 3$*, the subkeys of AES satisfy the relation:*

$$k_{r+2}(i,1) \oplus SB(k_{r+1}((i+1) \bmod 4, 3)) \oplus RCON_{r+2}(i) = k_r(i,1),$$

Proof In addition to the relation used in the proof of the previous observation, we use the relation

$$k_{r+2}(i,0) = k_{r+1}(i,0) \oplus SB(k_{r+1}((i+1) \bmod 4, 3)) \oplus RCON_{r+2}(i).$$

Thus,

$$\begin{aligned} &k_{r+2}(i,1) \oplus SB(k_{r+1}((i+1) \bmod 4, 3)) \oplus RCON_{r+2}(i) \\ &\quad = \Big(k_{r+2}(i,1) \oplus k_{r+2}(i,0)\Big) \\ &\qquad \oplus \Big(k_{r+2}(i,0) \oplus SB(k_{r+1}((i+1) \bmod 4, 3)) \oplus RCON_{r+2}(i)\Big) \\ &\quad = k_{r+1}(i,1) \oplus k_{r+1}(i,0) \\ &\quad = k_r(i,1). \end{aligned}$$

□

These two observations allow the adversary to use the knowledge of bytes of k_{r+2} (and the last column of k_{r+1}) to get the knowledge of bytes in k_r, while "skipping" (some of) the values of k_{r+1}.

3 Chessboard difference pattern in LEX

In this section we describe the special difference pattern used in our attacks. The probability of occurrence of the pattern is 2^{-64}, and hence, it is expected to be found within $2^{32.5}$ AES states, which in turn, are included in $2^{37.8}$ bytes of the LEX key stream produced by the same key. We note that by the specification of LEX, after every 500 AES encryptions, a new IV is chosen, and thus, at least $2^{23.5}$ different IVs are used in the key stream we examine. However, we emphasize that this has no effect on our attacks.

Our attacks are applicable when the special difference pattern starts either in odd rounds or in even rounds. For sake of simplicity of the description, we present the results assuming the difference pattern occurs in the odd rounds, and give in Appendix A the modified attack applicable when the difference pattern occurs in even rounds.

3.1 Detecting the difference pattern

Consider two AES encryptions under the same secret key, K. The special difference pattern corresponds to the following event: The difference between the intermediate values at the end of the $(r+1)$-th round is non-zero only in bytes $b_{0,0}, b_{0,2}, b_{1,1}, b_{1,3}, b_{2,0}, b_{2,2}, b_{3,1}$, and $b_{3,3}$. The probability of this event is 2^{-64}. The pattern, along with the evolution of the differences in rounds $r, r+1, r+2$, and $r+3$, is presented in Fig. 3.

The difference pattern can be partially observed by a 32-bit condition on the output key stream: If the pattern holds, then all the four output bytes in round $r+2$ (bytes $b_{0,1}, b_{0,3}, b_{2,1}, b_{2,3}$) have zero difference.

Therefore, it is expected that amongst 2^{64} pairs of AES states encrypted under the same key, one of the pairs satisfies the difference pattern, and about 2^{32} pairs satisfy the filtering condition. Thus, the following attack steps have to be repeated 2^{32} times on average (once for each candidate pair).

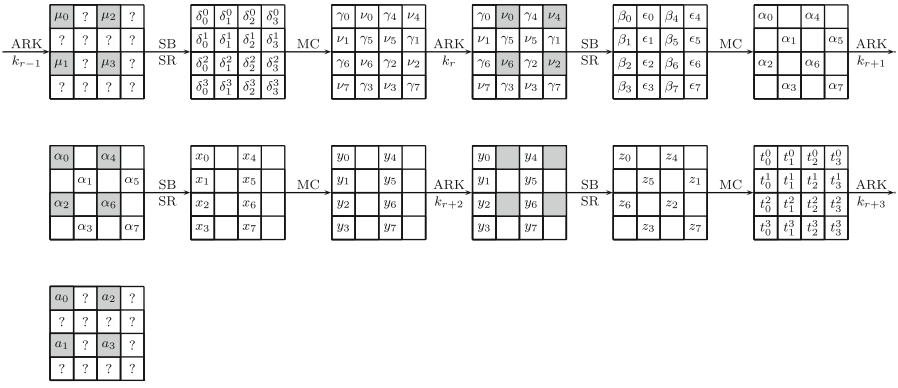


Fig. 3 The special difference pattern (for odd rounds). Gray cells denote bytes whose value is known from the output key stream. *Empty cells* denote zero difference

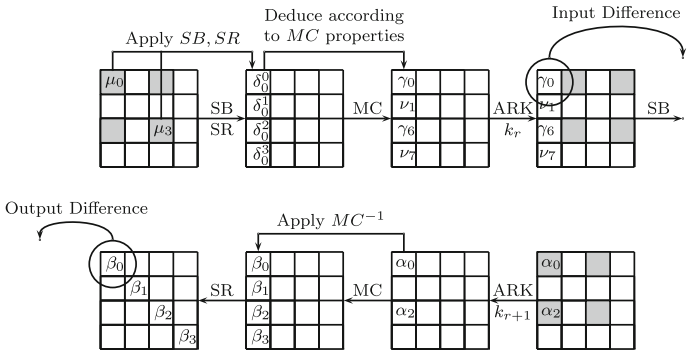


Fig. 4 Deducing the actual value of $b_{0,0}$ in the end of round r

It is easy to detect these pairs by either using hash tables or even by storing the data and comparing all pairs. The time complexity of this step is negligible ($2^{36.3}$ memory accesses using a hash table,[9] or 2^{64} memory accesses when comparing all pairs). This also determines the memory complexity, which is the amount of memory needed to store the data, i.e., $2^{36.3}$ bytes.

We note that if the special difference pattern is satisfied, then by the linearity of the MixColumns operation, there are only 255^2 possible values for the difference in each of the columns before the MixColumns operation of round $r+1$ (denoted by β_i and ϵ_j in Fig. 3), and in each of the columns after the MixColumns operation of round $r+2$ (denoted by t_i^j in Fig. 3). This property is used in the second step of the attack to retrieve the actual values of several state bytes.

3.2 Using the difference pattern to retrieve actual values of 16 intermediate state bytes

In this section we show how the adversary can use the special difference pattern, along with a guess of the difference in eight additional bytes, in order to recover the actual values of 16 intermediate state bytes in both encryptions. We show in detail how the adversary can retrieve the actual value of byte $b_{0,0}$ of the state at the end of round r using the guess of the two difference bytes denoted by ν_1 and ν_7.[10] The derivation of 15 additional bytes, which is performed in a similar way, is described briefly below.

The derivation of the actual value of byte $b_{0,0}$ of the state at the end of round r is composed of several steps (described also in Fig. 4):

1. The adversary finds the difference in Column 0 before the MixColumns operation of round $r+1$, i.e., $(\beta_0, \beta_1, \beta_2, \beta_3) = MC^{-1}(\alpha_0, 0, \alpha_2, 0)$. This is possible as the differences α_0 and α_2 are known from the output stream.
2. The adversary guesses the differences ν_1, ν_7 and applies the following steps for each such guess.
3. Given the differences ν_1 and ν_7, there are 255^2 possible differences after the MixColumns of round r in the leftmost column. Using the output bytes $b_{0,0}, b_{2,2}$ of round $r-1$, the adversary knows the difference in two bytes of the same column before the

[9] One can store the key stream in a hash table indexed by the value of the key stream in the FOUR bytes produced in round $r+2$. Once a collision in the hash table is found, a candidate pair of streams is identified. To cover all possible starting positions, 8 such tables are used.

[10] Note that while retrieving one byte by guessing two is inefficient, since the guessed bytes can be used in retrieving other bytes as well, the overall gain is positive.

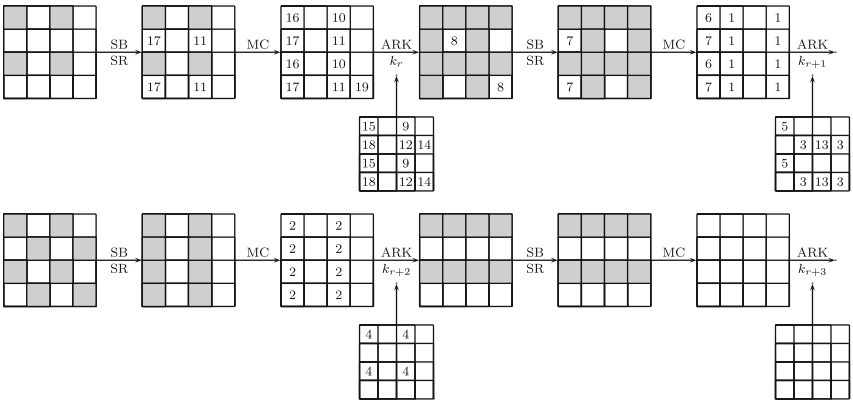


Fig. 5 The first phase of the guess-and-determine attack on LEX (for odd rounds). *Gray cells* are known bytes. Bytes (cells) marked with i, are bytes which are computed in step i. Transition 9 is based on Observation 3

MixColumns operation. Hence, using Observation 2 (the linearity of the MixColumns operation), the adversary retrieves the difference in the whole column, both before and after the MixColumns operation, including the difference γ_0.

4. At this point, the adversary knows the input difference (γ_0) and the output difference (β_0) to the SubBytes operation in byte $b_{0,0}$ of round $r+1$. Hence, using Observation 1 (the property of the SubBytes operation), the adversary finds the actual values of this byte using a single table look-up. In particular, the adversary retrieves the actual value of byte $b_{0,0}$ at the end of round r.

The additional 15 bytes are retrieved in the following way:

1. The value of byte $b_{2,2}$ at the end of round r is obtained in the same way using bytes $b_{0,2}, b_{2,0}$ of the output of round $r-1$ (instead of bytes $b_{0,0}, b_{2,2}$) and examining the third column (instead of the first one).
2. The value of bytes $b_{0,2}$ and $b_{2,0}$ at the end of round r is found by examining α_4, α_6 (instead of α_0, α_2), guessing the differences ν_3, ν_5 (instead of ν_1, ν_7), and repeating the process used in the derivation of bytes $b_{0,0}, b_{2,2}$.
3. In a similar way, by guessing the differences x_1, x_3, x_5, x_7 and using the output bytes of round $r+3$, the adversary can retrieve the actual values of bytes $b_{0,0}, b_{0,2}, b_{2,0}$ and $b_{2,2}$ in the output of round $r+2$.
4. Using the output of round r and Observation 2, the adversary can obtain the differences $\alpha_1, \alpha_3, \alpha_5, \alpha_7$. Then, she can use the guessed differences x_1, x_3, x_5, x_7 and Observation 1 to obtain the actual values of bytes $b_{1,1}, b_{1,3}, b_{3,1}$ and $b_{3,3}$ at the end of round $r+1$.
5. Finally, using again the output of round r and Observation 2, the adversary can obtain the differences $\epsilon_1, \epsilon_3, \epsilon_5, \epsilon_7$. Then, using the guessed differences $\nu_1, \nu_3, \nu_5, \nu_7$ and Observation 1, the adversary can obtain the actual values of bytes $b_{1,0}, b_{1,2}, b_{3,0}$, and $b_{3,2}$ at the end of round r.

The bytes whose actual values are known to the adversary at this stage (which are 8 bytes at round r, 4 in round $r+1$, and 4 in round $r+2$) are presented in Fig. 5 marked in gray.

We note that this process takes 2^{64} time for each candidate states (or streams), as we guess the eight values $\nu_1, \nu_3, \nu_5, \nu_7, \epsilon_1, \epsilon_3, \epsilon_5$, and ϵ_7 for each candidate. As we start with 2^{32} candidate pairs (after identifying the candidates pairs which agree on 32-bit of the output stream), this step takes about 2^{96} operations.

4 The basic attack on LEX

In this section we present our basic attack. The attack takes $2^{36.3}$ keystream bytes produced by the same key (under different IVs) and finds the key in time equivalent to $2^{110.3}$ AES encryptions. The adversary finds in the keystream one pair of states that satisfies the difference described in Sect. 3. Using a guess-and-determine procedure, the attack deduces the full key given the 16 bytes obtained by the process described in Sect. 3, and based on Observations 2 and 3 presented in Sect. 2 (properties of the MixColumns operation and of the key schedule algorithm of AES-128).

The first step of the attack is to identify a pair of states whose difference is indeed the chessboard difference pattern depicted in Fig. 3. As the analysis of Sect. 4.1 shows, the $2^{36.3}$ keystream bytes can be used to generate 2^{64} pairs of states, and it is expected that one of them satisfies the chessboard difference pattern. Luckily, we do not need to try all 2^{64} possible pairs of states, as a pair of states that satisfies the required difference, necessarily has a zero difference in 32 keystream bits, which means that we need to consider only 2^{32} pairs of states.

For each such candidate pair of states, we try all the 8 bytes needed for the procedure described in Sect. 3. Once these bytes are guessed, and the adversary computes all the bytes whose value can be determined, an additional guess-and-determine phase is executed. This deduction is composed of two phases. The first phase is presented in Fig. 5, where gray bytes denote bytes whose values is known before the phase starts (either from the output or from the process described in Sect. 3) and a cell containing a number i, is deduced as the ith step of this phase. We note that no additional key material is guessed in this phase. There are several types of deduction:

- Application of the AES operations on known values (steps 1, 2, 6, 8, 10, 16, 19).
- Deducing subkey bytes by knowing the input/output of the AddRoundKey operation (steps 3, 4, 12, 18).
- Deducing subkey material using 1-round key relations (steps 5, 13, 14, 15).
- Completing the values of a column given 4 input/output bytes of the MixColumns operation (steps 7, 11, 17).
- Deducing subkey material using Observation 3 (step 9).

This phase of the attack is applied to each candidate pair, where each series of analysis steps are very efficient (most operations are XORs or multiplication over $GF(2^8)$ by a fixed matrix).

At the beginning of the second phase, presented in Fig. 6, the adversary guesses the value of two additional subkey bytes, which are marked by black. As before, we use gray bytes to mark bytes which are known at the beginning of this phase and a cell with the number i, refers to a byte which is computed in the ith step of the deduction sequence.

As before, there are several types of deduction steps:

- Application of the AES operations on known values (steps 2, 4, 5, 6, 8, 12, 13, 15, 17).
- Deducing subkey bytes by knowing the input/output of the AddRoundKey operation (steps 9, 18).
- Deducing subkey material using 1-round key relations (steps 1, 10, 11, 14).
- Completing the values of a column given 4 input/output bytes of the MixColumns operation (steps 3, 7, 16).

As these steps are done for each guess of the two bytes, the time complexity of this step is 2^{16} operations for each candidate pair (or a total of 2^{112} operations), at the end of which, the adversary obtains a complete internal state of LEX. At this point, it is possible to "run" LEX

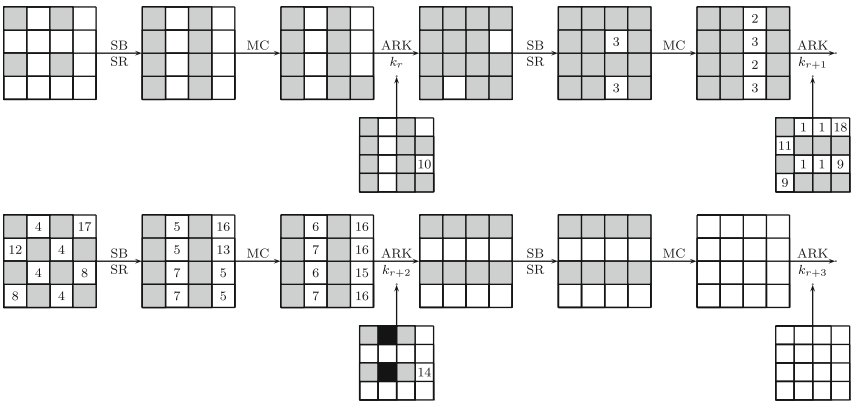


Fig. 6 The second phase of the guess-and-determine attack on LEX (for odd rounds). *Gray cells* are known bytes. *Black cells* are the two bytes guessed in this phase. Bytes (cells) marked with i, are bytes which are computed in step i

on the given internal state, and see whether the output produced by the retrieved state and the output stream agrees. Once a disagreement is found, the combination of candidate pair and guesses is discarded.

Summarizing the attack, the adversary guesses 10 bytes of information (8 bytes of differences guessed in the process of Sect. 3 of the attack, and 2 subkey bytes guessed in this step of the attack), and retrieves the full secret key. Therefore, the time complexity of this procedure is 2^{80} guesses for each candidate pair of streams, which is repeated for the 2^{32} candidate streams, with a total time complexity of $2^{32} \cdot 2^{80}$ operations. Since the time required for each application of retrieval phase is less than a 3-round AES encryption, the total time complexity of the attack is less than $2^{112} \cdot 2^{-1.7} = 2^{110.3}$ 10-round AES encryptions.

Since the most time-consuming step of the attack is a guess-and-determine procedure, it is very easy to parallelize the attack, and obtain a speed up equivalent to the number of used CPUs. Also, the simplicity of the attack enables the use of other implementation techniques, such as SIMD instructions and bitslicing.

4.1 Data complexity of the basic attack

The attack is based on examining special difference patterns. Since the probability of occurrence of a special pattern is 2^{-64}, it is expected that $2^{32.5}$ states of encryptions under the same key yield a single pair of states satisfying the special pattern.

However, we note that the attack can be applied to several values of the starting round of the difference pattern. The attack presented above is applicable if r is equal to 1, 3, 5, or 7, and a slightly modified version of the attack (presented in Appendix A) is applicable if r is equal to 0, 2, 4, or 6.[11] Hence, $2^{64}/8 = 2^{61}$ pairs of encryptions are sufficient to supply a pair satisfying one of the eight possible difference patterns. These 2^{61} pairs can be obtained from 2^{31} full AES encryptions, or equivalently, $2^{36.3}$ bytes of output key stream generated by the

[11] We note that while the attack considers five rounds of encryption (rounds $r-1$ to $r+3$), it is not necessary that all the five rounds are contained in a single AES encryption. For example, if $r = 7$ then round $r + 3$ considered in our attack is actually round 0 of the next encryption. The only part of the attack which requires the rounds to be consecutive rounds of the same encryption is the key schedule considerations. However, in these considerations only three rounds (rounds r to $r + 2$) are examined.

same key. According to LEX' specification, the IV is changed every 500 AES encryptions, and the data examined in the attack is generated under at least 2^{22} different IVs. However, the attack procedure does not assume anything on the IV, which means that the use of different IVs does not affect the analysis.

5 An improved attack

In this section we present an improved variant of the attack that allows to reduce the time complexity to $2^{100.3}$ encryptions, at the expense of enlarging the data complexity by a factor of 16. We start with the general idea of the attack, and then present some technical adjustments required in order to obtain optimal time and memory complexities.

5.1 The general outline of the attack

The main observation behind the improved attack is noting that after the first phase of the basic attack, the adversary already obtains the equivalent of 12 key bytes. (In Fig. 5, these are ten bytes of the subkey k_r and bytes $b_{1,1}$, $b_{3,1}$ of the subkey k_{r+1}. These 12 bytes are independent since by guessing four more bytes (bytes $b_{0,2}$, $b_{1,0}$, $b_{2,2}$, $b_{3,0}$ of k_{r+1}), one can immediately retrieve the full subkey k_r by the key schedule algorithm. The other bytes obtained at that stage depend on these 12 bytes).

Assume that the initial number of pairs of states examined in the attack is enlarged by a factor of C (i.e., $C \cdot 2^{64}$ pairs of states), so that it is expected that the data contains C pairs satisfying the special difference pattern. In this case, the first two phases of the attack are performed $C \cdot 2^{32} \cdot 2^{64}$ times (once for each pair remaining after the initial filtering and for each guess of the additional eight bytes of difference). The adversary can perform a key ranking procedure considering the number of suggestions of each value of the 12 key bytes described above. Since there are 2^{96} possible values, each of them is suggested C times on average.

At the same time, we claim that the correct key guess is expected to be suggested $2C$ times. Indeed, the correct value is suggested by each of the C pairs satisfying the special difference pattern (along with the correct guess of the eight additional byte differences). In addition, by a reasonable randomness assumption, for each pair that does not satisfy the special pattern, and for any guess of the difference in the additional eight bytes, the probability that the correct key value is suggested is 2^{-96}. Since there are approximately $C \cdot 2^{96}$ pairs of this form in the data, the correct value is expected to be suggested C more times.[12]

The number of times a specific key is suggested follows a Poisson distribution. Hence, the wrong guesses are distributed according to $Poi(C)$, and those of the correct guess have distribution $Poi(2C)$. Thus, for $C = 32$, the probability that the correct key value is amongst the 2^{85} highest values is at least 92.8%. At this stage, the adversary can perform the second phase of the guess and determine phase of the attack only for the remaining 2^{85} values. We note that this phase does not require any additional memory.

[12] This randomness assumption is founded on the reasonable assumption that a wrong pair (i.e., a pair that does not satisfy the special pattern), along with an incorrect guess of the eight additional difference bytes, offers a random value as the attack procedure is very far from approximating the "real" encryption procedure. However, since such assumptions are very delicate, it is very important to check their validity in each concrete case of study. Unfortunately, in our case, it is impossible to verify this assumption due to the high time complexity of such check.

Hence, the attack requires 2^{69} pairs of AES states, which are expected to contain 32 pairs of states with the required difference. We note that unlike the basic attack, the adversary cannot use different starting rounds simultaneously, and thus the data complexity is 2^{35} AES encryptions, or $2^{40.3}$ bytes of key-stream. The time complexity of the improved attack is $2^{37} \cdot 2^{64} = 2^{101}$ deductions for the first phase of analysis, and the same amount of work for the second phase.[13] Since the time complexity of each deduction is smaller than 3 AES rounds, the total time complexity is at most $2^{100.3}$ 10-round AES encryptions.

5.2 Reducing the memory requirements

While the attack procedure described above reduces the time complexity of the attack considerably, it requires a large amount of memory. Indeed, the key ranking procedure requires an array of almost 2^{96} bytes to store the number of times each 12-byte key value is suggested.

In order to reduce the memory requirements, we suggest to alter the second phase of the attack, such that four of the eight additional guessed bytes are *key bytes* rather than state difference bytes. In the discussion below we refer to Fig. 3. We allow ourselves a slight abuse of notation, and use the notations of the differences in the cells to denote also the actual values of the respective cells in both states.

Claim Instead of guessing the eight additional difference bytes, it is sufficient to guess the actual values in both states of the bytes denoted by $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ (either before or after the addition of k_{r+1}), and bytes $b_{1,1}, b_{1,3}, b_{3,1}, b_{3,3}$ of the subkey k_{r+1}. Moreover, guessing four bytes is sufficient to get the actual values of the bytes denoted by $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ in both states.

Proof Recall that the eight additional guessed difference bytes are those denoted by $\nu_1, \nu_3, \nu_5, \nu_7$ and x_1, x_3, x_5, x_7 in Fig. 3. Assume that the adversary knows the actual values in both states of the bytes denoted by $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ before the addition of k_{r+1}, and bytes $b_{1,1}, b_{1,3}, b_{3,1}, b_{3,3}$ of k_{r+1}. Clearly, this allows to retrieve also the actual values of the bytes denoted by $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ *after* the addition of k_{r+1}. The differences $\nu_1, \nu_3, \nu_5, \nu_7$ and x_1, x_3, x_5, x_7 can be then obtained by the following steps:

1. The knowledge of the actual values in the bytes marked by $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ before the addition of k_{r+1}, along with the actual values of the cells $\epsilon_0, \epsilon_2, \epsilon_4, \epsilon_6$ (which are known from the key-stream) allows to apply Observation 2 to the MixColumns operation of round $r+1$, and get the actual values in the cells denoted by $\epsilon_1, \epsilon_3, \epsilon_5, \epsilon_7$.
2. The actual values of the cells denoted by $\epsilon_1, \epsilon_3, \epsilon_5, \epsilon_7$ yield the actual values of the cells denoted by $\nu_1, \nu_3, \nu_5, \nu_7$, by performing $ShiftRows^{-1}$ and $SubBytes^{-1}$ operations.
3. The actual values in the cells denoted by $\nu_1, \nu_3, \nu_5, \nu_7$ yield the differences $\nu_1, \nu_3, \nu_5, \nu_7$.
4. The actual values in the cells denoted by $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ after the addition of k_{r+1} yield immediately the actual values in the cells denoted by x_1, x_3, x_5, x_7 by performing the SubBytes and MixColumns operations of round $r+2$.
5. The actual values in the cells denoted by x_1, x_3, x_5, x_7 yield the differences x_1, x_3, x_5, x_7.

In order to show that guessing four bytes is sufficient to get the actual values of the bytes denoted by $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ in both encryptions, we observe that the *difference* in these bytes is known to the adversary. Indeed, note that the differences $\epsilon_0, \epsilon_2, \epsilon_4, \epsilon_6$ are known from

[13] We note that there is no need to use trial encryption in order to check the subkeys suggested in the last phase of the attack. Instead, the adversary can perform key ranking on the two last retrieved subkey bytes, using the fact that the correct value is suggested by all 32 pairs satisfying the special difference pattern.

the key-stream, and the differences in bytes $b_{0,1}, b_{0,3}, b_{2,1}, b_{2,3}$ after the MixColumns of round $r+1$ are known to be zero. Thus, the adversary can apply Observation 2 to the MixColumns operation of round $r+1$ (with respect to the differences) and get the differences $\alpha_1, \alpha_3, \alpha_5, \alpha_7$, as desired. Thus, it is sufficient to guess the actual values of bytes $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ in one of the encryptions, and the values in the other encryption can be retrieved using the known difference. □

Using the suggested change to phase 2 of the attack, the memory requirement can be reduced to less than 2^{64} bytes of memory. In order to achieve this improvement, the attack is performed for each guess of bytes $b_{1,1}, b_{1,3}, b_{3,1}, b_{3,3}$ of the subkey k_{r+1} independently. In each single attack, the adversary considers 2^{35} states, out of which 2^{69} pairs are passed to the second phase of the analysis. This phase is performed for each of the 2^{32} possible actual values in bytes $\alpha_1, \alpha_3, \alpha_5, \alpha_7$, and the key ranking is performed on the eight deduced key bytes.

We note that as the attack is the same as in the previous section, the analysis of its time complexity or success rate remains. Hence, setting $C = 32$, with 92.8% probability, the right value of the key will ranked in the top 2^{85} keys passed for further analysis in phase 2.

The data complexity of the attack is $2^{40.3}$ bytes of output stream, that can be produced by about 2^{30} different IVs with a single fixed key. The time complexity of the attack is $2^{100.3}$ trial encryptions, and the memory complexity is 2^{64} bytes.

6 Sampling resistance of LEX

One of the main advantages of LEX, according to the designers (see [4], Sect. 1), is the small size of its internal state allowing for a very fast key initialization (a single AES encryption). It is stated that the size of the internal state (256 bits) is the minimal size assuring resistance to time-memory-data tradeoff attacks.

Time-memory-data tradeoff (TMDTO) attacks [2,7,8,14,21] are considered a serious security threat to stream ciphers, and resistance to this class of attacks is a mandatory in the design of stream ciphers (see, for example, [17]). A cipher with an n-bit key is considered (certificationally) secure against TMDTO attacks if any TMDTO attack on the cipher has either data, memory, or time complexity of at least 2^n.

In order to ensure security against conventional TMDTO attacks trying to invert the function (State $\rightarrow$ Key Stream), it is sufficient that the size of the internal state is at least twice the size of the key [8]. LEX satisfies this criterion (the key size is 128 bits and the size of the internal state is 256 bits). As a result, as claimed by the designers (see [4], Sects. 3.2 and 5), the cipher is secure with respect to TMDTO attacks.

However, as observed in [1], having the size of the internal state exactly twice larger than the key length is not sufficient if the cipher has a low *sampling resistance*. Roughly speaking, a cipher has a sampling resistance of 2^{-t}, if it is possible to enumerate all internal states which lead to some t-bit output string efficiently. In other words, if it is possible to find a (possibly special) string of t bits, whose "predecessor" states are easily computed, then the cipher has sampling resistance of at most 2^{-t}.

It is clear that LEX has sampling resistance of at most 2^{-32}, as out of the 256 bits of internal state, 32 bits are exposed every round. However, it turns out that LEX's sampling resistance is actually at most 2^{-64}. For example, assume that we would like to enumerate the states for which the output in two consecutive rounds is 0 (i.e., 8 bytes are set to 0). As the 256-bit state, we consider the 128-bit AES state before the key addition of the second round

and the 128-bit key. The condition that the output of the second round is zero is translated to equality between four bytes of the state to the four corresponding bytes of the key. The condition that the output of the first round is zero is translated to two 16-bit conditions on two columns of the state, which can be easily computed by partially decrypting through the second round. Thus, the 2^{192} states satisfying these conditions can be easily and efficiently enumerated. Moreover, it is trivial to adopt this algorithm to any desired 64-bit output string.

As a result, using the attack algorithm presented in [8], it is possible to mount a TMDTO attack on LEX with time, data, and memory complexities of $(2^{128})^{4/5} = 2^{102.4}$. Hence, LEX provides only 102-bit security with respect to TMDTO attacks.[14]

7 Summary and conclusions

In this article we presented a new attack on the LEX stream cipher. We showed that there are special difference patterns that can be easily observed in the output key stream, and that these patterns can be used to mount a key recovery attack.

We offered two variants of the attack. The first takes $2^{36.3}$ bytes of stream, $2^{36.3}$ bytes of memory, and $2^{110.3}$ time. The second takes $2^{40.3}$ bytes of stream, 2^{64} bytes of memory, and $2^{100.3}$ time.

Our results show that for constructions based on the Goldreich-Levin approach (i.e., PRNGs based on pseudo-random permutations), the pseudo-randomness of the underlying permutation is crucial to the security of the resulting stream cipher. In particular, a small number of rounds of a (possibly strong) block cipher cannot be considered random in this sense, at least when a non-negligible part of the internal state is extracted.

Acknowledgments We would like to express our gratitude to Adi Shamir for his extensive comments and the fruitful discussions. We would also like to thank the anonymous referees for their valuable comments and suggestions. Part of this research was carried out while the first author was with École Normale Supérieure, and was partially supported by the France Telecome Chaire and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). The second author was partially supported by the Koshland center for basic research. Part of this research was carried out while the second author was with the Hebrew University, and was partially supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

Appendix A: Special difference pattern starting with an even round

In this section we present the modified version of the attack that can be applied if the special difference pattern occurs in the even rounds. The first two steps of the attack (observing the difference pattern and deducing the actual values of 16 additional bytes of the state) are similar to the first two steps of the attack presented in Sect. 3. The known byte values after these steps are presented in Fig. 7, marked in gray. The third step of the attack is slightly different due to the asymmetry of the key schedule, and Observation 4 is used in this step along with Observations 2 and 3. The two phases of this step are presented in Figs. 7

[14] We note that in TMDTO attacks exploiting low sampling resistance, the tradeoff curve is $N^2 = TM^2D^2$ like in ordinary TMDTO attacks, and the low sampling resistance is used only to increase the value of D for which the curve can be applied. Thus, the optimal possible attack of this kind is obtained for $D = M = T$, which accounts to $n^{4/5}$ when n is the size of the key and $2n$ is the size of the state. Since in the case of LEX this value can already be obtained, this implies that even if the sampling resistance of LEX was lower, this would not improve the complexity of the best possible TMDTO attack on the cipher.

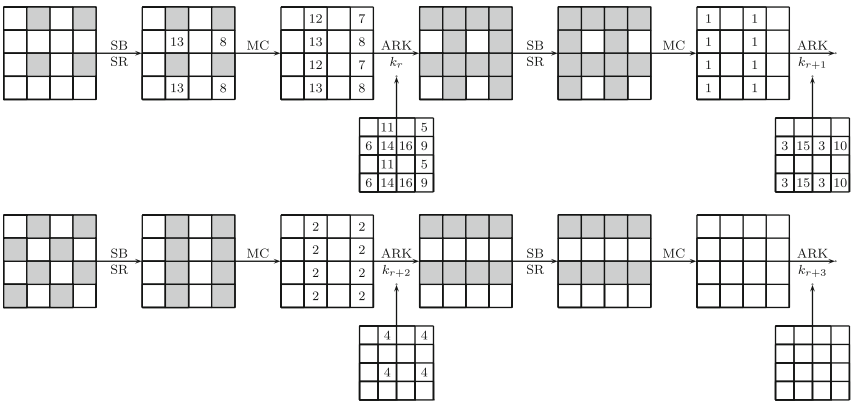


Fig. 7 The first phase of the guess-and-determine attack on LEX (in even rounds). *Gray cells* are known bytes. Bytes (cells) marked with i, are bytes which are computed in step i. Transition 5 based on Observation 3, and transition 11 is based on Observation 4

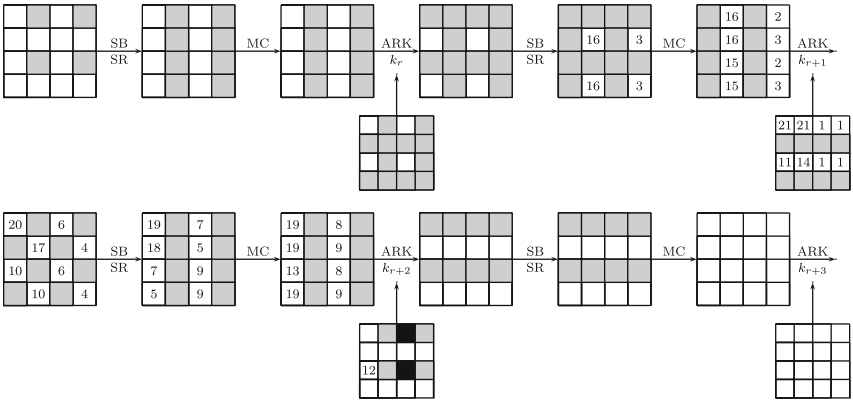


Fig. 8 The second phase of the guess-and-determine attack on LEX (in even rounds). *Gray cells* are known bytes. *Black cells* are the two bytes guessed in this phase. Bytes (cells) marked with i, are bytes which are computed in step i

and 8. The overall time complexity of the attack is $2^{110.3}$ encryptions, like in the case of a difference pattern in the odd rounds (in the basic attack). One can also apply the attack of Sect. 5 with a slightly increased data complexity in exchange for a significantly lower time complexity.

References

1. Babbage S.H., Dodd M.: Specification of the stream cipher mickey 2.0, submitted to eSTREAM, (2006). Available on-line at: http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf.
2. Babbage S.H.: Improved "exhaustive search" attacks on stream ciphers, IEE European Convention on Security and Detection, IEE Conference publication 408, pp. 161–165 (1995).
3. Biham E., Shamir A.: Differential cryptanalysis of the data encryption standard, Springer, London (1993).
4. Biryukov A.: The design of a stream cipher LEX. Proceedings of Selected Areas in Cryptography 2006, Lecture Notes in Computer Science 4356, pp. 67–75, Springer, Berlin (2007).

5. Biryukov A.: A new 128-bit key stream cipher LEX, ECRYPT stream cipher project report 2005/013. Available on-line at http://www.ecrypt.eu.org/stream.
6. Biryukov A.: The Tweak for LEX-128, LEX-192, LEX-256, ECRYPT stream cipher project report 2006/037. Available on-line at http://www.ecrypt.eu.org/stream.
7. Biryukov A., Mukhopadhyay S., Sarkar P.: Improved time-memory tradeoffs with multiple data. Proceedings of Selected Areas in Cryptography 2005, Lecture Notes in Computer Science 3897, pp. 245–260, Springer, Berlin (2006).
8. Biryukov A., Shamir A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers, Advances in Cryptology. Proceedings of ASIACRYPT 2000, Lecture Notes in Computer Science 1976, pp. 1–13, Springer, Berlin (2000).
9. Blum M., Micali S.: How to generate cryptographically strong sequences of pseudo-random bits, SIAM J. Comput. **13**(4), 850–864 (1984).
10. Bouillaguet C., Derbez P., Dunkelman O., Keller N., Rijmen V., Fouque, P-A.: Low data complexity attacks on AES. IEEE Trans. Inform. Theory. (2012). Available on-line at http://eprint.iacr.org/2010/633.
11. Bouillaguet C., Derbez P., Fouque P-A.: Automatic search of attacks on round-reduced AES and applications. Advances in Cryptography. Proceedings of CRYPTO 2011, Lecture Notes in Computer Science 6841, pp. 169–187, Springer, Berlin (2011).
12. Daemen J., Rijmen V.: AES proposal: rijndael. NIST AES proposal (1998).
13. Daemen J., Rijmen V.: The design of rijndael: AES, the advanced encryption standard, Springer, Berlin (2002).
14. Dunkelman O., Keller N.: Treatment of the initial value in time-memory-data tradeoff attacks on stream ciphers, Information Processing Letters, vol. 107, No. 5, pp. 133–137, Elsevier, Amsterdam (2008).
15. Dunkelman O., Keller N.: A new attack on the LEX stream cipher, Advances in Cryptology. Proceedings of ASIACRYPT 2008, Lecture Notes in Computer Science 5350, pp. 539–556, Springer, Berlin (2008).
16. Dunkelman O., Keller N., Shamir A.: Improved single-key attacks on 8-round AES-192 and AES-256, Advances in Cryptology, proceedings of ASIACRYPT 2010, Lecture Notes in Computer Science 6477, pp. 158–176, Springer, Berlin (2010).
17. ECRYPT: Call for stream cipher primitives version 1.3, 12.4.2005. Available on-line at http://www.ecrypt.eu.org/stream/call/.
18. Englund H.K., Hell M., Johansson T.: A note on distinguishing attacks. Preproceedings of State of the Art of Stream Ciphers workshop (SASC 2007), pp. 73–78, Bochum, Germany, (2007).
19. Ferguson N., Kelsey J., Lucks S., Schneier B., Stay M., Wagner D., Whiting D.: Improved cryptanalysis of rijndael. Proceedings of Fast Software Encryption 2000, Lecture Notes in Computer Science 1978, pp. 213–230, Springer, Berlin (2001).
20. Goldreich O., Levin L.A.: A hard-core predicate for all one-way functions. Proceedings of 21st STOC (1989), pp. 25–32, ACM, New York (1989).
21. Golic J.Dj.: Cryptanalysis of alleged A5 stream cipher, Advances in Cryptology. Proceedings of EUROCRYPT 1997, Lecture Notes in Computer Science 1233, pp. 239–255, Springer, Berlin (1997).
22. National Institute of Standards and Technology: Advanced Encryption Standard, Federal Information Processing Standards Publications No. 197, (2001).
23. Stad J.H., Näslund M.: BMGL: Synchronous key-stream generator with provable security, Submission to the NESSIE project, 2000. Available on-line at http://www.nessie.eu.org.
24. Wu H., Preneel B.: Attacking the IV setup of stream cipher LEX, ECRYPT stream cipher project report 2005/059. Available on-line at http://www.ecrypt.eu.org/stream.