A New Attack on the LEX Stream Cipher

Orr Dunkelman[1,2,*] and Nathan Keller[3,**]

[1] Katholieke Universiteit Leuven
Department of Electrical Engineering ESAT/SCD-COSIC
and
Interdisciplinary Institute for BroadBand Technology (IBBT)
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
`orr.dunkelman@esat.kuleuven.be`
[2] École Normale Supérieure
Département d'Informatique,
CNRS, INRIA
45 rue d'Ulm, 75230 Paris, France.
[3]Einstein Institute of Mathematics, Hebrew University.
Jerusalem 91904, Israel
`nkeller@math.huji.ac.il`

Abstract. In [6], Biryukov 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 phase 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 paper we suggest a key recovery attack on LEX. The attack requires about $2^{36.3}$ bytes of key-stream produced by the same key (possibly under many different IVs), and retrieves the secret key in time of 2^{112} simple operations. Following a preliminary version of our attack, LEX was discarded from the final portfolio of eSTREAM.

Keywords: LEX, AES, stream cipher design

1   Introduction

The design of stream ciphers, and more generally, pseudo-random number generators (PRNGs), has been a 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 [19]. An instantiation of this approach (even though an earlier one) is the Blum and Micali [11] construction (based on the hardness of RSA). A more efficiency-oriented construction is the BMGL stream cipher [21]

(based on the Rijndael block cipher). However, these constructions are relatively slow, and hence are not used in practical applications.

In [6], 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 filter 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 [6] 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 [7]). 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) phase of evaluation.

During the eSTREAM competition, LEX attracted a great deal of attention from cryptanalysts due to its simple structure, but nevertheless, only two attacks on the cipher were reported: A slide attack [23] requiring 2^{61} different IVs (each producing 20,000 keystream bytes), and a generic attack [17] requiring $2^{65.7}$ re-synchronizations. Both attacks are applicable only against the original version of LEX presented in [6], but not against the tweaked version submitted to the second phase of eSTREAM [8]. 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 paper we present an attack on LEX. The attack requires about $2^{36.3}$ bytes of key stream produced by the same key, possibly under different IVs. The time complexity of the attack is 2^{112} simple operations. Following a preliminary version of our attack, LEX was discarded from the final portfolio of eSTREAM.

Our attack is composed of three steps:

1. **Identification of a special state:** We focus our attention on pairs of AES encryptions whose internal states 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. Thus, the attacker repeats the following two steps for about 2^{32} cases which satisfy this 32-bit condition.

2. **Extracting information on the special state:** By using the special difference pattern of the pair of intermediate values, and guessing the difference in eight more bytes, the attacker can retrieve the actual values of 16 internal state bytes in both encryptions.

3. **Guess-and-Determine attack on the remaining unknown bytes:** Using the additional known byte values, the attacker can mount a guess-and-

determine attack that retrieves the key using about 2^{112} simple operations in total.

The second and the third steps of the attack use several observations on the structure of the AES round function and key schedule algorithm. [1] One of them is the following, probably 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.

This paper is organized as follows: In Section 2 we briefly describe the structures of AES and LEX, and present the observations on AES used in our attack. In Section 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 to retrieve the actual value of 16 bytes of the internal state (in both encryptions). In Section 4 we leverage the knowledge of these 16 bytes into a complete key recovery attack that requires about 2^{112} simple operations. We give several additional observations that may be useful for further cryptanalysis of LEX in Section 5. We conclude the paper in Section 6.

2 Preliminaries

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

2.1 Description of AES

The advanced encryption standard [14] is an SP-network that supports key sizes of 128, 192, and 256 bits. As this paper 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 4x4, where each byte represents a value in $GF(2^8)$. An AES round applies four operations to the state matrix:

---

[1] We note that in [6] 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 sub-keys by 1280 random bits. Our attack, which relies heavily on 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.
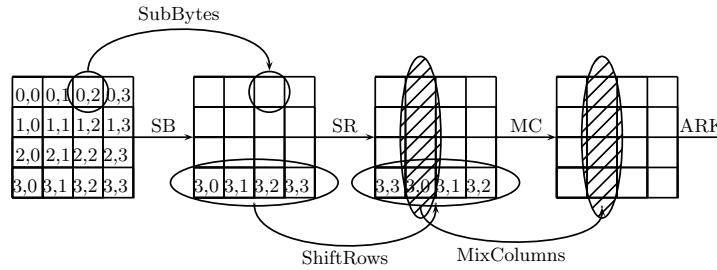
Fig. 1. An AES round

- 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 4x4 matrix over the field $GF(2^8)$, and
- AddRoundKey (ARK) — XORing the state with a 128-bit subkey.

We outline an AES round in Figure 1. 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 phase of eSTREAM [8]. The original version of LEX can be found in [6]. We note that our attacks can be easily adopted to the untweaked version as well.

In the initialization step, the publicly known IV is encrypted by AES[2] under the secret key K to get $S = AES_K(IV)$. Then, S is repeatedly encrypted in OFB

---

[2] 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.
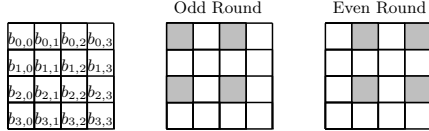
The gray bytes are the output bytes.

Fig. 2. Odd and Even Rounds of LEX

mode of operation under K, where during the execution of each encryption, 32 bits of the internal state are leaked each round. These state bits compose the key stream of LEX. The state bytes used in the key stream are shown in Figure 2.2. After 500 encryptions, another IV is chosen, and the process is repeated. After 2^{32} different IVs, the secret key is replaced. [3]

2.3 Notations Used in the Paper

As in [6], the bytes of each internal state during AES encryption, as well as the bytes of the sub-keys, 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 paper 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 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

---

[3] 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 [23], the number of IVs used with each key was restricted. This restriction also prevents the attack suggested later in [17] which requires $2^{65.7}$ re-synchronizations.

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.

Our attack uses this observation in situations where the attacker 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 attacker can retrieve the corresponding actual values using a simple table lookup.

Observation 2 *Since the MixColumns operation is linear and invertible, 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 [18]), it appears that the second was not published before.

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

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

and the second claim follows similarly.

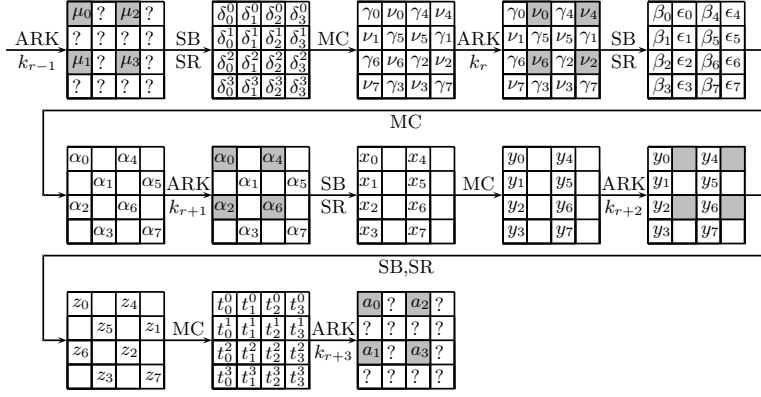**Observation 4** *For each $0 \leq i \leq 3$, the sub-keys of AES satisfy the relation:*

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

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

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

Thus,

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

In gray we mark bytes whose value is known from the output key stream.

Fig. 3. The Special Difference Pattern (for Odd Rounds)

These two observations allow the attacker 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 Observable Difference Pattern in LEX

Our attack is 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 is observed 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 (denoted by $\Delta(r+1)$) 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 Figure 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 encryptions under the same key, one of the pairs satisfies the difference pattern, and about 2^{32} pairs satisfy the filtering condition. Thus, the following steps of the attack have to be repeated 2^{32} times on average (once for each candidate pair).

7

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 (denoted by β-s and ϵ-s in Figure 3), and in each of the columns after the MixColumns operation of round $r + 2$ (denoted by t-s in Figure 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 attacker 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 attacker can retrieve the actual value of byte $b_{0,0}$ of the state in the end of round r. The derivation of additional 15 bytes, which is performed in a similar way, is described briefly.

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:

1. The attacker guesses the differences ν_1, ν_7 and applies the following steps for each such guess.
2. The attacker finds the difference in Column 0 before the MixColumns operation of round $r + 1$, i.e., $(\beta_0, \beta_1, \beta_2, \beta_3)$. This is possible since the attacker knows the difference in Column 0 at the end of round $r + 1$ (which is $(\alpha_0, 0, \alpha_2, 0)$ where α_0 and α_2 are known from the key stream), and since the AddRoundKey and the MixColumns operations are linear. By performing the inverse ShiftRows operation, the attacker can compute the output difference in byte $b_{0,0}$ after the SubBytes operation of round $r + 1$.
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 attacker knows the difference in two bytes of the same column before the MixColumns operation. Hence, using Observation 2 (the linearity of the MixColumns operation), the attacker retrieves the difference in the whole column, both before and after the MixColumns operation, including the difference γ_0.
4. At this point, the attacker 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 attacker finds the actual values of this byte using a single table look-up. In particular, the attacker 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).
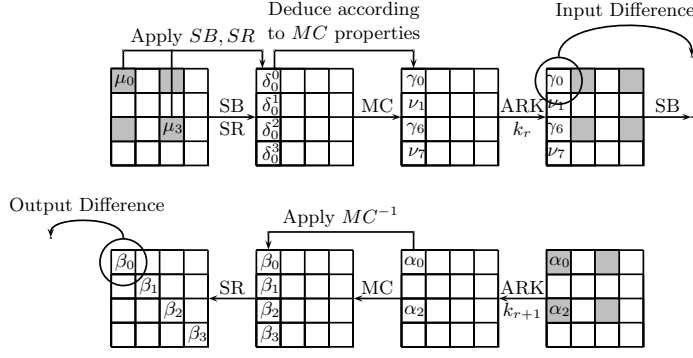
Fig. 4. Deducing the actual value of $b_{0,0}$ in the end of round r

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 attacker 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 attacker 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 attacker 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 attacker 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 attacker at this stage are presented in Figure 5 marked in gray.

4 Retrieving the Key in the Special Cases

The last step of the attack is a guess-and-determine procedure. Given the actual values of the 16 additional state bytes obtained in the second step of the attack, the entire key can be recovered using Observations 2 and 3 (properties of the MixColumns operation and of the key schedule algorithm of AES-128).

The deduction is composed of two phases. In the first phase, presented in Figure 5, no additional information is guessed. At the beginning of the second phase, presented in Figure 6, the attacker guesses the value of two additional sub-key bytes. In both figures we use gray bytes to mark bytes which are known at the beginning of that deduction phase. Then, if a byte contains a number i it means that this byte is computed in the ith step of the deduction sequence.

Summarizing the attack, the attacker guesses 10 bytes of information (8 bytes of differences guessed in the second step of the attack, and 2 sub-key bytes
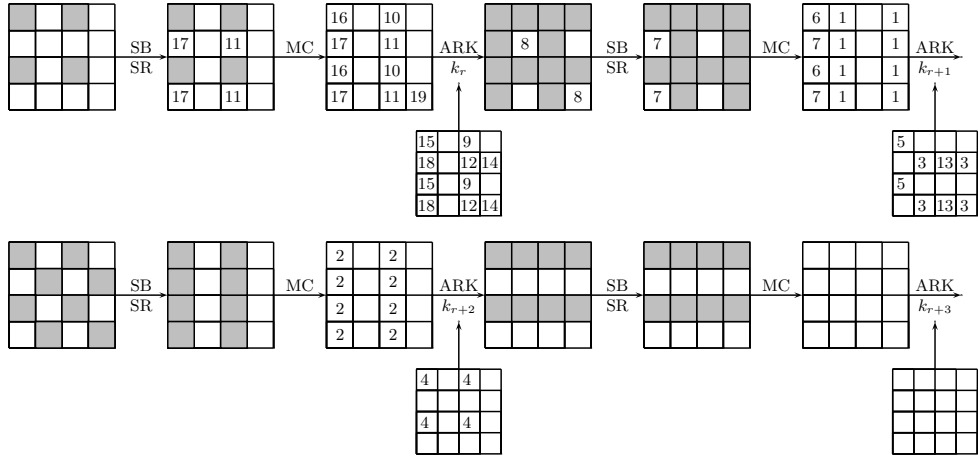
Gray boxes are bytes which are known.
Bytes marked with i, are bytes which are computed in step i.
Transition 9 is based on Observation 3.

Fig. 5. The First Phase of the Guess-and-Determine Attack on LEX (for Odd Rounds)

guessed in the third step of the attack), and retrieves the full secret key. Since all the operations used in the attack are elementary, the attack requires 2^{80} simple operations for each time the attack procedure is applied. Thus, as the attack procedure is repeated 2^{32} times, the total running time of the attack is 2^{112} operations. 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.

4.1 Data Complexity of the 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 $2^{32.5}$ encryptions under the same key (possibly with different IVs) yield a single pair of encryptions satisfying the special pattern.

However, we note that the attack can be applied for 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.[4] Hence, $2^{64}/8 = 2^{61}$ pairs of encryptions are sufficient to supply a pair satisfying one of the eight possible

---

[4] We note that while the attack considers five rounds of the 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
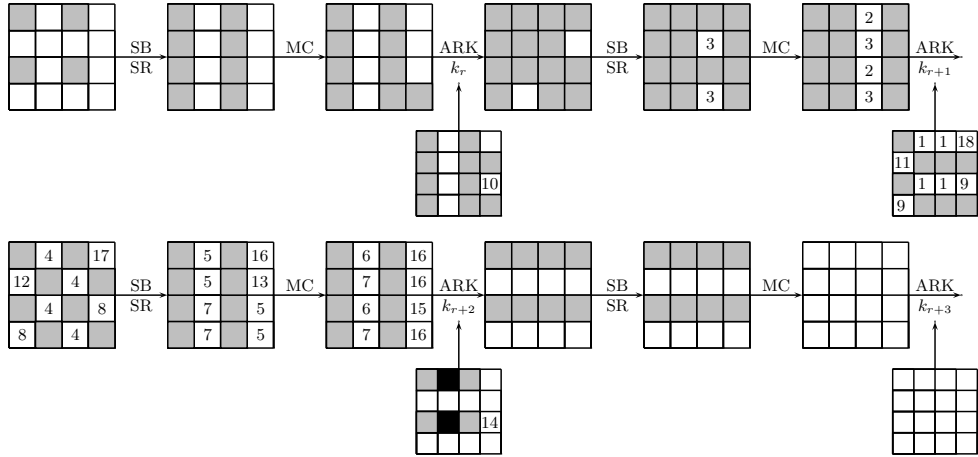
Gray boxes are bytes which are known.
Black boxes are the two bytes guessed in this phase.
Bytes marked with i, are bytes which are computed in step i of the second phase.

Fig. 6. The Second Phase of the Guess-and-Determine Attack on LEX (for Odd Rounds)

difference patterns. These 2^{61} pairs can be obtained from 2^{31} AES encryptions, or equivalently, $2^{36.3}$ bytes of output key stream generated by the same key, possibly under different IVs.

5   Further Observations on LEX

Sampling resistance of LEX:
   One of the main advantages of LEX, according to the designers (see [6], Section 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, 9, 10, 20] 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, [16]). 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

---

   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.

internal state is at least twice the size of the key [10]. 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 [6], Sections 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 2^{-t}, if it is possible to list 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 easy to see that LEX has maximal sampling resistance of 2^{-32}, as out of the 256 bits of internal state, 32 bits are output directly every round. As a result, using the attack algorithm presented in [10], it is possible to mount a TMDTO attack on LEX with data complexity 2^{88}, and time and memory complexities of 2^{112}. Hence, LEX provides only 112-bit security with respect to TMDTO attacks.

Loss of Entropy in the initialization of LEX:

Another observation concerning the way LEX behaves is as follows. The first step in the initialization of LEX, is the encryption of IV under K. When considering the function $AES_K(IV)$, once can easily see that under reasonable randomness assumptions on AES, this function is a random function for the key K. In other words, the first internal state S, used in LEX, does not contain 128 bits of entropy (even when the IV does). Actually, the expected number of possible S's for a given IV is about 63% of all possible values, i.e., about $2^{127.3}$ possible S's.

Even though our attack does not use this observation, it might still be used in attacks which rely on entropy. Especially, the variant of [15] of time-memory-data tradeoff attacks (trying to invert the function $(key, IV) \rightarrow keystream$) might use this observation by trying to invert the function $(key, S) \rightarrow keystream$.

Attacks on the submitted reference implementation of the untweaked LEX: After communicating our preliminary attack, we received a request to discuss the implementation of the untweaked LEX submitted to eSTREAM. According to a claim made in [3] and verified later by us, the submitted code of the untweaked LEX outputs different bytes than intended and specified (specifically, in the even rounds, $b_{1,1}, b_{1,3}, b_{3,1}$ and $b_{3,3}$ are given as the key stream).

When considering differential properties of this variant, given the difference in the key stream in one even round, and its consecutive odd round, it is possible to immediately find the difference in two columns (i.e., of 8 more internal bytes) without any assumption on the difference between the states. It is also possible to devise a simple meet-in-the-middle attack which uses 128 bits of output stream and takes 2^{112} simple operations.

Such results put emphasize on the crucial importance of verifying the implementations of cryptographic primitives very carefully. This was indeed first

observed in [12] with respect to public key encryption, and adopted in [5] to the symmetric key scenario. While differential fault analysis assumes that the attacker can access both a faulty implementation as well as a regular implementation, our observations are valid to the case when the attacker needs to attack only the faulty implementation.

6   Summary and Conclusions

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

The attack uses a total of $2^{36.3}$ bytes of key stream produced by a single key (possibly under different IVs) and takes 2^{112} simple operations to implement.

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.

References

1. Steve H. Babbage, Matthew Dodd, *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. Steve H. Babbage, *Improved "exhaustive search" attacks on stream ciphers*, IEE European Convention on Security and Detection, IEE Conference publication 408, pp. 161–165, IEE, 1995.
3. Dan J. Bernstein, personal communication, 2008.
4. Eli Biham, Adi Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
5. Eli Biham, Adi Shamir, *Differential Fault Analysis of Secret Key Cryptosystems*, Advances in Cryptology, proceedings of CRYPTO 97, Lecture Notes in Computer Science 1294, pp. 513–525, Springer, 1997.
6. Alex Biryukov, *The Design of a Stream Cipher LEX*, Proceedings of Selected Areas in Cryptography 2006, Lecture Notes in Computer Science 4356, pp. 67–75, Springer-Verlag, 2007.
7. Alex Biryukov, *A New 128-bit Key Stream Cipher LEX*, ECRYPT stream cipher project report 2005/013. Available at http://www.ecrypt.eu.org/stream.
8. Alex Biryukov, *The Tweak for LEX-128, LEX-192, LEX-256*, ECRYPT stream cipher project report 2006/037. Available at http://www.ecrypt.eu.org/stream.
9. Alex Biryukov, Sourav Mukhopadhyay, Palash Sarkar, *Improved Time-Memory Tradeoffs with Multiple Data*, Proceedings of Selected Areas in Cryptography 2005, Lecture Notes in Computer Science 3897, pp. 245–260, Springer-Verlag, 2006.

10. Alex Biryukov, Adi Shamir, *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-Verlag, 2000.
11. Manuel Blum, Silvio Micali, *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits*, SIAM journal of Computation No. 13, Vol. 4, pp. 850–864, SIAM, 1984.
12. Dan Boneh, Richard A. DeMillo, Richard J. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)*, Advances in Cryptology, proceedings of EUROCRYPT 97, Lecture Notes in Computer Science 1233, pp. 37–51, Springer, 1997.
13. Joan Daemen, Vincent Rijmen, *AES Proposal: Rijndael*, NIST AES proposal, 1998.
14. Joan Daemen, Vincent Rijmen, *The design of Rijndael: AES — the Advanced Encryption Standard*, Springer-Verlag, 2002.
15. Orr Dunkelman, Nathan Keller, *Treatment of the Initial Value in Time-Memory-Data Tradeoff Attacks on Stream Ciphers*, preproceedings of State of the Art of Stream Ciphers workshop (SASC 2008), pp. 249–258, Lausanne, Switzerland, 2008.
16. ECRYPT, *Call for Stream Cipher Primitives*, version 1.3, 12.4.2005. Available online at *http://www.ecrypt.eu.org/stream/call/*.
17. Håkan Englund, Martin Hell and Thomas Johansson, *A Note on Distinguishing Attacks*, preproceedings of State of the Art of Stream Ciphers workshop (SASC 2007), pp. 73–78, Bochum, Germany, 2007.
18. Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Mike Stay, David Wagner, Doug Whiting, *Improved Cryptanalysis of Rijndael*, Proceedings of Fast Software Encryption 2000, Lecture Notes in Computer Science 1978, pp. 213–230, Springer-Verlag, 2001.
19. Oded Goldreich, Leonid A. Levin, *A Hard-Core Predicate for all One-Way Functions*, Proceedings of 21st STOC (1989), pp. 25–32, ACM, 1989.
20. Jovan Dj. Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, Advances in Cryptology, Proceedings of EUROCRYPT '97, Lecture Notes in Computer Science 1233, pp. 239–255, Springer-Verlag, 1997.
21. Johan Håstad, Mats Näslund, *BMGL: Synchronous Key-stream Generator with Provable Security*, submission to the NESSIE project, 2000. Available online at *http://www.nessie.eu.org*.
22. National Institute of Standards and Technology, *Advanced Encryption Standard*, Federal Information Processing Standards Publications No. 197, 2001.
23. H. Wu and B. Preneel, *Attacking the IV Setup of Stream Cipher LEX*, ECRYPT stream cipher project report 2005/059. Available at http://www.ecrypt.eu.org/stream.

A   Attacking the Even Rounds

The attack on the even rounds is very similar to the attack on the odd rounds. For sake of completeness we give here the development of differences, and the deduced bytes (and the sequence of guesses) that allows to find the key. We note that the basic step of finding some internal state bytes after analyzing the differences is very similar to the odd rounds case. We then follow a slightly different path in the deduction step, due to the slightly different positions of known bytes. Despite that, the overall complexity of the attack in this situation is the same.
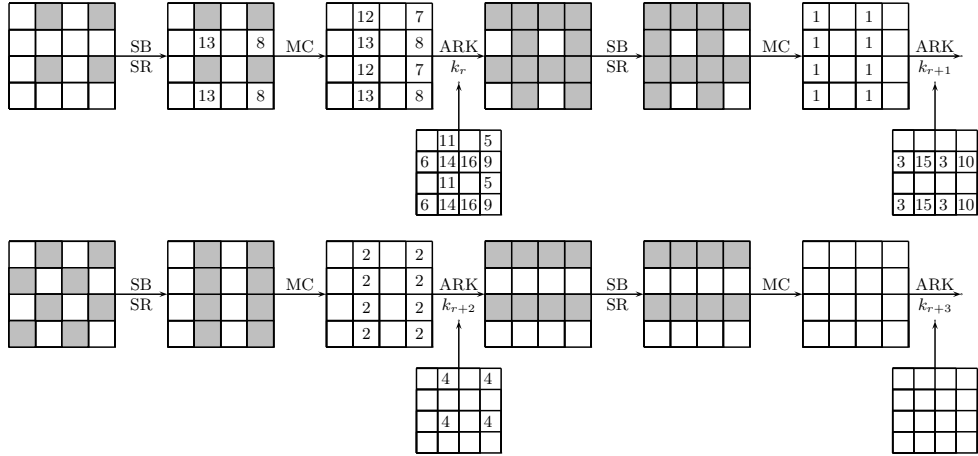
Gray boxes are bytes which are known.

Bytes marked with i, are bytes which are computed in step i.

Transition 5 is based on Observation 3, and transition 11 is based on Observation 4.

Fig. 7. The First Phase of the Guess-and-Determine Attack on LEX (in Even Rounds)
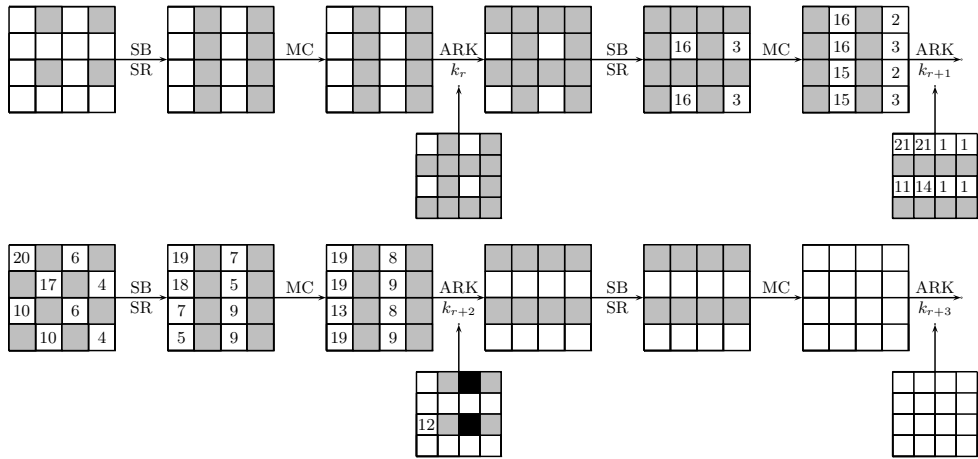
Gray boxes are bytes which are known.

Black boxes are the two bytes guessed in this phase.

Bytes marked with i, are bytes which are computed in step i of the second phase.

Fig. 8. The Second Phase of the Guess-and-Determine Attack on LEX (in Even Rounds)

15