Related Key Cryptanalysis of the LEX Stream Cipher

Mainack Mondal and Debdeep Mukhopadhyay

Dept. of Computer Science and Engineering
Indian Institute of Technology Kharagpur,India

{mainack,debdeep}@cse.iitkgp.ernet.in

Abstract. *LEX is a stream cipher proposed by Alex Biryukov. It was selected to phase 3 of the eSTREAM competition. LEX is based on the Advanced Encryption Standard (AES) block cipher and uses a methodology called "Leak Extraction", proposed by Biryukov himself. In this paper, we cryptanalyze LEX using two related keys. We have mounted a key recovery attack on LEX, which using $2^{54.3}$ key streams yields a complete round key with 2^{102} operations. This improves the existing best cryptanalysis of LEX which needs 2^{112} operations to ascertain the key.*

1 Introduction

LEX is a stream cipher designed by Alex Biryukov [1] using the round transformations of the Advanced Encryption Standard (AES). The proposal was built on the concept of *leak extraction* and was based on the analysis of the diffusion of the transformations of AES to decide the extent of the *leak* and its frequency. It is a 128 bit key stream cipher, which is developed by extracting 32 bits from the state matrix of AES after each round of the cipher. The property of LEX, which makes it different from AES is that the attacker never sees the entire 128 bit ciphertext, but sees a portion of it. The design objective of LEX was to design a fast stream cipher, and indeed was faster than AES in both hardware and software by at least 2.5 times.

LEX was selected as a candidate for eSTREAM competition and was seriously considered till the third phase for its elegance in construction, faster operation speed and high security margin. Its simplicity in construction attracted several cryptanalytic efforts from several researchers. The first reported attack on LEX was a slide attack by Wu and Preneel, which required 2^{61} different IVs (Initialization Vector), each producing $20,000$ key-stream bytes [2]. A generic attack followed by Englund et. al. , which needed $2^{65.7}$ resynchronizations [3]. However Biryukov submitted a tweaked version of LEX to the second phase of the eSTREAM competition in 2007, which could counter both the above attacks. However in 2008, Dunkelman and Keller [4] proposed a key recovery attack on *tweaked* LEX which required $2^{36.3}$ bytes of key-stream produced by the same key and had time complexity of 2^{112}. This attack removed LEX from the final portfolio of eSTREAM. But in spite of the above attacks, it may be appreciated that LEX is a stream cipher with high security margin although having an extremely simple design. The other selling point of LEX is that, the principle of leak extraction may be generalized to any block cipher, thus motivating deeper investigations of its strength. In this paper, we have performed a *Related Key Cryptanalysis* [5] of the LEX stream cipher.

The analysis is based upon a 5 round differential trail in the key-schedule of AES and is composed of the following steps:

- **Finding a special Difference Pattern:** This step focuses on a pair of AES encryptions, using two related keys K and K^*. Let the AES encryptions under the key K be sequenced as $E_1, E_2 \dots E_n$, and with the key K^* be denoted as $E'_1, E'_2 \dots E'_n$. The property being exploited is the collision of 12-bytes of the AES state matrices, before the r^{th} round of the AES encryption E_i with the corresponding state matrix obtained from E'_j. Here r can be either 3 or 5.
- **Key Recovery Step:** This step uses the above special Difference Pattern, along with the differential trail in the key schedule to obtain the 16 bytes of the key.

The attack requires $2^{54.3}$ key-streams and has a time complexity of 2^{102}, hence reducing the time complexity of the attack proposed by Dunkelman at an increased cost of required key streams.

We note that under the restriction that a secret key can be used to produce only $2^{46.3}$ key streams as given in the tweaked version of LEX our attack fails. This is due to the fact that it needs too much data. But one should still consider this attack important due to mainly two reasons

1. This attack shows a basic weakness of LEX against the differential attacks and does show that the key streams generated by LEX give a differential pattern that can be exploited by an attacker. Hence if one wants to improve the design of LEX, he should be concerned about this kind of related key based attacks.
2. Cryptographic attacks are always improving and their complexity is never increasing. Hence this attack provides a platform for the effort of possible future related key based attacks on the stream cipher LEX which most probably require lesser data and time complexity.

The rest of this paper is organized as follows. In the next section we state some key concepts and notations that are necessary to understand this work. In section 3 we demonstrate two differential properties of LEX. In section 4 using these differential properties we mount a key recovery attack on LEX. Then we discuss the time and data complexity of the attack in section 5. We give a comparative study of all the reported attacks on LEX in section 6. Lastly in section 7 we conclude this paper.

2 Preliminaries

2.1 Description of AES

The Advanced Encryption Standard (AES) is an 128 bit block cipher that supports key sizes of 128, 192 and 256. Since the construction of LEX is based on the structure of AES in this section we briefly state the AES algorithm and AES key schedule.

AES considers a 128 bit plaintext as a byte matrix of size 4×4. Here each byte represents an element of $GF(2^8)$. An AES round consists of 4 operations applied to the state matrix in the following order :

- SubByte − We apply a 8 bit to 8 bit invertible transformation to each of the bytes of the state matrix in parallel.

– ShiftRow (SR) − Each byte of i^{th} row is given a cyclic left shift of i bytes, $i = 0, 1, 2, 3$.
– MixColumn (MC) − Each column of the state matrix is multiplied by a constant 4×4 matrix. The multiplication is carried out in $GF(2^8)$
– AddRoundKey (ARK) − The state matrix is xor-ed with a 128 bit round subkey derived from the original 128 bit key following a key schedule algorithm.

In an `AES - 128` encryption consecutive 10 rounds are applied on the 128 bit plaintext. There is an extra xoring with a 128 bit subkey before the first round. This is called *'key whitening'*. In the last round the MC operation is omitted.

`AES - 128`, i. e. , `AES` with 128 bit keys uses 10 rounds as stated. For this a total of 11 round subkeys are required. By a key scheduling algorithm these round keys are constructed. Let us denote the subkey array as $W[0...43]$, where each of $W[i]$ consists of 32 bits. The first four words of W are filled with the supplied key. Rest of W is deduced according to the following rule :

for$(i = 4;\ i \leq 43;\ i++)$
  if $(i \equiv 0\ mod\ 4)$
    $W[i] = W[i-4] \oplus SubByte\,(W[i-1] <<< 8) \oplus RCON[i/4]$
  else
    $W[i] = W[i-1] \oplus W[i-4]$

Where $RCON$ is an array of predetermined constants and $<<<$ denotes cyclic rotation of the word by 8 bit to the left.

2.2 Description of `LEX`

We describe the tweaked version of `LEX` as explained in [1]. In the initialization step the publicly known `IV` value is encrypted by `AES` using a secret key K to get S , hence $S = AES_K(IV)$. In this phase no keystream bytes are generated. Then S is repeatedly encrypted by using a modified version of `AES` algorithm in OFB mode of operation. In this modified version of AES the *'key whitening'* step before the first round is omitted and the MC step in the last round is retained. For sake of clarity throughout this paper by the term AES we refer to this modified AES.

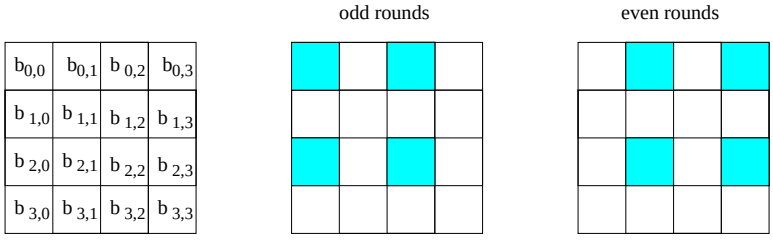

Fig. 1. Output bytes in odd and even rounds of `LEX`

While doing these `AES` encryptions, 32 bits of the state matrix after each round are leaked. These leaked bits compose the key stream of `LEX`. The bytes leaked in different rounds are shown in *figure 1.* Therefore each encryption produces 40 bytes of key stream.

After 500 encryptions, another IV is chosen, and the process is repeated. After 2^{32} different IVs the key is replaced. Hence under this restriction by a single secret key we can get at most $500 \times 2^{32} \times 40$ bytes of key stream $= 2^{46.3}$ bytes of keystream.

2.3 Notations

In this paper the bytes of any intermediate state matrix B, of an AES encryption are denoted by $\{B_{i,j}\}_{i,j=0}^{3}$. Let E and E' are two AES encryptions. The secret key for E is K and that for E' is K^*. Let an intermediate state matrix of $E(E')$ is B. Then the difference between B and the state matrix of $E'(E)$ corresponding to B is denoted by ΔB. The bytes of ΔB are denoted by $\{\Delta B_{i,j}\}_{i,j=0}^{3}$. The r^{th} round key derived from a key K is named as K^r and its bytes are $\left\{K_{i,j}^r\right\}_{i,j=0}^{3}$. In our attack we have used two related keys K and K^*. The difference between bytes of r^{th} round key derived from K and K^* is $\left\{\Delta K_{i,j}^r\right\}_{i,j=0}^{3}$. In this work $SB(x)$ denotes the output difference for an input difference of x to the SubByte operation and $SubByte(x)$ denotes the output byte for an input byte value of x to the SubByte operation. Throughout this work we have used addition in $GF(2^8)$. So the symbol '+' in this work refers to the operation xor.

2.4 Observations on AES SubByte operation

LEX uses the same SubByte operation as AES. We have extensively used two observations on SubByte operation of AES in our attack. We state these observations below. They are well explained in [4].

Observation 1 : *Given any non zero input(output) difference to the SubByte operation, there are a total of 127 output(input) differences possible.*

As a result, given a non zero input difference β to the SubByte operation and a random non zero difference γ, probability of the event that the difference propagation $\beta \rightarrow \gamma$ holds for SubByte operation $= \Pr\left[SB\left(\beta\right)) = \gamma\right] = \frac{127}{255} \approx \frac{1}{2}$. We have used this property as a filter in our attack.

Observation 2 : *Given the input and output differences to the SubByte operation we can retrieve the corresponding actual input and output values using a single table look up on average.*

In the following section, we outline two important properties of the LEX stream cipher, which we have used to perform the related key cryptanalysis. First we shall state a special differential in the key schedule of LEX. Next we outline a special type of collision of the state matrices of LEX, which is used to carry on the attack. We apply Birthday Paradox to compute the number of key-streams which shall make the probability for such a collision high.

3 Special Differential Properties of LEX

In this section, we outline two differential properties: one for the key-schedule, the other for the state matrices of LEX. Here it may be pointed out that we are

assuming that the key-schedule of LEX is the same as that of AES. However as mentioned in [1] if the key-schedule of AES is modified so that the round keys are randomly generated then the following analysis does not hold. In that case, the properties of the specific key scheduling technique needs to be investigated.

3.1 The Related Key differentials in LEX key schedule

LEX essentially uses the same key schedule as AES. We have used a five round differential in our attack. We explain the difference propagation pictorially (*figure 2*).

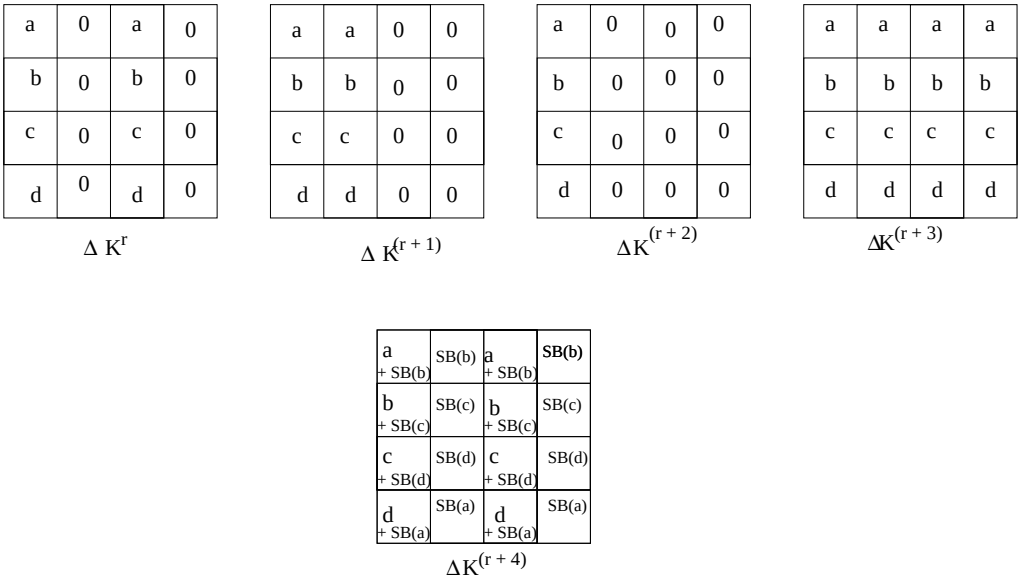

Fig. 2. Propagation of Differences in the LEX key Schedule

The figure shows the propagation of the differentials through the key-scheduling algorithm of LEX. The attacker considers the related keys, K and K^*, which maintains a known difference in r^{th} round keys. This known difference is $(\alpha, 0, \alpha, 0)$, where α is a 32 bit value equal to a, b, c, d, where a, b, c, d are four non-zero byte values. In out attack r has to be odd. Moreover we have used in our attack the key streams taken after the rounds $r-1$, r, $r+1$, $r+2$, $r+3$ and $r+4$ of the same AES encryptions. All these constraints set the value of r as 3 or 5. In the actual attack scenario the attacker can assign any of these values to r keeping the attack procedure unchanged.

3.2 Special Differential Properties in the LEX state matrices

In this attack we have used two related keys K and K^*. The r^{th} round keys generated from K and K^* follows the difference pattern shown in section 3.1. Our attack is successful when a special difference pattern appears just before the AddRoundKey step of the r^{th} round. We first describe how to find the difference pattern using the key streams.

Consider two AES encryptions generated by keys K and K^*. Here K and K^* are related keys. Let us consider the event when the state matrices just before the r^{th} AddRoundKey operation in both the encryptions will have zero differences in 12 specific byte positions. Hence mathematically, if the corresponding differential state matrix be denoted by Δb, then $\{\Delta b_{0,0}, \Delta b_{0,1}, \Delta b_{0,2}, \Delta b_{0,3}, \Delta b_{1,0}, \Delta b_{1,2}, \Delta b_{2,0}, \Delta b_{2,1}, \Delta b_{2,2}, \Delta b_{2,3}, \Delta b_{3,0}, \Delta b_{3,2}\} = 0$, and $\{\Delta b_{1,1}, \Delta b_{1,3}, \Delta b_{3,1}, \Delta b_{3,3}\}$ are non zero values. This pattern is shown in *figure 3* along with the value of the differences after the r^{th} AddRoundkey operation. The colored bytes have unknown differences.

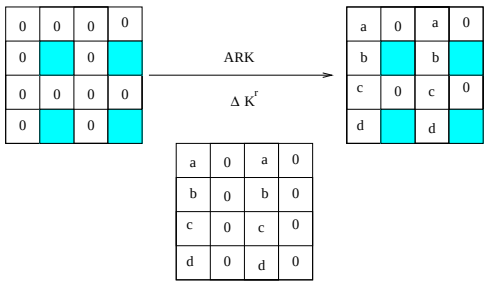

Fig. 3. The Special Difference Pattern

Next we compute the minimum number of key streams required to obtain such a difference pattern. Let the AES encryptions under the key K be sequenced as $E_1, E_2 \ldots E_n$, and with the key K^* be denoted as $E'_1, E'_2 \ldots E'_n$. The two colliding AES encryption pairs are obtained from the two sequences and may be denoted as E_i and E'_j.

Let us assume that with a minimum number of n AES encryptions for each of the keys K and K^* the attacker has at least one pair of encryptions with high probability where the required pattern holds. Consequently for that pair before the AddRoundKey operation of r^{th} round, at 12 byte positions the actual values of the two state matrices generated by K and K^* will be same. We call this event a collision for a pair. The probability of a collision between two random pairs is $\frac{1}{2^{96}}$ Let us define the following event:

X_{ij} : A collision occurs between the pair E_i and $E'_j \Rightarrow \Pr[X_{ij}] = \frac{1}{2^{96}}$
So, $\overline{X}_{ij}$: A collision will not occur between the pair E_i and $E'_j \Rightarrow \Pr[\overline{X}_{ij}] = 1 - \frac{1}{2^{96}}$

$$\begin{aligned}
\text{Now, } \Pr[\text{A collision occurs for at least one pair}] &= 1 - \Pr[\text{no pairs will be collided}] \\
&= 1 - \Pr\left[\bigcap_{i=1}^{n}\bigcap_{j=1}^{n}\overline{X}_{ij}\right] \\
&= 1 - \left(\Pr\left[\bigcap_{j=1}^{n}\overline{X}_{ij}\right]\right)^n \\
&= 1 - \left(1 - \Pr\left[\bigcup_{j=1}^{n}X_{ij}\right]\right)^n \\
&= 1 - \left(1 - \sum_{j=1}^{n}\Pr\left[X_{ij}\right]\right)^n \\
&= 1 - \left(1 - \sum_{j=1}^{n}\frac{1}{2^{96}}\right)^n
\end{aligned}$$

Thus, $\Pr[\text{A collision will occur for at least one pair}] \geq \frac{1}{2} \Rightarrow 1 - \left(1 - \frac{n}{2^{96}}\right)^n \geq \frac{1}{2}$

using the inequality $e^x \geq 1 + x, \forall x \in \mathbb{R}$ we deduce the optimal value of n to be 2^{48} using a birthday paradox kind of computation. Hence to get a colliding pair with high probability we need 2^{48} encryptions by the key K and for each of them we have to consider 2^{48} encryptions by the key K^*. Hence there are a total of 2^{96} pairs.

Now using a 32 bit condition on the key stream we shall reduce the number of pairs to be considered. Let E_i and E'_j have collided. If the state matrix after r^{th} round of E_i is denoted by B, then $\Delta B_{0,0}, \Delta B_{2,0}, \Delta B_{0,2}, \Delta B_{2,2}$ should be a, c, a, c respectively. a, b, c, d refers to the known differences in the key scheduling as shown in *figure 2*. This property follows directly from the special difference pattern propagation combined with subkey differences. $\Delta B_{0,0}, \Delta B_{2,0}, \Delta B_{0,2}, \Delta B_{2,2}$ can be got directly from the key stream and can be checked. It is expected that out of 2^{96} pairs only 2^{64} pairs will satisfy this condition.

In the next step we shall further reduce the number of pairs using another condition on the key stream. Let E_i and E'_j have collided. If the state matrix after $(r+1)^{th}$ round of E_i is denoted by C, we can observe the differences $\Delta C_{0,1}, \Delta C_{0,3}, \Delta C_{2,1}, \Delta C_{2,3}$ for the remaining 2^{64} pairs from the key stream. It can be shown that for E_i and E'_j each of these four differences give a distinct linear equation in four variables b'_1, b'_2, d'_1, d'_2. Thus we can form 4 linear equations in 4 variables. Detailed equations can be found in section 4. Thus after observing $\Delta C_{0,1}, \Delta C_{0,3}, \Delta C_{2,1}, \Delta C_{2,3}$ from the key stream we can solve those linear equations to evaluate b'_1, b'_2, d'_1, d'_2.

The special difference pattern propagation combined with subkey differences state that the following conditions needs to be satisfied for E_i and E'_j :

$$b'_1 = SB(b)$$
$$b'_2 = SB(b)$$
$$d'_1 = SB(d)$$
$$d'_2 = SB(d)$$

From observation 1 of section 2.4 the probability of satisfying all of these 4 conditions is 2^{-4}. So after checking this condition, out of 2^{64} pairs only 2^{60} pairs need to be considered.

Hence we have to repeat the attack procedure explained in the next section for each of the 2^{60} pairs.

4 Mounting A Key Recovery Attack Using Special Differential Properties of LEX

So far we have stated all the necessary properties required for our attack. In this section using those properties we describe our key recovery attack on LEX. The attack is described in 3 steps. After the third step of the attack the attacker obtains several probable candidates for the secret key K. Then a brute force search within those candidates will reveal the correct key.

1. This step retrieves 8 bytes of the state matrix after round r. Let E_i and E'_j form a colliding encryption pair. Then the special difference shown in *figure 3* holds for this pair. Now we shall concentrate on the propagation of this special difference pattern through round $(r+1)$. This propagation is shown in *figure 4*. We shall explain the rest of this step using this figure.

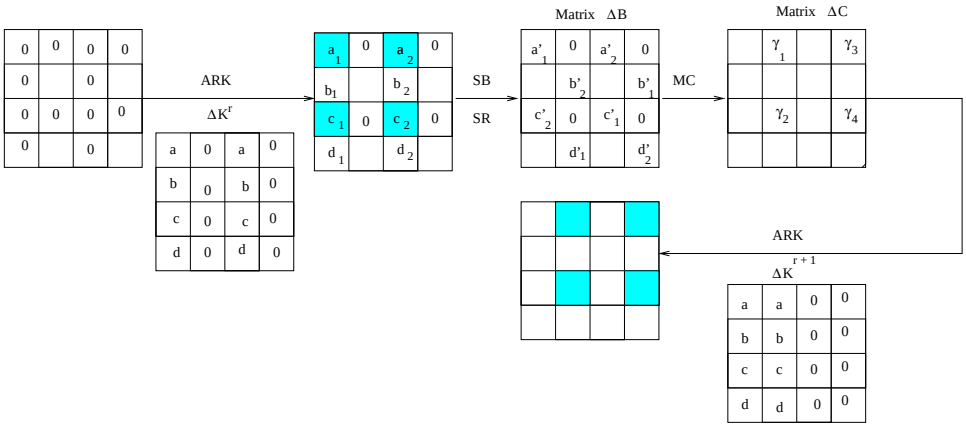

Fig. 4. Propagation of difference pattern in round (r + 1)(Bytes with known actual values are colored)

We know the actual values of the colored bytes of *figure 4* from the key stream. The symbols written in the byte positions are corresponding differences obtained for the pair E_i and E'_j. Let us first express the differences

$a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2$ in *figure 4* in terms of known differences a, b, c, d . The difference of the r^{th} subkeys for E_i and E'_j is known from the related key differential of section 3.1 and shown in *figure 4*. Observing the differences before and after xoring with round key K^r and using the linearity of ARK operation we immediately get $a_1 = a_2 = a$, $b_1 = b_2 = b$, $c_1 = c_2 = c$, $d_1 = d_2 = d$. In *figure 4* x' denotes output difference after SubByte operation applied to input difference x. Mathematically if x is a difference, $x' = SB(x)$ where $x = a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2$.
Now we observe the differences between the key stream generated by E_i and E'_j after $(r+1)^{th}$ round. From the related key differentials in `LEX` (section 3.1) we know the subkey difference ΔK^{r+1} as shown in *figure 4*. Hence using the linearity of ARK operation we can get the values of the differences $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ in *figure 4*. Let us denote the state matrix after SR operation of $(r+1)^{th}$ round as B. C is the state matrix after applying MC operation to B. Then using the linearity of MC operation we can express each of $\Delta C_{0,1}, \Delta C_{2,1}$ as linear combination of b'_2, d'_1. Also $\Delta C_{0,3}, \Delta C_{2,3}$ can be expressed as linear combination of b'_1, d'_2. Since $\Delta C_{0,1}, \Delta C_{2,1}, \Delta C_{0,3}, \Delta C_{2,3}$ are nothing but known values $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ respectively hence we get four equations in four variables b'_1, b'_2, d'_1, d'_2 as follows,

$$
\begin{aligned}
3b'_2 + d'_1 &= \gamma_1 \\
b'_2 + 3d'_1 &= \gamma_2 \\
3b'_1 + d'_2 &= \gamma_3 \\
b'_1 + 3d'_2 &= \gamma_4
\end{aligned}
$$

The attacker deduces b'_1, b'_2, d'_1, d'_2 from these equations. By virtue of knowing the differences b, d we know b_1, b_2, d_1, d_2. Hence using observation 2 of section 2.4 actual values of bytes corresponding to these four differences b_1, b_2, d_1, d_2 can be retrieved by four table look ups. Hence in this way we recover four bytes of the state matrix after the r^{th} round. Combined with the bytes we get from the key stream a total of eight bytes of the state matrix after the r^{th} round is recovered. The bytes of round $(r+1)$ whose actual values are known after this step are shown in *figure 5* by coloring them.

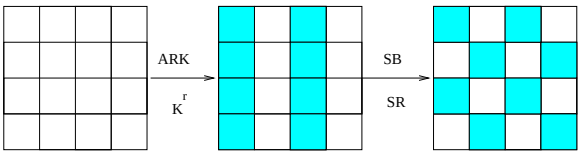

Fig. 5. Known bytes after the first step of attack

2. We recover some more bytes of another state matrix in this step. We continue with the colliding encryption pair E_i and E'_j from step 1. Here we concentrate on the propagation of differences in the r^{th} round. *figure 6* shows the differences of different bytes in the r^{th} round. The bytes whose actual values are known

to us from the key stream are colored. In this diagram, all of c_1, c_2, c_3, c_4 are differences we can get directly from the key stream. We want to determine x_i s, $i = 1, 2, .., 8$. They are unknown differences.

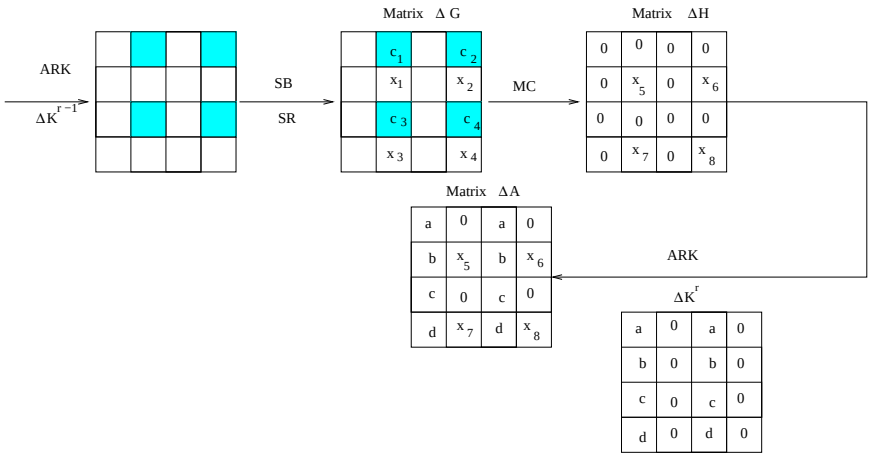

Fig. 6. Propagation of differences in round r

Here G is the state matrix after SR operation of round r in the encryption E_i. H is got after MC operation is applied to G. Since MC operation is linear we apply MC operation on 2^{nd} and 4^{th} column of ΔG and equate them with corresponding columns of ΔH. In this way we can form 8 linear equations involving x_i as follows :

$$
\begin{aligned}
3x_1 + x_3 &= 2c_1 + c_3 \\
2x_1 + x_3 + x_5 &= c_1 + 3c_3 \\
x_1 + 3x_3 &= c_1 + 2c_3 \\
x_1 + 2x_3 + x_7 &= 3c_1 + c_3 \\
3x_2 + x_4 &= 2c_2 + c_4 \\
2x_2 + x_4 + x_6 &= c_2 + 3c_4 \\
x_2 + 3x_4 &= c_2 + 2c_4 \\
x_2 + 2x_4 + x_8 &= 3c_2 + c_4
\end{aligned}
$$

We solve this system and in particular focus on the values of x_1, x_2, x_3, x_4 i. e the values of $\Delta G_{1,1}, \Delta G_{1,3}, \Delta G_{3,1}, \Delta G_{3,3}$. Thus we get the total 2^{nd} and 4^{th} column of ΔG. Next we guess the actual values of the 2^{nd} and 4^{th} column of the matrix G. This takes a complexity of 2^{32}. Hence using the MC operation we get the actual values of 2^{nd} and 4^{th} column of H too. Let the state matrix at the end of r^{th} round of E_i is denoted by A. Using the values of x_5, x_6, x_7, x_8 combined with the known value of ΔK^r we know the 2^{nd} and 4^{th} column of ΔA. The bytes whose actual values are known after step 2 are shown in

figure 7 by coloring them. We have also shown known differences in some byte positions.

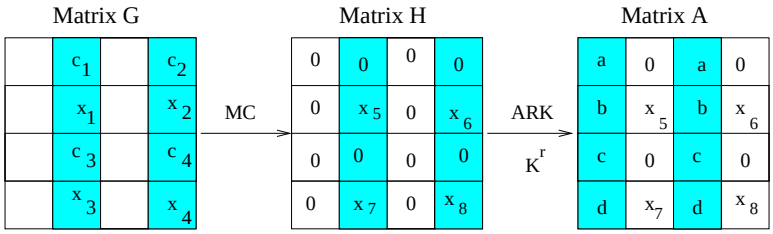

Fig. 7. Known bytes after the second step of attack

3. In this final step we derive candidates for the subkey K^{r+1}. Since we know ΔK^{r+1} (*figure 2*) once we retrieve K^{r+1} we shall automatically get candidates for $K^{*(r+1)}$. As in the previous two steps we consider E_i and E'_j as a colliding encryption pair. E_i uses secret key K and E'_j uses secret key K^*. Let us denote the state matrix of E_i at the end of round r by A. From step 2 we have the 2^{nd} and 4^{th} columns of ΔA. Now we guess the actual values of 2^{nd} and 4^{th} column of A. If we make all possible guesses for these 2 columns, then this step should have taken a complexity of 2^{64}. But we shall use several filters on these guesses, so that the average number of possible candidates for these 2 columns becomes 2^8. *figure 8* shows the known bytes at the beginning of this step by coloring them.

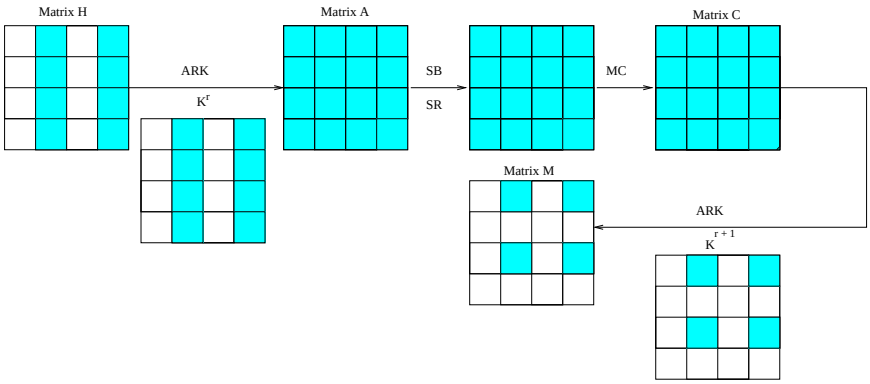

Fig. 8. Known bytes after the guessing phase at the third step of attack

After this guessing phase mentioned the actual values of 2^{nd} and 4^{th} column of A are known. Let us denote the state matrix after MC operation of round r in the encryption E_i by H. From step 2 of the attack we have got the actual values of the 2^{nd} and 4^{th} column of H. Hence by simple xor we got the 2^{nd} and 4^{th} column of K^r. The 1^{st} and 3^{rd} column of A is known from step 1 of the attack. Hence after guessing the 2^{nd} and 4^{th} column of A we got the actual values of all the 16 bytes of A. Let the state matrix of E_i after the MC operation of

round $(r+1)$ is denoted by C. Also let M be the state matrix after adding the round key K^{r+1} with C. We know the values $M_{0,1}, M_{0,3}, M_{2,1}, M_{2,3}$ from the key stream. Again by applying SubByte, SR and MC operation on A we get 16 bytes of C. Hence by simple xor we get $K^{r+1}_{0,1}, K^{r+1}_{0,3}, K^{r+1}_{2,1}, K^{r+1}_{2,3}$. Using these 4 bytes of K^{r+1} and 2^{nd} and 4^{th} column of K^r we can retrieve 4 more bytes of the $(r+1)^{th}$ round subkey (the detailed equations are in appendix A).

Now we want to recover some more bytes of K^{r+1} using the information available at this stage. The current known information is shown in *figure 9*. Here the colored bytes are known bytes. In *figure 9* we denote $K^{r+1}_{1,1}, K^{r+1}_{1,3}, K^{r+1}_{3,1}, K^{r+1}_{3,3}$ by unknown variables z_1, z_2, z_3, z_4. Then by the value of ΔK^{r+1}(*figure 2*) ${K^*}^{r+1}_{1,1}$, ${K^*}^{r+1}_{1,3}$, ${K^*}^{r+1}_{3,1}$, ${K^*}^{r+1}_{3,3}$ are (z_1+b), z_2, (z_3+d), z_4 respectively, where a, b, c, d are known subkey differences as shown in *figure 2*. In *figure 9* we have shown some rounds of both E_i and E'_j. E_i uses the subkey K^r and E'_j uses the subley K^{*r}. The symbols in the byte positions denote the actual value of the byte. In the same figure k_i, $i = 1, 2, ..., 12$ are known constants. We note that we use the ARK operation by the subkey K^{*r+1} on C^*, the counterpart of matrix C in E'_j and get M^*, the counterpart of matrix M in E'_j. Thus we directly get the equations

$$M^*{}_{i,j} = C^*{}_{i,j} + {K^*_{i,j}}^{r+1}, \quad \mathrm{i,j} \in \{1,3\}$$

Putting the values of the corresponding bytes as shown in *figure 9* we get the following relations : $k_5 = k_9 + b$, $k_6 = k_{10}$, $k_7 = k_{11} + d$, $k_8 = k_{12}$. These will help the attacker to evaluate the constants. In *figure 9* the symbol [.] denotes the SubByte operation. We color the bytes whose actual values are known.

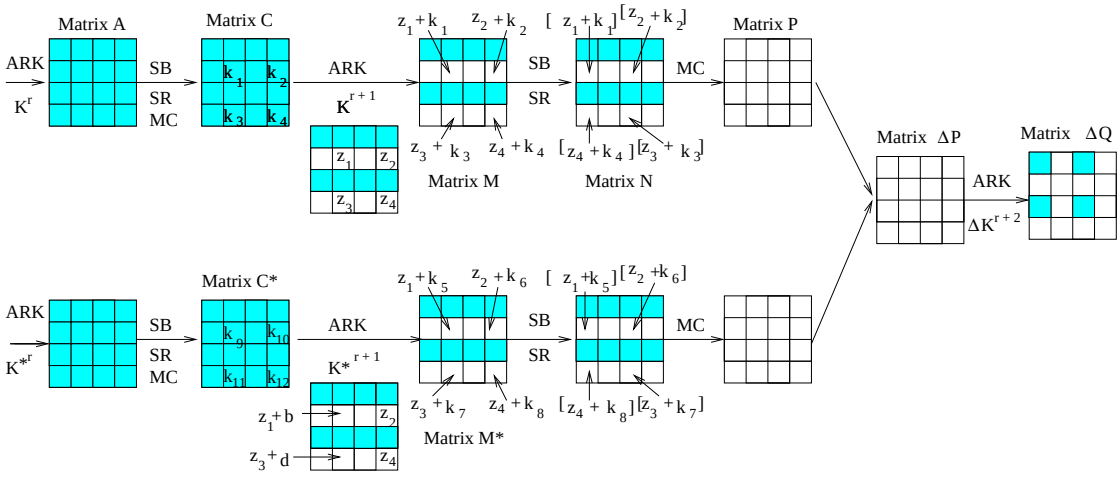

Fig. 9. Retrieving four more bytes of $(r+1)^{th}$ round key

Let us denote the state matrix of E_i after MC operation of $(r+2)^{th}$ round by P. Also let Q is the state matrix after adding the round key K^{r+2} with P. From the key stream we get $\Delta Q_{0,0}$, $\Delta Q_{0,2}$, $\Delta Q_{2,0}$, $\Delta Q_{2,2}$. Since we know ΔK^{r+2} (from *figure 2*) by the linearity of ARK operation we get $\Delta P_{0,0}$, $\Delta P_{0,2}$,

$\Delta P_{2,0}$, $\Delta P_{2,2}$. The state matrix of E_i after SR operation of $(r+2)^{th}$ round is N as shown in *figure 9*. Now after applying MC operation on 1^{st} column of N the bytes of the resultant column can be expressed as a function of two unknown values $SubByte(z_1+k_1)$ and $SubByte(z_4+k_4)$. The corresponding column for E'_j can be expressed as a function of two unknown values $SubByte(z_1+k_5)$ and $SubByte(z_4+k_8)$. Hence each of $\Delta P_{0,0}, \Delta P_{2,0}$ can be expressed as a function of four unknown variables $SubByte(z_1+k_1)$, $SubByte(z_4+k_4)$, $SubByte(z_1+k_5)$ and $SubByte(z_4+k_8)$. Since the actual values for $\Delta P_{0,0}, \Delta P_{2,0}$ are known, Hence we get two equations involving four variables. Similarly $\Delta P_{0,2}$, and $P_{2,2}$ also give two more equations involving four variables. We present the 4 equations below.

$$\begin{aligned}
3SubByte(z_1+k_1) + SubByte(z_4+k_4) + 3SubByte(z_1+k_5) \\
+SubByte(z_4+k_8) = \alpha_0 \\
SubByte(z_1+k_1) + 3SubByte(z_4+k_4) + SubByte(z_1+k_5) \\
+3SubByte(z_4+k_8) = \alpha_1 \\
3SubByte(z_2+k_2) + SubByte(z_3+k_3) + 3SubByte(z_2+k_6) \\
+SubByte(z_3+k_7) = \alpha_2 \\
SubByte(z_2+k_2) + 3SubByte(z_3+k_3) + 3SubByte(z_2+k_6) \\
+SubByte(z_3+k_7) = \alpha_3
\end{aligned}$$

Here α_0, α_1, α_2, α_3 are known constants. We can manipulate these equations to get equations like :

$$\begin{aligned}
SubByte(z_1+k_1) + SubByte(z_1+k_5) &= 3^{-1}(\ 4^{-1}(\alpha_0+\alpha_1)+\alpha_0) \\
SubByte(z_4+k_4) + SubByte(z_4+k_8) &= 3^{-1}(\ 4^{-1}(\alpha_0+\alpha_1)+\alpha_1) \\
SubByte(z_2+k_2) + SubByte(z_2+k_6) &= 3^{-1}(\ 4^{-1}(\alpha_2+\alpha_3)+\alpha_2) \\
SubByte(z_3+k_3) + SubByte(z_3+k_7) &= 3^{-1}(\ 4^{-1}(\alpha_2+\alpha_3)+\alpha_3)
\end{aligned}$$

Here 3^{-1} and 4^{-1} are finite field inverses computed as in `AES`. Since k_i, $i = 1, 2, .., 8$ and α_j, $j = 0, 1, 2, 3$ are known, these equations can be interpreted as, given the input and output differences to subbyte operation we have to find the actual input output pairs. First we have to examine whether each of these equations give a valid input output difference pair for the SubByte operation. Given α_i , $i = 0, .., 3$ this has a probability of $\frac{1}{2^4}$. Hence out of 2^{64} possible guesses only 2^{60} guesses pass through this condition. For the valid guesses we can determine the value of z_i, $i = 1, .., 4$ by simple table look ups.
After getting the values of z_1, z_2, z_3, z_4 at this stage we have 8 bytes of the r^{th} round subkey and 12 bytes of $(r+1)^{th}$ round subkey. We can deduce the full 16 bytes of $(r+1)^{th}$ round subkey from this information. Detailed equations are in appendix A. Now we can derive $(r+2)^{th}$ and $(r+3)^{th}$ round subkey from `LEX` key scheduling.
At this point we do another filtering. We know all 16 bytes of A for a particular guess about the 2^{nd} and 4^{th} column of A. Thus if the state matrix of E_i after

round $r+3$ is R we can derive all 16 bytes of R. Now we shall have actual values of $R_{0,1}, R_{0,3}, R_{2,1}, R_{2,3}$ from the keystream. If these 4 guessed values of R match with the corresponding actual values got from the key stream we shall keep the original guess about the 2^{nd} and 4^{th} column of A, otherwise discard it. The probability of matching for all these four values is $\frac{1}{2^{32}}$. Hence the expected number of guesses that pass this condition is 2^{28}.
Let us denote the state matrix of E_i just after the MC operation of the $(r+4)^{th}$ round by S and after the ARK operation of the $(r+4)^{th}$ round by T. Then for a guess about the 2^{nd} and 4^{th} column of A we can deduce values for S and hence ΔS can also be computed. From the key stream we shall get actual values for $\Delta T_{0,0}$, $\Delta T_{0,2}$, $\Delta T_{2,0}$, $\Delta T_{2,2}$. If our guess is correct we shall get the values for $\Delta K_{0,0}^{r+4}$, $\Delta K_{0,2}^{r+4}$, $\Delta K_{2,0}^{r+4}$, $\Delta K_{2,2}^{r+4}$ using the simple equations

$$\Delta K_{i,j}^{r+4} = \Delta S_{i,j} + \Delta T_{i,j}, \quad \mathrm{i,j} \in \{0,2\}$$

As we can see from the values of ΔK^{r+4} (*figure 2*) if the computed values for $\Delta K_{0,0}^{r+4}$, $\Delta K_{0,2}^{r+4}$, $\Delta K_{2,0}^{r+4}$, $\Delta K_{2,2}^{r+4}$ are correct the following properties should hold.

$$\Delta K_{0,0}^{r+4} = \Delta K_{0,2}^{r+4} = a + SB(b)$$
$$\Delta K_{2,0}^{r+4} = \Delta K_{2,2}^{r+4} = SB(b)$$

The probability that the ΔK^{r+4} given by a guess will satisfy the conditions $\Delta K_{0,0}^{r+4} = \Delta K_{0,2}^{r+4}$ and $\Delta K_{2,0}^{r+4} = \Delta K_{2,2}^{r+4}$ is $\left(\frac{1}{2^8}\right)^2$. Also when $\Delta K_{0,0}^{r+4}$ and $\Delta K_{0,2}^{r+4}$ are equal then the event that they have the exact form of $a+SB(b)$ has a probability of $\frac{1}{2}$. Similar reasoning goes for the pair $\Delta K_{2,0}^{r+4}$ and $\Delta K_{2,2}^{r+4}$ also. Thus the event that all of the differences $\Delta K_{0,0}^{r+4}$, $\Delta K_{0,2}^{r+4}$, $\Delta K_{2,0}^{r+4}$, $\Delta K_{2,2}^{r+4}$ has the correct form has a total probability of $\frac{1}{2^2}$. So out of 2^{28} remaining guesses only $\left(\frac{1}{2^8}\right)^2 \times \frac{1}{2^2} \times 2^{28} = 2^{10}$ guesses will remain.
So applying all stages of filtering we have to consider a total of only 2^{10} cases for guessing the actual values of 2^{nd} and 4^{th} column of the state matrix A and for each of them we shall deduce 16 bytes of the key.

Hence after executing 3 steps of the attack we have retrieved all 16 bytes of the subkey K^{r+1}. Consequently by key scheduling of `LEX` we shall retrieve all 16 bytes of the key K.

5 Complexity analysis of the attack

5.1 Time complexity

The step 1 of the attack methodology is deterministic and does not effect the total complexity of the attack. The second step has a complexity of 2^{32} since we have to make 2^{32} guesses and we have corresponding key suggestions. Step 3 has a average number of 2^{10} guesses for each of the 2^{32} guesses made at step 2. Thus the 3 steps of the attack procedure has a total complexity of 2^{42}.

We have to repeat this attack for 2^{60} pairs as explained in section 3.2. Hence the total complexity of the attack is 2^{102}.

5.2 Data complexity

Our attack depends on finding the special difference pattern mentioned in section 3.2. The detailed analysis of the same section shows that we need a minimum of 2^{48} encryptions under each of the key K and K^*. Now one AES encryption gives away 40 bytes of key stream. Hence the total number of key stream bytes required is $\left(2^{48} + 2^{48}\right) \times 40 \approx 2^{54.3}$. Equivalently we need $2^{53.3}$ bytes of key streams under each of the keys K and K^*.

Thus the data complexity of our attack is $2^{54.3}$.

6 Results

In this section we shall compare our attack with the existing attacks on `LEX`. We present this comparative study in the form of a table. Including this work there are only 4 reported attacks on `LEX` as mentioned in the introduction of this work.

Attack	Complexity
Sliding attack [2]	2^{61} different IVs , each producing $20,000$ key-stream bytes
Generic distinguishing attack [3]	$2^{65.7}$ resynchronizations
Key recovery attack based on differential propagation in `LEX` [4]	$2^{36.3}$ bytes of key-stream produced by the same key and time complexity of 2^{112}
Related key based key recovery attack	$2^{53.3}$ key stream bytes produced by each of 2 related keys and 2^{102} time complexity.

7 Conclusion

In this paper we have taken a related key approach to cryptanalyze `LEX`. We have observed a five round differential in the key scheduling of `AES`. Combining this differential with a special difference pattern we have observed predictable differential pattern propagation in the key streams generated by LEX and consequently mounted a key recovery attack on `LEX`. In the proposed attack we have used two related keys, $2^{54.3}$ bytes of keystreams and 2^{102} operations to retrieve 16 bytes of the secret key. The observation of this paper can serve as a guideline for developing stream ciphers using the interesting technique of *"leak extraction"*.

References

1. Alex Biryukov, "A New 128-bit Key Stream Cipher LEX," Ecrypt Stream Cipher Project Report, 2005/013, 2005, http://ecrypt.eu.org/stream.
2. H. Wu and B. Preneel, "Attacking the IV setup of the stream cipher LEX," Ecrypt Stream Cipher Project Report, 2005/059, 2005, http://ecrypt.eu.org/stream.
3. Håkan Englund, Martin Hell and Thomas Johansson, "A Note on Distinguishing attacks," in *Preproceedings of State of the Art of Stream Ciphers workshop (SASC 2007)*, 2007, pp. 73–78.

4. Orr Dunkelman and Nathan Keller, "A new attack on the lex stream cipher," in *ASIACRYPT*, 2008, pp. 539–556.
5. Eli Biham, "New types of cryptanalytic attacks using related keys," in *J. Cryptology*, 1994, pp. 229–246.

A Retrieval of subkey bytes using key schedule

We have mentioned in step 3 of the attack procedure described in section 4 that we need to recover some unknown subkey bytes using known subkey bytes evaluated till that stage. For example in one case the attacker retrieves 8 key bytes of the subkey K^r and 4 key bytes of the subkey K^{r+1}. Since the subkey K^{r+1} is derived from the subkey K^r itself using key schedule, hence we can combine the partial knowledge about bytes of K^{r+1}, K^r and the key scheduling algorithm to deduce more bytes of K^{r+1}. We have used this kind of analysis twice in our work. Here we present those cases with detailed equations

1. At the step 3 of the attack procedure described in section 4 we have shown the known bytes of the subkey K^r and K^{r+1} after the guessing phase in *figure 8*. At that point we have retrieved 8 key bytes of K^r and 4 key bytes of K^{r+1}. We show the situation in *figure 10*. The colored bytes are known subkey bytes.

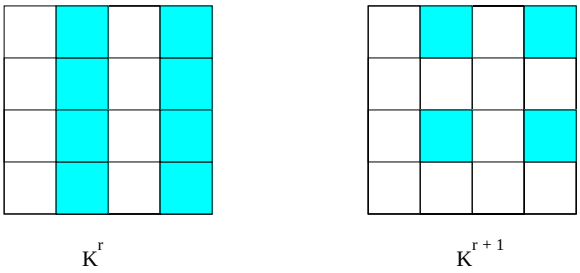

Fig. 10. Known subkey bytes before retrieval

Now we use the following equations to determine 4 more bytes of the round key K^{r+1}. These equations are based on `LEX` key schedule.

$$
\begin{aligned}
K^{r+1}_{0,0} &= K^{r+1}_{0,1} + K^r_{0,1} \\
K^{r+1}_{0,2} &= K^{r+1}_{0,3} + K^r_{0,3} \\
K^{r+1}_{2,0} &= K^{r+1}_{2,1} + K^r_{2,1} \\
K^{r+1}_{2,2} &= K^{r+1}_{2,3} + K^r_{2,3}
\end{aligned}
$$

After using these equations we retrieve four subkey bytes $K^{r+1}_{0,0}, K^{r+1}_{0,2}, K^{r+1}_{2,0}, K^{r+1}_{2,2}$.

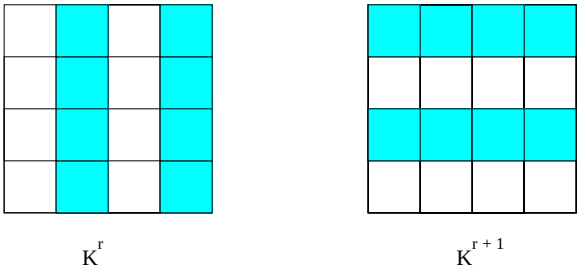

Fig. 11. Known subkey bytes after retrieval

In *figure 11* we have represented these revealed bytes of K^{r+1} by coloring them. These known bytes are used in the calculation of step 3 of section 4, illustrated in *figure 9*.

2. After retrieving the value of z_1, z_2, z_3, z_4 in *figure 9* we shall get 8 key bytes of K^r and 12 key bytes of K^{r+1}. We show the situation in *figure 12*. The colored bytes are known subkey bytes.

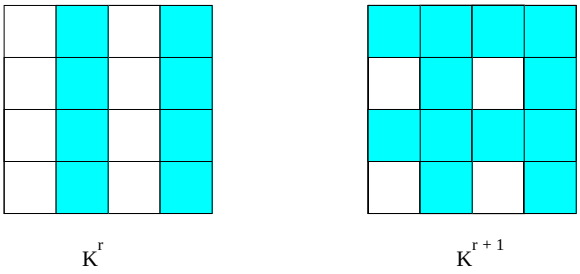

Fig. 12. Known subkey bytes before retrieval

Using the following equations based on `LEX` key schedule we can determine 4 more bytes of the round key K^{r+1} :

$$\begin{aligned} K^{r+1}_{1,0} &= K^{r+1}_{1,1} + K^{r}_{1,1} \\ K^{r+1}_{1,2} &= K^{r+1}_{1,3} + K^{r}_{1,3} \\ K^{r+1}_{3,0} &= K^{r+1}_{3,1} + K^{r}_{3,1} \\ K^{r+1}_{3,2} &= K^{r+1}_{3,3} + K^{r}_{3,3} \end{aligned}$$

Hence using these equations the full 16 bytes of the subkey K^{r+1} is retrieved. These bytes are used in step 3 of section 4.