

Improved Linear Sieving Techniques with Applications to Step-Reduced LED-64

Itai Dinur^{1,*}, Orr Dunkelman^{2,4,**}, Nathan Keller^{3,4,***}, and Adi Shamir⁴

¹ Département d'Informatique, École Normale Supérieure, Paris, France

² Computer Science Department, University of Haifa, Israel

³ Department of Mathematics, Bar-Ilan University, Israel

⁴ Computer Science department, The Weizmann Institute, Rehovot, Israel

Abstract. In this paper, we describe new techniques in meet-in-the-middle attacks. Our basic technique is called a *linear key sieve* since it exploits as filtering conditions linear dependencies between key bits that are guessed from both sides of the attack. This should be contrasted with related previous attacks, which only exploited a *linear state sieve* (i.e., linear dependencies between state bits that are computed from both sides of the attack). We apply these techniques to the lightweight block cipher LED-64, and improve some of the best known attacks on step-reduced variants of this cipher in all attack models. As a first application of the linear key sieve, we describe a chosen plaintext attack on 2-step LED-64, which reduces the time complexity of the best previously published attack on this variant from 2^{56} to 2^{48} . Then, we present the first attack on 2-step LED-64 in the *known plaintext model*. In this attack, we show for the first time that the splice-and-cut technique (which inherently requires chosen messages) can also be applied in the known plaintext model, and we use the linear key sieve in order to obtain an attack with the same time complexity as our chosen plaintext attack. Finally, we describe a related-key attack on 3-step LED-64 which improves the best previously published attack (presented at Asiacrypt 2012) in all the complexity parameters of time/data/memory from 2^{60} to 2^{49} . As our first two single-key attacks, the related-key attack is also based on the linear key sieve, but it uses additional techniques in differential meet-in-the-middle which are interesting in their own right.

Keywords: Cryptanalysis, LED, AES, Even-Mansour, Meet-in-the-middle attack, Splice-and-cut

* Some of the work presented in this paper was done while the first author was a postdoctoral researcher at the Weizmann Institute, Israel.

** The second author was supported in part by the German-Israeli Foundation for Scientific Research and Development through grant No. 2282-2222.6/2011.

*** The third author was supported by the Alon Fellowship.

1 Introduction

Meet-in-the-middle (MITM) attacks on block ciphers were first introduced more than 30 years ago [12]. A block cipher was shown to be vulnerable to such attacks if it is possible to independently compute a variable of its inner state from the encryption and decryption sides without having to guess the full key. The value of this inner variable (or variables) is used to efficiently sieve the key suggestions obtained from both sides and mount an efficient attack. This motivated block cipher designers to incorporate relatively complex key schedule algorithms into the design, thus assuring very quick diffusion of the full key into the state and making the cipher resistant to MITM attacks. On the other hand, complex key schedule algorithms are difficult to implement in resource-constrained environments (such as RFID tags and wireless sensors). Thus, in recent years, with the rise of lightweight cryptography, designers have proposed many schemes with simplified key schedule algorithms. At the extreme end of the scale lie block ciphers such as LED-64 [9], Zorro [8] and PRINCE [3] which have no key schedule at all, and simply XOR the key to the internal state of the cipher several times during the encryption process.

Naturally, the tendency to simplify the key schedule of block ciphers was accompanied by the development of interesting new techniques in MITM attacks in order to break these schemes. Examples of these techniques include sieve-in-the-middle [5] and splice-and-cut [1, 14] which is one of the most notable techniques, initially applied to hash functions, but was quickly shown to be applicable to block ciphers as well. Splice-and-cut attacks are adaptations of Merkle and Hellman's attack on 2K-3DES [12] to single encryption. The main idea is to obtain the encryptions of several chosen plaintexts in order to view the first and the last rounds of the cryptosystem as consecutive rounds. As a result, the adversary can split the cipher into two parts in an unconventional way, and mount an efficient MITM attack in cases where such an attack seems difficult otherwise.

Another important technique used in several MITM attacks exploits the ability to independently and efficiently compute linear combinations of variables of the inner state (rather than the actual variables) of the cipher from the encryption and decryption sides. This linear state sieving technique allows the adversary to apply a MITM attack through a linear layer of the block cipher, and was specifically exploited in [4, 10, 13] in order to mount such attacks through the linear MixColumns operation of AES-based constructions.

In this paper, we extend the linear sieving technique and show how to obtain additional filtering conditions by exploiting linear dependencies in the key suggestions that are computed from both sides in the MITM attack, which we call a *linear key sieve*. Although we do not expect such linear dependencies to exist in block ciphers with complex non-linear key schedules, they are much more likely to occur in lightweight designs with simple key schedules. To demonstrate the power of our linear key sieve techniques, we apply them to LED-64 [9], and use them to improve some of the best previously known attacks on step-reduced variants of this block cipher in all attack models.

The lightweight block cipher LED was presented at CHES 2011 [9], and due to its elegant AES-based design, it has been the target of significant cryptanalytic effort in the past few years. In the single-key chosen plaintext model, the best previously known attack on 2 steps of LED-64 (reduced from the full 8) was presented in [10], and we reduce its time complexity from 2^{56} to 2^{48} . Both the previous attack and our new attack apply the splice-and-cut technique in order to mount a MITM attack on the cipher. The main element that enables us to improve the previous attack of [10] is the linear key sieve, which we use in order to filter the key suggestions obtained during the attack in a more efficient way.

In addition to the chosen plaintext attack, we present the first attack on 2-step LED-64 in the known plaintext model. The main novelty of this attack is that it uses, for the first time, the splice-and-cut technique (which seems to require chosen messages in an inherent way) in the known plaintext model. Once again, in this attack we use the linear key sieve technique, and it enables our known plaintext attack to maintain the same running time as our chosen plaintext attack.

Finally, in the stronger related-key model, we analyze 3-step LED-64, on which the best previously known attack [11] used a classical differential method. In this model, we use a differential MITM attack (which is interesting in its own right), in addition to the linear key sieve, in order to mount a more efficient attack which improves the previous attack in all the complexity parameters of time/memory/data from 2^{60} to 2^{49} .

The paper is organized as follows: in Section 2, we describe LED-64 and summarize the previous attacks as well as our new attacks on this cipher. In Section 3, we describe the notations and conventions that are used in this paper. Our new chosen plaintext, known plaintext and related-key attacks are described in Sections 4, 5 and 6, respectively. Finally, we conclude in Section 7.

2 Description of LED-64

LED [9] is a 64-bit block cipher built using several public permutations, interleaved with round-key additions over $GF(2)$ (i.e., XOR operations). This construction is generally known as iterated Even-Mansour (see Figure 1), which generalizes the original one-round construction [7]. In the case of LED, the public permutations are called steps, and each step is composed of 4 rounds. A round of LED uses an AES-like design, where given a 64-bit input X , it is treated as a concatenation of 16 four-bit nibbles $X[0]||X[1]||\dots||X[15]$, which are (conceptually) arranged in a 4×4 array:

$X[0]$	$X[1]$	$X[2]$	$X[3]$
$X[4]$	$X[5]$	$X[6]$	$X[7]$
$X[8]$	$X[9]$	$X[10]$	$X[11]$
$X[12]$	$X[13]$	$X[14]$	$X[15]$

The round function uses 4 AES-like mappings AddConstants (AC), SubCells (SC), ShiftRows (SR), and MixColumnsSerial (MCS). The structural proper-

ties of these mappings (given below) are similar to those of the AES mappings AddRoundKey, SubBytes, ShiftRows and MixColumns, respectively, and these are the only properties which are exploited by our attacks. For the complete implementation details of the LED mappings, refer to [9].

1. AddConstants adds (over $\text{GF}(2)$) a round-dependant constant to each cell of the two first columns.
2. SubCells applies a 4-bit Sbox to every cell of the internal state.
3. ShiftRows rotates each cell located in row i by i positions to the left.
4. MixColumnsSerial independently applies an MDS (Maximum Distance Separable) matrix to each column.

LED has two main variants, LED-64 and LED-128, which differ according to the key size. In this paper, we are mainly interested in the 64-bit version, which uses 32 rounds (or 8 steps). The key schedule of LED-64 simply adds the 64-bit key K before rounds $4i + 1$ for $i = 0, 1, \dots, 7$, and after the final round 32.

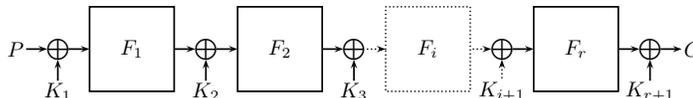


Fig. 1. Iterated Even-Mansour

2.1 Summary of Our New Attacks and the Previous Attacks on Step-Reduced LED-64

Our new attacks on step-reduced LED-64 are summarized in table 1. In the single-key model, the previous best chosen plaintext attack on 2-step LED-64 is described in [10], and we improve its time complexity from 2^{56} to 2^{48} . In addition, we present the first attack on 2-step LED-64 in the known plaintext model, which has the same time complexity as our best chosen plaintext attack (but requires more data and memory). We also mention the attack on 3-step LED-64 given in [6]. However, despite its theoretical significance, the attack is non-practical with respect to the memory complexity (which is 2^{60}) and it is only about 16 times faster than exhaustive search. On the other hand, our attacks on 2-step LED-64 are more than 250,000 times faster than exhaustive search and are more practical.

In the related-key model, the best attack on 3-step LED-64 is described in [11], and we simultaneously improve it in all the complexity parameters of time/memory/data from 2^{60} to 2^{49} . We also mention the related-key attack on 4-step LED-64 given in [11]. However, although the attack may be theoretically

interesting, it is very marginal with respect to the time, memory and data complexities, which are all¹ about 2^{63} .

Section	Model	Steps	Time	Data	Memory
4.2	Single-key	2	2^{48}	2^{16} CP	2^{17}
5	Single-key	2	2^{48}	2^{48} KP	2^{48}
6.2	Related-Key	3	2^{49}	2^{49} CP	2^{49}

The data complexity is given in chosen plaintexts (CP), or in known plaintexts (KP).

Table 1. Our New Attacks of Step-Reduced LED-64

3 Notations and Conventions

Notations We denote by R_r the public function of round r of LED-64 (without the key addition), i.e., given a 64-bit state X , $R_r(X) \triangleq MCS(SR(SC(AC(X))))$. We denote by F_i the public function of step i , i.e., given a 64-bit state X , $F_i(X) \triangleq R_{4i+4}(R_{4i+3}(R_{4i+2}(R_{4i+1}(X))))$. The functions R_r^{-1} and F_i^{-1} are defined as the inverses of R_r and F_i , respectively.

Given a plaintext-ciphertext pair (P, C) , we denote the state after r encryption rounds by X_r (e.g., $X_0 = P$ and X_1 is the state after one round of LED-64). In order to simplify our notation, we define $\hat{X}_{4i} = X_{4i} \oplus K$, and so $F_i(\hat{X}_{4i}) = X_{4(i+1)}$. In some of our attacks, in addition to obtaining plaintext-ciphertext pairs, we independently evaluate the public step function F_i (for some i) on some input states \hat{Y}_{4i} , and we define $F_i(\hat{Y}_{4i}) = Y_{4(i+1)}$ (i.e., $\hat{Y}_{4i} = Y_{4i} \oplus K_i$).

We denote the j 'th column of X_i by $X_{i,|j|}$, i.e., $X_{i,|0|}$ is composed of nibbles $\{0, 4, 8, 12\}$. Similarly, we denote by $X_{i,|j,l|}$ columns j and l of X_i . We define two more column-related sets: the first is $X_{i,/j/}$ which is composed of the nibbles in X_i corresponding to the places after the ShiftRows operation on column j , e.g., $X_{i,/0/}$ is composed of nibbles $0, 7, 10, 13$. The second set is $X_{i,\setminus j\setminus}$ which is composed of the nibbles in the positions of column j after having applied the inverse ShiftRows operation.

Conventions Throughout this paper, we use the standard conventions and calculate the time complexity of our attacks in terms of evaluations of the full cipher, while calculating their memory complexity in terms of 64-bit words (since the block size of LED-64 is 64 bits). Some of the attacks presented in this paper

¹ In fact, since the time complexity of exhaustive search given 2 related keys is 2^{63} , the attack of [11] in its current form is faster than exhaustive search by a factor which is less than 2.

involve basic linear algebra algorithm (such as solving a system of linear equations with a few dozen variables² over $GF(2)$). Since our attacks execute these basic linear algebra algorithms no more than a few times per evaluation of the full cipher, we can ignore them in our time complexity analysis.

4 An Improved Chosen Plaintext Single-Key Attack on 2-Step LED-64

In this section, we introduce the linear key sieve technique, and apply it (combined with splice-and-cut) to 2-step LED-64. Our attack improves the time complexity of the previously best known attack on 2-step LED-64 [10] (which is based on the techniques of [13]) from 2^{56} to 2^{48} .

To simplify the description of the technique, before presenting the full 2-step attack, we introduce the linear key sieve in a simple example of a basic MITM attack on 1-step LED-64.

4.1 A Meet-in-the-Middle Attack on 1-Step LED-64

We describe a basic MITM attack on 4-round (1-step) LED-64, using a single known plaintext-ciphertext pair ($P = X_0, C = \hat{X}_4$). The attack is based on a few simple and well-known observations on AES-based constructions:

1. The order of the linear operations ARK and MCS is interchangeable, i.e., $MCS^{-1}(ARK^{-1}(C)) = ARK'^{-1}(MCS^{-1}(C))$, where ARK' adds the key $K' \triangleq MCS^{-1}(K)$ to the state. As in many attacks on AES-based constructions, we can thus apply MCS^{-1} to the ciphertext C , and “peel-off” the last-round MCS operation.
2. Given an inverse-shifted column $X_{r,\setminus i\setminus}$ (at the beginning of any round r), we can fully compute the shifted column of the state after the first 7 operations $SR(SC(AC(MCS(SR(SC(AC(X_r)))))))/_{i/}$. Similarly, given a shifted column $X_{r,/i/}$, we can fully compute the inverse-shifted column of the state after the 7 inverse operations $AC^{-1}(SC^{-1}(SR^{-1}(MCS^{-1}(AC^{-1}(SC^{-1}(SR^{-1}(X_r)))))))/_{i\setminus}$. Such a permutation, mapping 4 nibbles to 4 nibbles of the state through a “round and a half” is called a “Super-Sbox” of LED.
3. Given knowledge of any b_1 bits of the state X , we can compute the values of b_1 linear expressions (over $GF(2)$) on the state $MCS(X)$.

Observation 2 implies that given $\hat{X}_{0,\setminus i,j\setminus}$ (for any two columns i and j) we can compute $SR(SC(AC(X_1)))/_{i,j/} = MCS^{-1}(X_2)/_{i,j/}$, as shown by the gray nibbles in Figure 2. Combined with Observation 3, we can compute $16 \cdot 2 = 32$ linear expressions on the state X_2 , spanning a subspace of dimension 32.

² In order to reduce the $O(n^3)$ bit operations required to solve a system of n linear equations $Ax = b$ in the online phase of the attack, we compute A^{-1} offline. Given the vector b in the online phase, we simply compute $x = A^{-1}b$ in $O(n^2)$ bit operations.

Similarly, we can apply Observation 2 from the decryption side (with three columns), i.e., $MCS^{-1}(ARK^{-1}(C))_{/l,m,n/}$ gives us the knowledge of 48 bits of X_2 (namely, $X_{2,\setminus l,m,n\setminus}$) as shown by the striped nibbles in Figure 2.

Since the full state contains 64 bits, the intersection of these two subspaces is a linear subspace of dimension³ $32+48-64 = 16$. The basis of this subspace gives rise to 16 linearly independent expressions in the bits of the state X_2 (denoted by A , as shown in Figure 2) whose values are computable independently from the known 2 inverse-shifted columns (from the encryption side), and from the known 3 shifted columns (from the decryption side).

The computation of the $|A| = 16$ joint expressions, requires the knowledge of $16 \cdot 2 = 32$ bits of K from one side and $16 \cdot 3 = 48$ bits of $K' = MCS^{-1}(K)$ from the other side. For the correct guess of these bits, the values of the joint expressions in A match with probability 1, whereas for an arbitrary incorrect suggestion, based on standard randomness assumptions, the values of the expressions in A match with probability $2^{-|A|} = 2^{-16}$. Therefore, incorrect suggestions in the MITM attack are discarded (as in the related attacks of [4, 10, 13]), and we refer to the expressions in A as a *linear state sieve*.

In standard MITM attacks, one treats the keys K and K' as independent, and thus there are $2^{32+48} = 2^{80}$ suggestions for the key from both sides. Given the 16 bits of the linear state sieve, we expect about 2^{64} suggestions to remain, which we need to further analyze. Thus, in its current form, this attack is not faster than exhaustive search.

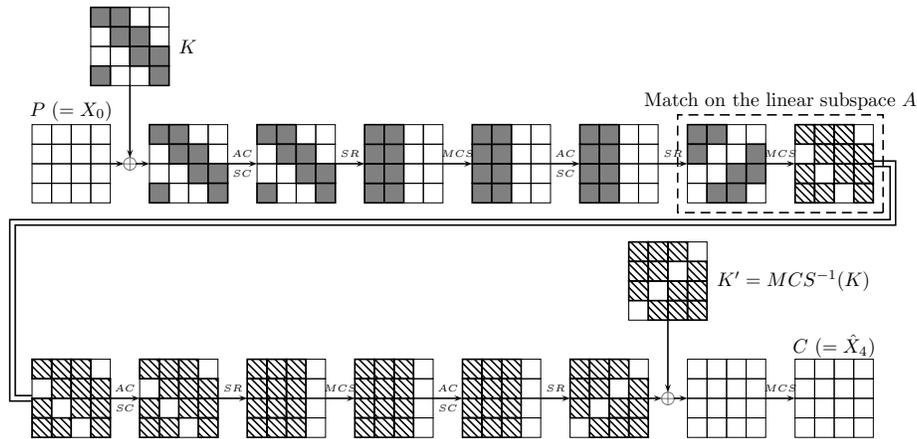


Fig. 2. A Meet-in-the-Middle Attack on 1-Step LED-64

³ In general, the dimension of the intersection can be bigger. However, in AES-based constructions (where the MixColumns operation is implemented using an MDS matrix), the dimension of the intersection is exactly $32 + 48 - 64 = 16$.

The Linear Key Sieve A simple solution to the problem of insufficient filtering is to use an additional plaintext-ciphertext pair, which will offer an additional 16-bit filtering condition. However, we now introduce the *linear key sieve* which provides these 16 bit-conditions with *no additional data*, by exploiting the linear dependency of K and K' . This novel observation (which [10] did not use) is at the basis of our improved attack on 2-step LED-64 (described in the next section), and all the other attacks presented in the paper.

Recall that the MITM attack requires 32 bits of K and 48 bits of $K' = MCS^{-1}(K)$, which are linear expressions in the bits of K . Just as the state subspaces intersect (and allow us to obtain the linear state sieve), so do the two linear subspaces spanned by the expressions of K and K' that we guess. The intersection is a linear subspace of dimension $32 + 48 - 64 = 16$, giving rise to 16 linearly independent expressions in the bits of K (denoted by B_3), whose values are computable independently from both sides. The expressions in B_3 are used in order to filter our wrong key guesses (for the right key they agree with probability 1, and for wrong key guesses they agree with probability $2^{-|B_3|}$), and thus we call this set of expressions a *linear key sieve*.

Let B_1 be additional 16 linear expressions of K needed for the attack⁴ (i.e., B_1 and B_3 determine the partial encryption of two inverse-shifted columns), and let B_2 be additional 32 linear expressions of K' needed for the attack (i.e., B_2 and B_3 determine the partial decryption of three shifted columns). Our MITM attack is composed of an outer loop, iterating over the value of B_3 , where in each iteration, we independently iterate over the values of B_1 and B_2 .⁵ Thus, we *force* the key suggestions obtained from both sides of the attack to agree on B_3 (rather than randomly achieving agreement). The resultant attack (described for two arbitrarily columns i, j in the forward direction and three arbitrary columns l, m, n in the backward direction) is as follows:

1. For each value of the 16 expressions of B_3 :
 - (a) For each value of the 16 expressions of B_1 :
 - i. Compute $K_{\setminus i, j \setminus}$, and use it to compute $\hat{X}_{0, \setminus i, j \setminus}$.
 - ii. Compute the values of the 16 expressions of A , and store them in a sorted list L , next to the value of the 16 expressions of B_1 .
 - (b) For each value of the 32 expressions of B_2 :
 - i. Compute $K'_{/l, m, n /}$, and use it to compute $MCS^{-1}(ARK^{-1}(C))_{/l, m, n /}$.
 - ii. Compute the values of the 16 expressions of A , and search for matches in the list L .
 - iii. For each match:
 - A. Obtain the value of the 16 expressions of B_1 .
 - B. Compute K using linear algebra, given the values of B_1 , B_2 and B_3 .

⁴ There are many options for the basis B_1 , and we choose one arbitrarily.

⁵ We note that the approach of taking out shared bits to an outer loop is a very common practice in saving memory. The main novelty in this attack is the fact that we take out shared *linear expressions*.

C. Test K using a trial encryption, and if it succeeds, return the key.

The list L contains 2^{16} values, and thus we expect a single match for each value of the 16 expressions of A in Step 1.(b).ii. This implies that the expected time complexity of each iteration of Step 1 is about 2^{32} , and thus the expected time complexity of the whole attack is 2^{48} , which is faster than exhaustive search by a factor of 2^{16} . The memory complexity of the attack is about 2^{16} , which is required in order to store the list L . Note that the memory needed for storing L in each iteration of Step 1 can be reused.

4.2 The Improved Chosen Plaintext Single-Key Attack on 2-Step LED-64

Our attack on 2-Step LED-64 follows the same general structure as the previous one of [10, 13]. We use the splice-and-cut technique in combination with a MITM attack on 4 rounds (1 step) of the cipher. The advantage of our attack comes from the linear key sieve (missing from [10]), i.e., using the linear relations between K and $K' = MCS^{-1}(K)$.

In order to apply splice-and-cut to 2-step LED-64, we (as in the previous attack [10]) partition the indices of 64-bit state into two lexicographically ordered sets, S_1 and S_2 . The attack requires the encryptions of $2^{|S_1|}$ plaintexts P^1, P^2, \dots in which all the bits of S_2 are fixed to zero (or any arbitrary constant), and the bits of S_1 range over all the possible values. Independently, we evaluate the first 4 key-less rounds of LED-64 (i.e., F_1) on $2^{|S_2|}$ inputs $\hat{Y}_0^1, \hat{Y}_0^2, \dots$ in which the bits of S_1 are fixed to zero (or any other constant), and the bits of S_2 range over all the possible values, and obtain the corresponding outputs Y_4^1, Y_4^2, \dots .

In the splice-and-cut technique, we look for a plaintext $P^i = X_0^i$ and an internal state \hat{Y}_0^j such that $\hat{X}_0^i = X_0^i \oplus K = \hat{Y}_0^j$. This occurs if and only if P^i and K “agree” on the bits of S_1 and \hat{Y}_0^j and K “agree” on the bits of S_2 , or formally $P^i|_{S_1} = K|_{S_1}$ and $\hat{Y}_0^j|_{S_2} = K|_{S_2}$ (where $W|_S$ denotes the $|S|$ -bit value of the word W on the indices of the ordered set S). In other words, each plaintext P^i is associated with a potential value of $K|_{S_1}$, and each state \hat{Y}_0^j is associated with a potential value of $K|_{S_2}$. Using splice-and-cut, there is only one such correct pair, and we denote its plaintext by P , and its evaluated state by $\hat{Y}_0 = P \oplus K$. Thus, the knowledge of i and j is equivalent to the knowledge of the key K .

Consider the correct pair (P, \hat{Y}_0) . Applying F_1 to \hat{Y}_0 gives $Y_4 = X_4$. Thus, if we consider all $F_1(\hat{Y}_0^j) = Y_4^j$ values, one of them is indeed X_4 . As a result, given the ciphertext C that corresponds to P and X_4 , the splice-and-cut technique reduces the problem to attacking 4 rounds of LED-64. Hence, as shown in Figure 3, we choose the bits of S_2 to be the 48 bits of 3 inverse-shifted columns (and thus S_1 contains the 16 bits of the remaining inverse-shifted column). As a result, we can take each Y_4^j value (associated with a suggestion for 3 inverse-shifted columns of K) and continue its partial encryption (as per Observation 2), resulting in the knowledge of three shifted columns before the MCS operation of

round 6. This knowledge gives rise to suggestions for the values of 48 linear expressions of Y_6 . Independently, we try all possible values of 32 bits of two shifted columns of K' , and partially decrypt all the ciphertexts to obtain suggestions of 32 bits of X_6 .

Now, we can apply the 4-round attack (as $Y_6 = X_6$), obtaining a linear state sieve of $48 + 32 - 64 = 16$ linear expressions on the state bits (each independently computed from a different side), denoted by A . To obtain more filtering conditions, we again use the linear key sieve: each ciphertext is associated with 16 bits of K , and thus each suggestion for the 32 bits of X_6 is associated with the values of $16 + 32 = 48$ linear expressions on the bits of K . Since each suggestions for Y_6 depends on 48 bits of K , we have $48 + 48 - 64 = 32$ expressions which we use as a linear key sieve (denoted by B_3). Similarly to our basic MITM attack, we complement the 32 linear expressions of B_3 to a basis of the subspace spanned by the 48 bits of S_2 using 16 additional expressions, denoted by B_2 . Similarly, we complement B_3 to a basis of the 48-dimensional subspace spanned by the 16 bits of S_1 and the 32 bits of the 2 shifted columns of K' , using 16 additional expressions, denoted by B_1 .

The attack proceeds as follows (see figure 3):

1. Request the encryptions of the 2^{16} plaintexts P^i such that $P_{\setminus 0,1,2}^i = 0$, and store all plaintext-ciphertext pairs.
2. For each value of the 32 linear expressions of B_3 :
 - (a) For each value of the 16 linear expressions of B_2 :
 - i. Using the values of B_2 and B_3 , compute a suggestion for $K_{\setminus 0,1,2}$.
 - ii. Let \hat{Y}_0^j be the state such that $Y_{0,\setminus 3}^j = 0$ and $Y_{0,\setminus 0,1,2}^j = K_{\setminus 0,1,2}$.
 - iii. Compute $F_1(\hat{Y}_0^j) = Y_4^j$, and use the partial knowledge of $K_{\setminus 0,1,2}$ to obtain a suggestion for the values of the 16 linear expressions of A . Store the suggestion for the values of A in a sorted list L , next to the value of B_2 .
 - (b) For each value of the 16 linear expressions of B_1 :
 - i. Using the values of B_1 and B_3 , compute a suggestion for $K_{\setminus 3}$ and $K'_{/0,1}$.
 - ii. Let (P^i, C^i) be the plaintext-ciphertext pair such that $P_{\setminus 3}^i = K_{\setminus 3}$ (recall that $P_{\setminus 0,1,2}^i = 0$).
 - iii. Compute a suggestion for the values of the 16 linear expressions of A using C^i and $K'_{/0,1}$.
 - iv. Search for the suggestion for the values of A in the list L .
 - v. For each match, obtain the value of B_2 , use it to obtain a suggestion for the key K and test it using a trial encryption. If the trial succeeds, return the key.

The data complexity of the attack is 2^{16} chosen plaintexts. The memory complexity of the attack is about 2^{17} , required in order to store the plaintext-ciphertext pairs, and in order to store the list L . Since L contains 2^{16} values per iteration of Step 2.(a), we expect one match in Step 2.(b).iv, and a total of 2^{16} matches per iteration of Step 2. Thus, the time complexity of an iteration of

Step 2 is equivalent to about 2^{16} 2-step LED-64 encryptions, and the total time complexity of the attack is about 2^{48} encryptions.

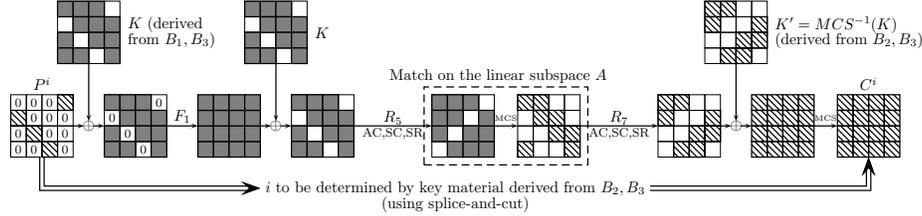


Fig. 3. Our Improved Chosen Plaintext Attack on 2-Step LED-64

5 A Known Plaintext Single-Key Attack on 2-Step LED-64

In this section, we describe the first splice-and-cut attack (which was believed to be inherently a chosen message technique) in the known plaintext model. In particular, we devise the first known plaintext attack on 2-step LED-64. Due to the efficient sieving techniques, this attack has the same time complexity as our chosen plaintext attack, presented in the previous section.

One could expect the transformation of the splice-and-cut attack to the known plaintext model to be as follows: similarly to the chosen plaintext attack, collect $2^{|S_2|}$ inputs $\hat{Y}_0^1, \hat{Y}_0^2, \dots$ such that $\hat{Y}_0^i|_{S_1} = 0$ and $\hat{Y}_0^i|_{S_2}$ range over all the possible values. Then, find a plaintext P^i such that $P^i \oplus K = \hat{Y}_0^j$ for some (i, j) , and repeat the same attack procedure. However, in the known plaintext model, $P|_{S_2}$ is not fixed, and thus we “lose” the association of an input \hat{Y}_0^j to a potential value of $K|_{S_2}$. Consequently, it is not clear how to continue the (partial) evaluation of \hat{Y}_0^j beyond the 4th (keyed) round of LED in order to obtain the suggestions for the values of the linear expression on Y_6 . Thus, for each \hat{Y}_0^j , we need to guess additional key bits in order to obtain these necessary filtering conditions.

Since an input \hat{Y}_0^j is now associated with many key guesses, we are forced to evaluate fewer such inputs in order to obtain an efficient attack. Indeed, while in the chosen plaintext attack, we evaluated 2^{48} such inputs, here we evaluate only 2^{16} inputs, and for each one we guess the 32 bits of $K_{\setminus\{0,1\}}$ and obtain a suggestion for the values of 32 linear expressions on Y_6 . According to the birthday paradox (assuming that the plaintexts are uniformly distributed⁶), in order to

⁶ In case that the plaintexts are not uniformly distributed, we can apply a similar attack assuming that the ciphertexts are uniformly distributed (by exchanging the roles of encryption and decryption).

obtain a pair (P^i, \hat{Y}_0^j) such that $P^i \oplus K = \hat{Y}_0^j$ with good probability, we need 2^{48} known plaintexts-ciphertext pairs (P^i, C^i) ($i \in \{1, 2, \dots, 2^{48}\}$). The values $\hat{Y}_0^1, \hat{Y}_0^2, \dots$ which we evaluate are defined by the 16-dimensional linear subspace $\{\hat{Y}_0^j | MCS^{-1}(\hat{Y}_0^j)_{/1,2,3/} = 0\}$. Thus, a plaintext P^i is implicitly associated with a partial key value $P^i_{/1,2,3/} = K'_{/1,2,3/}$. This implies that we can partially decrypt C^i in order to obtain the values of 48 bits of X_6^i without additional key guesses. Note that this is not the traditional way in which splice-and-cut is applied, as all previous attacks (including our previous chosen plaintext attack) directly partitioned the bits of the state into two groups S_1 and S_2 . Instead, in this attack we work with linear subspaces constructed to exploit the linear dependency between K and K' in order to be able to partially decrypt C^i without additional key guesses (whereas our previous chosen plaintext attack did not directly exploit this dependency).

As in the chosen plaintext attack, we have a linear state sieve of $32+48-64 = 16$ expressions on the bits of $X_6 = Y_6$ which are independently computable from each side, and we denote it by A_1 . Each value of the expressions of A_1 , computed from an input \hat{Y}_0^j , is associated with a suggestion for $K_{\setminus 0,1 \setminus}$, and as a result, we can also compute $Y_{0,\setminus 0,1 \setminus}^j = \hat{Y}_{0,\setminus 0,1 \setminus}^j \oplus K_{\setminus 0,1 \setminus}$. Since $P^i = X_6^i$, the 32-bit value of $Y_{0,\setminus 0,1 \setminus}^j$ can be directly matched with each plaintext, and we denote this sieve by A_2 . We note that since $MCS^{-1}(\hat{Y}_0^j)_{/1,2,3/} = 0$, then Y^j (and $Y_{0,\setminus 0,1 \setminus}^j$), can only attain 2^{16} values, and thus effectively, the 32 bits of A_2 give only 16 bits of filtering.

From the decryption side, each value computed from (P^i, C^i) is associated with a suggestion for $K'_{/1,2,3/}$. Thus, we can identify $32 + 48 - 64 = 16$ linear expressions (i.e., a linear key sieve) which are independently computable from each side, and we denote this sieve by B . In total, we have 48 bits of filtering, as A_1 gives us 16 bits, A_2 (effectively) gives us 16 bits of filtering, and B gives us additional 16 bits.

The attack proceeds as follows:

1. For each of the 2^{16} possible values of \hat{Y}_0^j such that $MCS^{-1}(\hat{Y}_0^j)_{/1,2,3/} = 0$:
 - (a) Compute $F_1(\hat{Y}_0^j) = Y_4^j$
 - (b) For each of the 2^{32} values of $K_{\setminus 0,1 \setminus}$:
 - i. Compute the values of the 16 linear expressions of A_1 , the values of the bits of A_2 , and the values of the 16 linear expressions of B . Store these values in a sorted list L , next to the values of $K_{\setminus 0,1 \setminus}$.
2. For each plaintext-ciphertext pair (P^i, C^i) :
 - (a) Assume that $K'_{/1,2,3/} = MCS^{-1}(P^i)_{/1,2,3/}$, compute the values of A_1 , A_2 and B , and search the list L for matches.
 - (b) For each match, obtain $K_{\setminus 0,1 \setminus}$, compute a suggestion for the full key K , and test it using a trial encryption. If the trial succeeds return the key.

Since we evaluate 2^{16} inputs \hat{Y}_0^j , we expect that after obtaining the encryptions of about 2^{48} arbitrary plaintexts, we will have a pair (P^i, \hat{Y}_0^j) satisfying

$P^i \oplus K = \hat{Y}_0^j$, which will enable us to recover the correct key.⁷ Thus, the expected data complexity of the attack is 2^{48} known plaintexts. Since the size of L is 2^{48} , and (effectively) we have 48 bits of filtering conditions, we expect one match for each plaintext in Step 2.(a), and thus the time and memory complexities of the attack are 2^{48} as well.

6 An Improved Related-Key Attack on 3-Step LED-64

In this section, we describe a related-key attack on 3-step LED-64 using two related keys. The attack improves the previously best known attack on this scheme, described in [11], in all the complexities parameters of time/memory/data from 2^{60} to 2^{49} . The 3-step attack uses the linear key sieve technique on top of a rather involved differential MITM attack. Before describing the full attack, we describe a simple differential MITM attack on 1-step LED-64 in the single-key model, which serves as background to our 3-step related-key attack. We note that in the case of 1-step LED-64, our simple attack is closely related to the attack on Pelican-MAC described in [4].

6.1 A Differential Single-Key Meet-in-the-Middle Attack on 1-Step LED-64

The simple differential MITM attack on 4-round (1-step) LED-64 requires 2 chosen plaintexts, and its memory and time complexities are slightly more than 2^{16} . In order to obtain an efficient attack, we compute and use the difference distribution tables for the LED Super-Sboxes (spanning the third round and part of the fourth round). Namely, given an entry $[\delta_{in}, \delta_{out}]$, specifying a 16-bit input/output difference to the Super-Sbox, the table stores the actual pairs of values that conform to this entry. A single full table can be easily computed during preprocessing in 2^{32} simple operations, and it requires about 2^{32} words of memory. However, in this simple attack, the output difference δ_{out} to each Super-Sbox is fixed by the ciphertexts, and thus we only need a single column in each table. Such a column is computed in the online phase (after obtaining the encryptions of the plaintexts) in 2^{16} time, using 2^{16} storage.

The details of the attack are given below.

1. Obtain the encryptions of P^1 and P^2 , chosen such that $(P^1 \oplus P^2)_{\setminus 0,1,2} = 0$. Denote $\Delta_r = X_r^1 \oplus X_r^2$, i.e., Δ_r is the 64-bit state difference after round r .
2. Compute 3 columns in the difference distribution tables of the LED Super-Sboxes, corresponding to the output differences specified by the three shifted columns $MCS^{-1}(\Delta_4)_{/1,2,3}$.
3. For each value of $K'_{/0}$:

⁷ We note that unlike our chosen plaintext attack (that succeeds in finding the key with probability 100%), our known plaintext attack succeeds with probability of about 63%, which is the probability suggested by the birthday paradox (given 2^{48} known plaintext-ciphertext pairs).

- (a) Use C^1 to compute $X_{2,\setminus 0\setminus}^1$ and C^2 to compute $X_{2,\setminus 0\setminus}^2$, and calculate $\Delta_{2,\setminus 0\setminus}$.
- (b) Given $\Delta_{2,\setminus 0\setminus}$, and the fact that $MCS^{-1}(\Delta_2)_{/0,1,2/} = 0$, calculate the full Δ_2 by solving a system of linear equations.
- (c) Given the input difference Δ_2 and $MCS^{-1}(\Delta_4)$, use the Super-Sbox (partial) difference distribution tables to obtain the possible values for $ARK'(MCS^{-1}(C^1))_{/1,2,3/}$, and use these values to obtain suggestions for the full K' , thus obtaining suggestions for K .
- (d) Test each suggestion for K using a trial encryption, and if it succeeds return the key.

Since we expect, on average, a single suggestion for K in Step 3.(c), the time complexity of Step 3 is about 2^{16} , which is the time complexity of the full attack. The memory complexity is about 2^{16} , required in order to store the columns of the difference distribution tables for the Super-Sboxes.

We note that this attack is faster than the attack of Section 4.1 since the collision on the shifted columns of P allows us to obtain a suggestion for the full key after guessing only 16 bits (enabling us to compute the full state difference Δ_2). This observation will be further exploited in the next section.

6.2 The Improved Related-Key Attack on 3-Step LED-64

In this section, we describe the details of our related-key attack on 12-round (3-step) LED-64 which assumes that we can obtain the encryptions of plaintexts with keys K_1 and K_2 , such that $K_1 \oplus K_2 = \Delta$ is known (in fact, as explained below, we need the ability to partially choose the value of Δ). During the online phase of the attack, we request the encryptions of 2^{48} chosen plaintexts encrypted with K_1 and 2^{48} (different) chosen plaintexts encrypted with K_2 . The time and memory complexities of the attack are about 2^{49} .

Our attack uses the basic framework of [11] for related-key attacks on iterated Even-Mansour schemes. Namely, we ask for the encryptions of pairs of plaintexts $P^{i,1}$ and $P^{i,2} = P^{i,1} \oplus \Delta$, encrypted with K_1 and K_2 , respectively. Considering the encryption process of these two plaintexts, the input difference to the public F_1 function is zero, which implies that the output difference of F_1 is zero, and after the second key addition, the input difference to F_2 is $\hat{\Delta}_4 = \Delta$ (namely, $\hat{X}_4^{i,1} \oplus \hat{X}_4^{i,2} = \hat{\Delta}_4 = \Delta$). At this point, our algorithm diverges from [11] (which assumes that the function F_2 has some high-probability differential characteristic).

Our attack is based on the 4-round differential MITM attack of the previous section. Here, we apply a similar attack to F_3 by processing plaintext pairs whose ciphertexts collide on a shifted column (before the MCS operation). However, unlike the 4-round attack, we do not know the input difference to the public function on which we perform the MITM attack (F_3 in this case). Thus, we preprocess F_2 by computing and storing pairs of inputs to this function with an input difference of Δ , and we use a birthday argument to claim that one of the input pairs will collide with a plaintext pair with high probability. However, this

is insufficient, as we were not able to find parameters for which the differential MITM algorithm yields an efficient attack, under the constraint that we store sufficiently many pairs of inputs to F_2 (in order to obtain a collision with a processed plaintext pair). The problem is that we need to guess too many key bits before we can compute filtering conditions and eliminate some key guesses.

In order to reduce the number of key guesses, we again exploit collisions on (inverse) shifted columns, but this time at the input of F_3 . Namely, we require that the difference at the output of F_2 on some (inverse) shifted columns cancels out after the key addition. More specifically, we preprocess F_2 , and find 2^{31} pairs of inputs to this function, $(\hat{Y}_4^{j,1}$ and $\hat{Y}_4^{j,2} = \hat{Y}_4^{j,1} \oplus \Delta$), such that their output difference is equal to Δ in two inverse-shifted columns (i.e., $(Y_8^{j,1} \oplus Y_8^{j,2})_{\setminus 0,1\setminus} = \Delta_{\setminus 0,1\setminus}$, implying that $(\hat{Y}_8^{j,1} \oplus \hat{Y}_8^{j,2})_{\setminus 0,1\setminus} = 0$). We expect that 2^{31} such pairs indeed exist, since there are 2^{63} unordered input pairs to F_1 with an input difference of Δ , and based on standard randomness assumptions, about 2^{31} of them satisfy the 32-bit condition on the output difference (a slightly smaller number will only slightly increase the complexity of the attack). The trivial algorithm to find these pairs is to exhaustively enumerate all the 2^{63} input pairs, however, this is wasteful as it requires 2^{64} time (and our model does not allow free precomputation). Instead, we use yet again a MITM approach, and devise an auxiliary preprocessing algorithm that finds the required pairs in about 2^{48} time. In order to run efficiently, our algorithm requires that 48 specific bits of Δ are zero, and thus we have to assume that we can partially choose the key difference⁸ Δ . The details of this preprocessing step are specified in Appendix A.

The full preprocessing algorithm (which calls the algorithm of Appendix A) is given below.⁹ In this attack, we assume that we have computed during preprocessing the full difference distribution tables for the LED Super-Sboxes, using about 2^{32} simple operations and 2^{32} memory.

1. Use the auxiliary preprocessing algorithm of Appendix A to obtain 2^{31} pairs $(\hat{Y}_4^{j,1}, \hat{Y}_4^{j,2} = \hat{Y}_4^{j,1} \oplus \Delta)$, such that $(Y_8^{j,1} \oplus Y_8^{j,2})_{\setminus 0,1\setminus} = \Delta_{\setminus 0,1\setminus}$.
2. For each of the 2^{31} pairs $(\hat{Y}_4^{j,1}, \hat{Y}_4^{j,2})$:
 - (a) For each value of $(K_1)_{\setminus 2\setminus}$:
 - i. Compute $(K_2)_{\setminus 2\setminus} = (K_1)_{\setminus 2\setminus} \oplus \Delta_{\setminus 2\setminus}$. Assume that $\hat{Y}_4^{j,1}$ is encrypted with K_1 and $\hat{Y}_4^{j,2}$ is encrypted with K_2 , and let $\Delta_r = Y_r^{j,1} \oplus Y_r^{j,2}$. Use the LED Super-Sbox to compute $MCS^{-1}(\Delta_{10})_{/2/} = MCS^{-1}(Y_{10}^{j,1})_{/2/} \oplus MCS^{-1}(Y_{10}^{j,2})_{/2/}$.
 - ii. Since $MCS^{-1}(\Delta_{10})_{/0,1/} = 0$ and $MCS^{-1}(\Delta_{10})_{/2/}$ is known, we know 48 bits of $MCS^{-1}(\Delta_{10})$. We now assume that $\Delta_{10, \setminus 0\setminus} = 0$, and compute the full $MCS^{-1}(\Delta_{10})$ using linear algebra (as MCS is a linear operation).

⁸ The related key attack of [11] required that we can choose the full 64 bits of Δ , so our attack is slightly more generic.

⁹ Since the preprocessing algorithm of this attack is more involved than in the previous attacks, we describe it separately from the online algorithm.

- iii. Use the difference distribution table for the LED Super-Sbox, and the knowledge of $\hat{\Delta}_8, MCS^{-1}(\Delta_{10})$ to compute suggestions for the actual $MCS^{-1}(Y_{10}^{j,1})_{/3/}, MCS^{-1}(Y_{10}^{j,2})_{/3/}$ and $(K_1)_{\setminus 3\setminus}$.
- iv. We now have suggestions for 32 bits of K_1 , 32 bits of $MCS^{-1}(Y_{10}^{j,1})$ and 32 bits of $MCS^{-1}(Y_{10}^{j,2})$. During the online phase, we will obtain suggestions for (specific) 48 bits of K'_1 , 48 bits of $X_{10}^{i,1}$ and 48 bits of $X_{10}^{i,2}$ (the encryption values of some $P^{i,1}$ and $P^{i,2}$ after 10 rounds). Thus, we can compute the values of a total of 48 linear expressions to serve as filtering bits: 16 expressions on the bits of K_1 (a linear key sieve), 16 expressions on $Y_{10}^{j,1}$ and 16 expressions on $Y_{10}^{j,2}$. We store the values of these expressions in a sorted list L , next to $(K_1)_{\setminus 2,3\setminus}$.

As described in Appendix A, the time complexity of Step 1 is about 2^{48} evaluations of 1-step LED. On average, we expect a single suggestion for the values computed in Step 2.(a).iii (using the difference distribution table of the LED Super-Sbox). Thus, we perform only a few simple operations for each of the 2^{31} pairs (computed in Step 1) and the 2^{16} possible values of $(K_1)_{\setminus 2\setminus}$, implying that the total time complexity of the preprocessing algorithm is about 2^{48} evaluations of 1-step LED. In order to slightly reduce the data complexity of the online algorithm (at the expense of using slightly more memory), we repeat Step 2.(a) twice for each ordered pair, exchanging the roles of $\hat{Y}_4^{j,1}$ and $\hat{Y}_4^{j,2}$ (i.e., by assuming that $\hat{Y}_4^{j,1}$ is encrypted with K_2). Thus, the time complexity of the preprocessing algorithm is about 2^{49} evaluations of 1-step LED, and its memory complexity is about 2^{49} .

The online algorithm of the attack is given below.

1. For 2^{48} arbitrary values of the plaintext $P^{i,1}$:
 - (a) Ask for the encryption of $P^{i,1}$ under the key K_1 , and for the encryption of $P^{i,2} = P^{i,1} \oplus \Delta$ under the key $K_2 = K_1 \oplus \Delta$. Let $A_r = X_r^{j,1} \oplus X_r^{j,2}$.
 - (b) Compute $C'^{i,1} = MCS^{-1}(C^{i,1})$ and $C'^{i,2} = MCS^{-1}(C^{i,1})$. Check if $(C'^{i,1} \oplus C'^{i,2})_{/0/} = 0$, and if not, return to Step 1.
 - (c) For each value of $(K'_1)_{/1/}$:
 - i. Compute $(K'_2)_{/1/} = (K'_1)_{/1/} \oplus MCS^{-1}(\Delta)_{/1/}$ and use the LED Super-Sbox to compute $A_{10,\setminus 1\setminus} = X_{10,\setminus 1\setminus}^{i,1} \oplus X_{10,\setminus 1\setminus}^{i,2}$.
 - ii. Assume that $MCS^{-1}(A_{10})_{/0,1/} = 0$, and compute A_{10} (using the fact that $A_{10,\setminus 0\setminus} = 0$ and $A_{10,\setminus 1\setminus}$ is known).
 - iii. Use the difference distribution table for the LED Super-Sbox, and the knowledge of A_{10} and $(C'^{j,1} \oplus C'^{j,2})$ to compute suggestions for $X_{10,\setminus 2\setminus}^{i,1}, X_{10,\setminus 2\setminus}^{i,2}$ and $(K'_1)_{/2/}$, and similarly compute suggestions for $X_{10,\setminus 3\setminus}^{i,1}, X_{10,\setminus 3\setminus}^{i,2}$ and $(K'_1)_{/3/}$.
 - iv. From the knowledge of 48 bits of K'_1 , 48 bits of $X_{10}^{i,1}$ and 48 bits of $X_{10}^{i,2}$, compute the filtering values of the 48 linear expressions, and look for matches in the list L .
 - v. For each match, obtain $(K_1)_{\setminus 2,3\setminus}$, compute a suggestion for K_1 and test it.

A pair $((P^{i,1}, C^{i,1}), (P^{i,2}, C^{i,2}))$ passes the 16-bit filtering condition of Step 1.(b) with probability 2^{-16} , and thus we expect to process about $2^{48-16} = 2^{32}$ pairs in Step 1.(c). As we store 2^{32} ordered input pairs of values for F_2 with an input difference of Δ , and each of the processed 2^{32} plaintext pairs has a difference of Δ at the input to F_2 , we expect a collision between these two groups of pairs. For such a collision, the assumptions made during the online and preprocessing algorithms hold ($\Delta_{10, \setminus 0} = 0$ is assumed in preprocessing Step 2.(a).ii, and $MCS^{-1}(A_{10})_{/0,1/} = 0$ is assumed in online Step 1.(c).ii). Thus, this collision will yield a match in Step 1.(c).iv, suggesting the correct value of K_1 .

Since we expect a single suggestion for the values computed in Step 1.(c).iii, we perform a few simple operations for each of the 2^{32} processed pairs in Step 1.(b) and the 2^{16} possible values of $(K'_1)_{/1/}$, on which we iterate in Step 1.(c). Thus, the total time complexity of the online algorithm is about 2^{49} evaluations of 1-step LED. Including the preprocessing time, the total time complexity of the attack is about 2^{49} evaluations of 2-step LED (which is a bit less than 2^{49} evaluations of the full 3-step scheme). The data complexity of the attack is 2^{49} , and its memory complexity is 2^{49} , required in order to store the list L .

A Single-Key Attack on a Variant of 2-Step LED with Independent Keys

Consider a variant of 8-round LED, where the three round keys K_1 , K_2 and K_3 are independent. We now show how to adapt the above attack to this scheme with about the same time/memory/data complexities. We note that this construction has a similar structure to the block cipher AES² [2], which is composed of two key-less AES-128 permutations such that K_1 and K_2 are added before and after the first permutation and K_3 is added after the second permutation. Since our techniques only exploit the AES structure of the LED step function, this attack can also be applied to AES², reduced from 20 rounds to 8 rounds (with the complexity of the attack adjusted to the 128-bit cipher).

In this attack, we select Δ in a similar way to the related-key attack above. However, in the single-key attack we cannot inject a pair of different messages such that they have a zero difference at the input to F_1 . Thus, we now preprocess F_1 (instead of F_2) and find 2^{31} pairs of inputs to this function, $\hat{Y}_0^{j,1}, \hat{Y}_0^{j,2} = \hat{Y}_0^{j,1} \oplus \Delta$, such that $(\hat{Y}_4^{j,1} \oplus \hat{Y}_4^{j,2})_{\setminus 0,1} = 0$. Consequently, in the online algorithm, we request the encryptions of 2^{48} pairs of plaintexts with an input difference of Δ , and apply the differential MITM technique to F_2 .

Another difference between this attack and the previous related-key attack on LED-64, is that now K_1 , K_2 and K_3 are independent, and thus we do not have any filtering conditions on the key when matching the suggestions during the MITM phase (i.e., we do not have a linear key sieve). This implies that if we use only one pair of plaintexts, the complexity of the attack will be at least 2^{64} . In order to speed up the attack, we use the same idea used in [6] in the attack on AES²: we choose an arbitrary non-zero difference $\Delta' \neq \Delta$, and ask for the encryption of another plaintext $P^{i,1} \oplus \Delta'$ for each pair $P^{i,1}$ and $P^{i,1} \oplus \Delta$. Similarly, we attach another evaluation of $\hat{Y}_0^{j,1} \oplus \Delta'$ to each evaluated pair $\hat{Y}_0^{j,1}$ and $\hat{Y}_0^{j,1} \oplus \Delta$. This allows us to obtain the required filtering values such that

the attack has similar time/memory/data complexities to the related-key attack. The full details of the filtering technique are found in [6].

7 Conclusions

In this paper, we introduced various techniques in MITM attacks including the linear key sieve technique, a known plaintext splice-and-cut attack, and new techniques for differential MITM. We applied these techniques to step-reduced LED-64 and obtained the best known results on this block cipher, both in the single-key and related-key models. Although our techniques are mainly applied to LED-64, we believe that they will be useful in the analysis of other cryptosystems, in particular AES-based cryptosystems and lightweight block ciphers.

References

1. Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In Roberto Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.
2. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, François-Xavier Standaert, John P. Steinberger, and Elmar Tischhauser. Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations - (Extended Abstract). In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 45–62. Springer, 2012.
3. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Wang and Sako [15], pages 208–225.
4. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 169–187. Springer, 2011.
5. Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-Middle: Improved MITM Attacks. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 222–240. Springer, 2013.
6. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Key Recovery Attacks on 3-round Even-Mansour, 8-step LED-128, and Full AES². *IACR Cryptology ePrint Archive*, 2013:391, 2013. Accepted to Asiacrypt 2013. To appear in *Lecture Notes in Computer Science*.
7. Shimon Even and Yishay Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. *J. Cryptology*, 10(3):151–162, 1997.
8. Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block Ciphers that are Easier to Mask: How Far Can we Go? In *CHES*, 2013. To appear in *Lecture Notes in Computer Science*.

9. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.
10. Takanori Isebe and Kyoji Shibutani. Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. In Willy Susilo, Yi Mu, and Jennifer Seberry, editors, *ACISP*, volume 7372 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2012.
11. Florian Mendel, Vincent Rijmen, Deniz Toz, and Kerem Varici. Differential Analysis of the LED Block Cipher. In Wang and Sako [15], pages 190–207.
12. Ralph C. Merkle and Martin E. Hellman. On the Security of Multiple Encryption. *Commun. ACM*, 24(7):465–467, 1981.
13. Yu Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 378–396. Springer, 2011.
14. Yu Sasaki and Kazumaro Aoki. Preimage Attacks on Step-Reduced MD5. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP*, volume 5107 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2008.
15. Xiaoyun Wang and Kazue Sako, editors. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*. Springer, 2012.

A The Auxiliary Preprocessing Algorithm of the Related-Key Attack on 3-Step LED-64

Our goal in the auxiliary preprocessing algorithm is to find (about) 2^{31} pairs $(\hat{Y}_4^{j,1}, \hat{Y}_4^{j,2} = \hat{Y}_4^{j,1} \oplus \Delta)$, such that $(Y_8^{j,1} \oplus Y_8^{j,2})_{\setminus 0,1} = \Delta_{\setminus 0,1}$. In order to run in time of about 2^{48} evaluations of 1-step LED, we assume that $\Delta_{\setminus 0,1,2} = 0$, and the value of $\Delta_{\setminus 3}$ is arbitrary (but non-zero).

Let $\Delta_r = Y_r^{j,1} \oplus Y_r^{j,2}$. The difference Δ_8 can obtain $2^{32} - 1$ non-zero values (since we require that $\Delta_{8,\setminus 0,1} = (Y_8^{j,1} \oplus Y_8^{j,2})_{\setminus 0,1} = \Delta_{\setminus 0,1} = 0$), and in addition $MCS^{-1}(\Delta_6)$ can obtain (at most) $2^{16} - 1$ non-zero values, implying that Δ_6 can obtain $2^{16} - 1$ non-zero values. The algorithm is given below.

1. For each of the possible $2^{16} - 1$ non-zero value of Δ_6 :
 - (a) For each non-zero value of Δ_8 such that $\Delta_{8,\setminus 0,1} = 0$:
 - i. Calculate $MCS^{-1}(\Delta_8)$. Given Δ_6 , use the difference distribution tables for the LED Super-Sboxes to obtain the actual values $Y_8^{j,1}, Y_8^{j,2}$.
 - ii. Compute $\hat{Y}_4^{j,1} = F_2^{-1}(Y_8^{j,1})$ and $\hat{Y}_4^{j,2} = F_2^{-1}(Y_8^{j,2})$, and if $(\hat{Y}_4^{j,1} \oplus \hat{Y}_4^{j,2})_{\setminus 3} = \Delta_{\setminus 3}$, store the pair.

Since we obtain an average of one value for $Y_8^{j,1}$ and $Y_8^{j,2}$ in Step 1.(a).i (when considering ordered pairs), the expected time complexity of the algorithm is 2^{48} . The condition of Step 1.(a).ii holds for about 2^{-16} of the pairs, and thus we expect to return 2^{32} ordered pairs, or 2^{31} unordered pairs as claimed. We note that it is possible to implement the algorithm such that it always returns (at least) 2^{31} pairs. This can be achieved by storing all the 2^{48} pairs in Step 1.(a).ii, and finally setting the value of $\Delta_{\setminus 3}$ to a value for which there is a maximal number of pairs.