Key Recovery Attacks on 3-round Even-Mansour, 8-step LED-128, and Full AES^2

Itai Dinur[1], Orr Dunkelman[1,2,⋆], Nathan Keller[1,3,⋆⋆], and Adi Shamir[1]

[1] Computer Science department, The Weizmann Institute, Rehovot, Israel
[2] Computer Science Department, University of Haifa, Israel
[3] Department of Mathematics, Bar-Ilan University, Israel

Abstract. The Even-Mansour (EM) encryption scheme received a lot of attention in the last couple of years due to its exceptional simplicity and tight security proofs. The original 1-round construction was naturally generalized into r-round structures with one key, two alternating keys, and completely independent keys. In this paper we describe the first key recovery attack on the one-key 3-round version of EM which is asymptotically faster than exhaustive search (in the sense that its running time is $o(2^n)$ rather than $O(2^n)$ for an n-bit key). We then use the new cryptanalytic techniques in order to improve the best known attacks on several concrete EM-like schemes. In the case of LED-128, the best previously known attack could only be applied to 6 of its 12 steps. In this paper we develop a new attack which increases the number of attacked steps to 8, is slightly faster than the previous attack on 6 steps, and uses about a thousand times less data. Finally, we describe the first attack on the full AES^2 (which uses two complete AES-128 encryptions and three independent 128-bit keys, and looks exceptionally strong) which is about 7 times faster than a standard meet-in-the-middle attack, thus violating its security claim.

Keywords: Cryptanalysis, key recovery attacks, iterated Even-Mansour, LED encryption scheme, AES^2 encryption scheme.

1 Introduction

The Even-Mansour (EM) block cipher was first proposed at Asiacrypt'1991 [8]. It uses a single publicly known random permutation P on n-bit values and two secret n-bit keys K_1 and K_2, and defines the encryption of the n-bit plaintext m as $E(m) = P(m \oplus K_1) \oplus K_2$. The decryption of the n-bit ciphertext c is similarly defined as $D(c) = P^{-1}(c \oplus K_2) \oplus K_1$. It can be naturally generalized into an r-round iterated EM encryption function (a.k.a. a key-alternating scheme in [1, 5]), which is defined using r permutations $P_1, P_2, \ldots, P_r$ and $r+1$ keys $K_1, K_2, \ldots K_{r+1}$ as $E(m) = P_r(\ldots P_2(P_1(m \oplus K_1) \oplus K_2) \oplus K_3 \ldots \oplus K_r) \oplus K_{r+1}$, where decryption is defined in an analogous way.

⋆ 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.

For about 20 years this scheme received very little attention in the cryptographic literature, but in the last couple of years it became a very active research area: multiple papers about this scheme appeared at Crypto, Eurocrypt, Asiacrypt, CHES and FSE [1, 5, 7, 11, 13–15], analyzing its theoretical properties, generalizing it in various ways, and proposing concrete constructions of block ciphers which are based on the EM structure.

In this paper we describe several new key recovery attacks on iterated EM schemes, analyze their complexity, and apply them to some concrete proposals of block ciphers which have this structure. The origin of the observations used in our attacks goes back to the first paper which attacked EM, by Daemen [6] in 1991. Daemen observed that in single-round one-key EM, an attacker can use the fact that the XOR of the unknown input and output of the permutation P is equal to the known XOR of the plaintext and the ciphertext. This observation can be used to break 1-round EM significantly faster than exhaustive search.

At FSE'13, Nikolić et al. [15] extended the basic observation considerably. They considered the graph of the function $P'(x) = x \oplus P(x)$,[1] and showed that vertices with a large in-degree in this graph can be exploited to bypass an additional round of EM, but at the expense of enlarging the time complexity to slightly less than exhaustive key search.

In this paper, we develop the techniques one step further, and show that graphs of the functions P'_1 and P'_3 (corresponding to the permutations P_1 and P_3) can be deployed simultaneously, resulting in an attack on 3-round EM. However, this enhancement is not sufficient by itself, since the time complexity becomes very close to that of exhaustive key search. Nevertheless, a surprising feature of our 3 round attack is that it has about the same time complexity as the 2-round attack. This feature is due to a novel filtering technique based on tailor-made linear subspaces that we develop in Section 2.2, and allows us to quickly dispose of data which is useless for our attack. Another novel technique that we develop in this paper allows us to adapt the differential-based attack of [14] (which was originally applied to 2-round iterated EM with one key) to 2-round iterated EM with completely independent keys, and thus to attack the full AES^2 scheme. While the attack of [14] makes use of plaintext pairs with a fixed difference, we notice that in its original form it cannot improve the standard meet-in-the-middle attack on this scheme. In our attack, we work on non-standard structures of plaintext triplets which allow us to filter out wrong guesses for the key more efficiently.

Throughout the paper we follow the standard conventions in the analysis of time and memory complexities. Our basic unit of memory is an n-bit block. Our basic unit of time is a single evaluation of the encryption or the decryption function, i.e., the full r-round iterated EM scheme. The scheme requires the evaluation of the r permutations P_i (which are assumed to be heavy operations) and a small number of simple operations (such as XORs) which are assumed to

[1] In [15], the permutation P is actually the full encryption function, and thus x is a message and $P(x)$ is its corresponding ciphertext.

require negligible time.[2] Thus, an invocation of a single permutation P_i (or its inverse) costs $1/r$ time units. For the sake of convenience, we often partition the attack into an *offline preprocessing phase* which analyzes the properties of the public P_i's, and an *online attack phase* which analyzes the given plaintexts and ciphertexts. However, we always define the time complexity of the attack as the sum of the complexities of its offline and online phases. This is different from the model used by Hellman in his time/memory tradeoff attack, which allowed unlimited free preprocessing and considered only the online complexity (note that in our model, Hellman's attack is no better than exhaustive search). To prevent other types of "cheating", we always add the time required to generate the data to the final time complexity, and add the space required to hold the data to the final space complexity.

All our attacks are only slightly better than exhaustive search, which raises the natural question whether they should be considered as legitimate attacks. This is a general problem in cryptanalysis, since it is difficult to decide whether an attack such as the Biclique attack on AES-128 [4] which requires 2^{126} time really "breaks" a scheme whose exhaustive search requires 2^{128} time. Some researchers suggested that this issue should be decided by the nature of the attack: If an attack on an n-bit scheme has an outer loop which tries 2^n different possibilities, but performs for each one of them an operation which is cheaper than a single encryption, then the attack should be called an "improvement of exhaustive search" rather than a "real attack", and the scheme is not said to be "broken" by it. However, this is a fragile definition since the same attack can be described in multiple ways, and it is not always clear whether it tries 2^n or fewer possibilities.

Fortunately, in cryptographic schemes such as EM which can be naturally defined for arbitrarily large key sizes n, we can avoid this fragility by analyzing the asymptotic complexity of the attack. As we show in this paper, our attacks are about $n/\log(n)$ times faster than exhaustive search. Since this ratio is unbounded when n increases, our attacks are asymptotically better than any standard or improved version of exhaustive search, and this is a robust statement since it ignores all the multiplicative constants which are associated with a particular model of computation.

Some of the concrete schemes we consider in this paper (such as LED and AES^2) pose the following problem: they use the general EM framework, but instantiate P with a fixed-key AES-like permutation which is defined only for a few values of n, and thus it is difficult to define their asymptotic security. We solve this problem in two ways. First, we observe that all our attacks are completely generic, and do not exploit any particular properties of P besides its randomness. We can thus analyze the performance of our attacks assuming that AES is replaced in these schemes by a random permutation over n-bit

[2] This complexity gap is typically large for normal choices of n, and likely to grow even larger as n increases: the number of 2-bit to 1-bit gates in the Boolean circuit of P_i which are needed to thoroughly and independently mix the n input bits into n output bits is expected to grow super-linearly with n, whereas the number of gates in the Boolean circuit of XOR grows only linearly with n.

values, and show that their asymptotic time complexity is $o(2^n)$. In addition, we carefully analyze the exact complexity of our attacks for the particular values of n recommended for these schemes, and show that they are between 7 and 20 times faster than exhaustive search, depending on which scheme we attack.

We would like to point out that some of the previously published attacks on these schemes (such as [5]) are distinguishing attacks, and thus they are incomparable to our key recovery attacks. In addition, our attacks may fail to find the key or require longer than expected time for a small fraction of "bad" permutations, since we only analyze their expected behavior when the P_i's are randomly chosen permutations.

The paper is organized as follows. In Section 2, we introduce our new cryptanalytic techniques, and use them to attack the one-key, three-round version of EM (the best previous attack could only handle the two-round version of EM). In particular, our new attack influenced the decision of the designers of the Zorro block cipher [10] to increase its number of steps from 3 to 6. In Section 3, we consider the LED block cipher, which was proposed at CHES 2011 [11]. It has two flavors: a one-key version called LED-64, and a two-key version (in which the two keys are alternately used) called LED-128. In the case of LED-64, the best previously published attack [12] appeared at ACISP 2012, and could only handle 2 steps. We increase the number of steps we can attack from 2 to 3. In the case of LED-128, the best previously published attack [15] appeared at FSE 2013, and could handle 6 steps out of the 12 steps of full LED-128. We increase this number to 8, using smaller time and data complexities. In Section 4, we consider the generalized version of EM in which all the keys are completely independent, and show how to attack the 2-round version of this scheme. We then use the new techniques in order to describe the first published attack on the full version of the block cipher AES^2, which was presented at Eurocrpyt 2012 by [5]. The scheme looks exceptionally strong, using two complete AES encryptions and three independent 128-bit keys. In fact, the designers of AES^2 conjectured that the best attack on their scheme is a meet-in-the-middle attack, but our new attack disproves this claim since it is about 7 times faster.

2 Attacks on Iterated Even-Mansour with One Key

We first consider iterated EM schemes with one key K and r permutations $P_1, P_2, \ldots, P_r$, as shown in Figure 1 (note that if all the permutations are also the same, the scheme is extremely vulnerable to slide attacks [3]). Our goal is to use properties of one of the public permutations $P \in \{P_1, P_2, \ldots, P_r\}$ in order to deduce properties of the associated keyed permutation[3] $Q(K, x) = K \oplus P(x \oplus K)$ (used inside the EM construction), which hold for any value of K. As Daeman pointed out in 1991 [6], for any value of K and in any invocation of $Q(K, x)$, the XOR of its input and output is equal to the XOR of the input and output of the internal P function in the same invocation, i.e., $x \oplus Q(K, x) = (x \oplus K) \oplus P(x \oplus K)$.

[3] In general, given some public permutation P_i, we denote $Q_i(K, x) = K \oplus P_i(x \oplus K)$.

Another interesting observation is that when K is unknown we cannot determine $x \oplus K$, but the addition of K just renames the input vertices in the bipartite graph of $P'(x) = x \oplus P(x)$, and thus it preserves the distribution of in-degrees of its output vertices. In particular, if some output values of P' are more likely than expected (i.e., appear more than the average), then we can predict the value $Q(K, x)$ with a higher probability than expected even when K is unknown. More specifically, any t-way collision on the value v in P', namely $x_1, x_2, \ldots, x_t$ such that $x_1 \oplus P(x_1) = x_2 \oplus P(x_2) = \ldots = x_t \oplus P(x_t) = v$ for some value of v, yields a t-way collision on the value v in the function $Q'(K, x) = x \oplus Q(K, x) = x \oplus K \oplus P(x \oplus K)$. Assume that indeed we manage to find during a preprocessing phase a large t-way collision in the public $P'(x)$ on the output value v. Since it also yields a t-way collision on the value v in the keyed function $Q'(K, x)$, there are at least t values of x for which $Q'(K, x) = v$, and thus $Q(K, x) = x \oplus v$. Consequently, we can guess $Q(K, x)$ with a probability which is t times higher than the expected $1/2^n$ even when we know nothing about K.

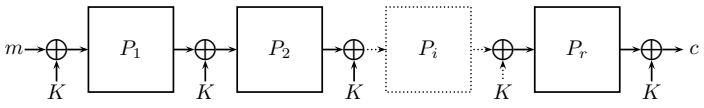

Fig. 1. An iterated EM with one key

This graph theoretic property is strongly related to the one used in [15], but we use it in a different way. Whereas we use properties of the public permutations (which can be observed during a preprocessing phase), [15] exploits properties of the given plaintext-ciphertext pairs: assume that $m_j \oplus c_j = v$ for multiple plaintext-ciphertext pairs (m_j, c_j). Then, for all of these pairs, they know that $(m_j \oplus K) \oplus (c_j \oplus K) = v$. Thus, the attack of [15] is based on the property that the XOR of the inputs to the first and last public permutations P_1 and P_r^{-1} attain the value v more than the expected number of times. In particular, in their attack it is not clear how to compute such a v during a preprocessing phase, and they have to wait for the actual data in order to search for the best v in it. Our attacks, on the other hand, are based on the property that the XOR of the input and output of a single public permutation attains some value v more than the expected number of times, and thus we can find the best v once and for all, before any data is given for a particular key.

In order to estimate the highest expected in-degree in the bipartite graph of $P'(x) = x \oplus P(x)$, we assume that for a random choice of the permutation P, the function P' behaves as a random function. This is not completely true, since there are some extremely expensive ways to distinguish between such cases (for example, the XOR of all the 2^n values of P' is zero, whereas the XOR of all the outputs of a truly random function is unlikely to be zero). However, it is easy to

verify with appropriate simulations that the in-degree distributions of the two models behave almost identically, which is all we need in our attack.[4]

The main problem in applying this attack is that going over all the 2^n possible values of x in order to find the most popular v will make our attack slower than exhaustive search (since we do not allow free preprocessing in our model). Fortunately, we can find vertices v' which are almost as popular by trying only a small subset $X \subseteq \{0,1\}^n$ of possible inputs. We denote this restricted function by $f_{|X}$, and note that it induces a subgraph in the bipartite graph associated with f, in which the left side of the graph contains only the vertices in X. Our goal now is to analyze the expected distribution of the in-degrees in random subgraphs of random functions.

Random functions have been extensively analyzed in the literature (e.g., see [9]). It is well-known that the in-degree of an element in the range of $f_{|X}$ is distributed according to the Poisson distribution with an expectation λ, which is equal to the average in-degree (i.e., $\lambda = |X|/2^n$, which is the ratio between the sizes of the domain and range of $f_{|X}$). Given a parameter t, the probability that an arbitrary element v will have an in-degree of t is thus $(\lambda^t e^{-\lambda})/t!$. We have 2^n elements in the range, implying that we expect that about $(2^n \cdot \lambda^t e^{-\lambda})/t!$ vertices will have an in-degree of t. If we equate this number to 1 and ignore low order terms, we can deduce that the largest expected in-degree t satisfies $t \cdot \log(t) = n$, and thus t is approximately equal to $n/\log(n)$. The crucial point is that this highest in-degree grows in an unbounded way as n increases, and thus any complexity of the form $O(2^n/t)$ behaves asymptotically as $o(2^n)$. If we reduce this maximal t to $t-i$ for a small i, we expect to find about $(t/\lambda)^i$ vertices which have this reduced in-degree. Since $t > 1$ and $\lambda < 1$, this number grows exponentially with i, and we can thus find a huge number of vertices which have almost maximal in-degrees.

To get a sense of the concrete values implied by this distribution, consider the recommended value of $n = 64$ in the LED block cipher. If we consider all the 2^{64} possible inputs, we expect to see 2 or 3 vertices of degree 20, 55 vertices of degree 19, and 1060 vertices of degree 18. If we reduce the number of possible inputs to 2^{63}, we expect to see 1 vertex of degree 17, 8 vertices of degree 16, and 260 vertices of degree 15. If we further reduce the number of possible inputs to 2^{60}, we expect to see 4 vertices of degree 10, 695 vertices of degree 9, and $100130 \approx 2^{16.6}$ vertices of degree 8.

The attacks in this paper are described in terms of several parameters, and it is usually possible to obtain various tradeoffs between their time, data and memory complexities by tweaking the parameter values. However, since there is no simple formula which describes the exact tradeoff curves, one needs to determine favorable tradeoff points on the curves by plugging in a few values for

[4] In fact, collisions in P'(x) are slightly less likely to occur when P is a random function, since if $P(x) = P(y)$ (for $x \neq y$) then $P'(x) \neq P'(y)$, whereas if P is random permutation then $x \neq y$ implies $P(x) \neq P(y)$, and the probability for $P'(x) = P'(y)$ is a bit higher. As a result, our analysis slightly underestimates the highest expected in-degree, and thus the attacks that we describe are actually (negligibly) faster.

the parameters and calculating the resultant complexities of the algorithms. This is demonstrated in our attacks, where we suggest concrete points on the curves which minimize the time complexity, but stress that there are other options as well.

2.1 Attacks on 2-Round Iterated Even-Mansour with One Key

We start by describing a very basic attack, $2Round1KeyBasic$. Let S and D be parameters.

Preprocessing:

PR1. Evaluate P_1' on an arbitrary subset of inputs X, such that $|X| = S$, and store the output values (without their associated input values) in a sorted list.

PR2. Traverse the sorted list and find the output v_1 which occurs the maximal number of times (in t_1 consecutive locations).

Online:

O1. Ask for the encryption of D arbitrary plaintexts.

O2. For each plaintext-ciphertext pair (m_i, c_i):
   (a) Assume that $Q_1(K, m_i) = m_i \oplus v_1 \triangleq z_i$ and calculate $P_2(z_i)$.
   (b) Test the suggestion for the key $K' = P_2(z_i) \oplus c_i$ by checking whether indeed $Q_1(K', m_i) = m_i \oplus v_1$. If the test fails, increment i and return to Step O2. Otherwise, return the suggested key.

The time complexity of the preprocessing phase is S evaluations of P_1, and its memory complexity is also S. Note that the output of the preprocessing phase is only the value v_1 and the corresponding number t_1, and we can discard the rest of the sorted list (In our model, we can ignore the sorting time of the list, since sorting uses only cheap comparison operations.[5]). In addition, since we can execute the online phase in streaming mode by working on each given plaintext-ciphertext pair independently and discarding it afterwards, its memory complexity is negligible. The expected time and data complexities of the online phase depend on the value of t_1: we know that there are at least t_1 values of x such that $Q_1(K, x) = x \oplus v_1$. According to the birthday paradox, after trying about $2^n/t_1$ arbitrary messages we expect that at least one m_i will satisfy $Q_1(K, m_i) = m_i \oplus v_1$ and suggest the correct value of K. Thus, the expected data complexity of the online algorithm is $2^n/t_1$, and in order to compute its time complexity, we need to sum $2^n/t_1$ evaluations of P_2 in Step O2.(b), and $2^n/t_1$ encryptions in order to generate the data.

[5] One may notice that since sorting requires $O(n \log(n))$ basic operations, then our algorithm actually requires about 2^n basic operations. However, as mentioned before we expect the circuit size of any reasonable choice of P_1 to grow at least as $n^{1+\epsilon}$ (for some $\epsilon > 0$) when n increases, and thus the real time complexity of exhaustive search is in fact $O(n^{1+\epsilon} \cdot 2^n)$ basic operations, which is asymptotically larger than the number of basic operations performed by our algorithm when we take the sorting time of $\tilde{O}(2^n/n)$ values into account.

Optimizing the Basic Algorithm We now describe several useful optimizations of the $2Round1KeyBasic$ algorithm. The first optimization is to use the freedom to choose the subset X during the preprocessing phase in order to immediately filter out most of the wrong key suggestions that are now filtered only in Step O2.(b) of the online algorithm, and thus avoid the Q_1 evaluations in these cases. The idea uses a technique that resembles (but is not the same as) splice-and-cut [2]: assume that we choose the set X of size S as the subspace of values x in which the $n-\log(S)$ LSBs are zero (or any other constant). Then the value of these $n-\log(S)$ LSBs in all the t_1 inputs x that satisfy $P'_{1|X}(x) = v_1$ is zero. Consequently, we know that for any plaintext m_i, if $m_i \oplus K$ is one of these t_1 inputs, then the $n-\log(S)$ LSBs of K are equal to those of m_i. Thus, before testing the suggested key in Step O2.(b), we can check whether its $n-\log(S)$ LSBs are equal to those of m_i, and otherwise discard it without evaluating Q_1. We note that in this attack, the saving in time complexity due to this optimization is small, however, in Section 2.2 we show that a similar idea yields a more significant saving in our attacks on 3-round iterated EM. We alert the reader that even though the values in X are now chosen in a specific way, the attack remains a known plaintext attack since there is no restriction on the choice of the m_i's.

The second optimization is to consider $\ell > 1$ outputs of P'_1 with a high in-degree instead of just one. This allows us to reduce the data complexity of the attack at the expense of using more memory and slightly more time during the online phase of the attack. Since the original online algorithm required only negligible memory, this tradeoff seems favorable. Our optimized algorithm $2Round1KeyOpt$ is described below, using S, D and ℓ as parameters.

Preprocessing:

PR1. Evaluate P'_1 on a subset of S inputs, X, such that the $n-\log(S)$ LSBs of each $x \in X$ are zero. Store the output values in a sorted list.

PR2. Traverse the sorted list and store the outputs $v_1, v_2, \ldots, v_\ell$ which have the highest in-degrees. Denote the in-degrees of the outputs $v_1, v_2, \ldots, v_\ell$ by $t_1, t_2, \ldots, t_\ell$, respectively.

Online:

O1. Ask for the encryption of D arbitrary plaintexts.

O2. For each plaintext-ciphertext pair (m_i, c_i):

(a) For $j \in \{1, 2, \ldots, \ell\}$:

i. Assume that $Q_1(K, m_i) = m_i \oplus v_j \triangleq z_{ij}$ and calculate $P_2(z_{ij})$.

ii. Let $K' = P_2(z_{ij}) \oplus c_i$. If the $n-\log(S)$ LSBs of K' are different from those of c_i, discard it and return to Step O2.(a) (if $j = \ell$ return to Step O2). Otherwise, test K' by checking whether $Q_1(K', m_i) = m_i \oplus v_j$. If the test succeeds, return K', otherwise, if $j < \ell$ return to Step O2.(a) and if $j = \ell$ return to Step O2.

As in the $2Round1KeyBasic$, the time complexity of the preprocessing phase is S evaluations of P_1, and its memory complexity is also S. However, in $2Round1KeyOpt$,

a bigger list of size ℓ is carried over to the online algorithm, and thus its memory complexity is increased to ℓ. In order to calculate the time and data complexities, we denote by $\bar{t}$ the average value of $t_1, t_2, \ldots, t_\ell$, and thus there are $\bar{t}\ell$ values of x for which $Q_1(K, x) = x \oplus v_j$ for $j \in \{1, 2, \ldots, \ell\}$. According to the birthday paradox, after trying about $2^n/(\bar{t}\ell)$ arbitrary messages, we expect that at least one m_i will satisfy $Q_1(K, m_i) = m_i \oplus v_j$ and suggest the correct value of K. Thus, the expected data complexity of the attack is $2^n/(\bar{t}\ell)$. Since we perform ℓ evaluations of P_2 per given message, the expected time complexity of the online algorithm is about $\ell \cdot D = 2^n/\bar{t}$ evaluations of P_2, $S/2^n \cdot 2^n/\bar{t} = S/\bar{t}$ evaluations of P_1 in Step O2.(a).ii, and $2^n/(\bar{t}\ell)$ time to generate the data.

Concrete Parameters For $n = 64$, let $S = 2^{60}$, which implies $\lambda = 2^{60}/2^{64} = 2^{-4}$. As shown before, by using the formula $(2^n \cdot \lambda^t e^{-\lambda})/t! = 2^{64} \cdot (2^{-4t} e^{-1/16})/t!$ with $t = 10$, it is easy to check that in such an evaluated subgraph of a random function we expect to see at least $\ell = 4$ vertices with an in-degree of 10. With these parameters, the time complexity of the preprocessing phase is 2^{60} evaluations of P_1 (which is equivalent to 2^{59} evaluations of the 2-round scheme), and its memory complexity is 2^{60}. The memory complexity of the online algorithm is negligible, its data complexity is $2^{64}/(10 \cdot 4) = 2^{58.7}$ known plaintexts and its time complexity is $2^{64}/10$ evaluations of P_2 and $2^{60}/10$ evaluations of P_1, which is equivalent to about $2^{59.8}$ time units. Adding the $2^{58.7}$ time required to generate the data, we obtain a total time complexity of about $2^{60.4}$, which is about 12 times faster than exhaustive search.

We can significantly reduce the data complexity by considering all the vertices with an in-degree of at least 8, whose number ℓ is expected to exceed 2^{16} . This does not affect the time and memory complexities of the preprocessing phase. The memory complexity of the online algorithm is now 2^{16} (which is still quite small), its data complexity is $2^{64}/(8 \cdot 2^{16}) = 2^{45}$ known plaintexts and its time complexity is now $2^{64}/8$ evaluations of P_2 and $2^{60}/8$ evaluations of P_1, which is equivalent in total to about $2^{60.1}$ time units, or about 15 times faster than exhaustive search (note that we actually gain in time complexity since we use significantly less data).

2.2 Attacks on 3-Round Iterated Even-Mansour with One Key

In the attacks on 2-round iterated EM with one key, we use properties of P_1 in order to guess a value of $Q_1(K, x)$ with a higher probability than expected. We then apply to this guess the public permutation P_2, which immediately gives us a suggestion for the key by XORing the obtained value with the ciphertext. In order to attack 3-round iterated EM with one key, we start with the same idea. However, after the evaluation of P_2, we cannot immediately get a suggestion for the key, as we still have to apply the complex operation of XOR'ing the unknown key, applying P_3, and XOR'ing the unknown key again, before we can compare the result to the ciphertext. Nevertheless, we notice that given the value at the output of P_2, we reduce the key recovery problem to attacking a single-round

EM scheme with one key, to which we can apply the simple attack of [7]. Thus, we run an additional preprocessing step which evaluates and stores in a sorted list of values of $P'_3(x) = x \oplus P_3(x)$ for various inputs x. The sorted list is used in the online algorithm in order to obtain suggestions for the key, as described in the basic algorithm $3Round1KeyBasic$ below, which uses S_1, S_3 and D as parameters.

Preprocessing:

PR1. Evaluate P'_1 on an arbitrary subset of inputs X_1 such that $|X_1| = S_1$, and store the output values in a sorted list.

PR2. Traverse the sorted list and find an output v_1 with a maximal in-degree, denoted by t_1.

PR3. Evaluate P'_3 on an arbitrary subset of inputs X_3, such that $|X_3| = S_3$, and store the output values $P'_3(x)$ in a sorted list L_3 next to the corresponding value of $P_3(x)$.

Online:

O1. Ask for the encryption of D arbitrary plaintexts.

O2. For each plaintext-ciphertext pair (m_i, c_i):
   (a) Assume that $Q_1(K, m_i) = m_i \oplus v_1 \triangleq z_i$ and calculate $P_2(z_i)$.
   (b) Look for the value of $P_2(z_i) \oplus c_i$ in L_3. If there is no match, return to Step O2 and increment i.
   (c) For each match of $P_2(z_i) \oplus c_i$, obtain the value of $P_3(x)$ (for which $P_2(z_i) \oplus c_i = P'_3(x) = x \oplus P_3(x)$), and test the key suggestion $K' = P_3(x) \oplus c_i$ by checking whether $Q_1(K', m_i) = m_i \oplus v_1$. If the test fails, continue with the next match (if none remain, return to Step O2). Otherwise, return the key.

The time complexity of the preprocessing phase is S_1 evaluations of P_1 and S_3 evaluations of P_3, and its memory complexity is $max(S_1, S_3)$. Note that we do not need to store any of the values generated in the first step of the preprocessing after Step PR2 terminates. The memory complexity of the online algorithm is S_3. In order to calculate the expected time and data complexities of the online algorithm, we notice that after we process D pairs (m_i, c_i), we expect that at least $(t_1 \cdot D)/2^n$ of them satisfy $Q_1(K, m_i) = m_i \oplus v_1$, and consequently at least $(t_1 \cdot D \cdot S_3)/2^{2n}$ pairs will be matched and suggest the correct value for the key in Step O2.(c). Thus, in order to obtain a correct suggestion for the key, we require $(t_1 \cdot D \cdot S_3)/2^{2n} = 1$, implying that the data complexity of the attack is $D = 2^{2n}/(t_1 \cdot S_3)$. We expect a match in Step O2.(c) for a fraction of $S_3/2^n$ of the (m_i, c_i) pairs. Thus, we estimate the time complexity of the online algorithm as $D = 2^{2n}/(t_1 \cdot S_3)$ evaluations of P_2, $S_3/2^n \cdot 2^{2n}/(t_1 \cdot S_3) = 2^n/t_1$ evaluations of P_1, and $2^{2n}/(t_1 \cdot S_3)$ time required to generate the data.

Optimizing the Basic Algorithm Similarly to our $2Round1KeyOpt$ attack, we would like to use the freedom to choose the subset X_1 during preprocessing in order to reduce the time complexity of the attack. However, in this attack we

will use this freedom in a different way: we "synchronize" the sets X_1 and X_3 such that we can instantly rule out most pairs (m_i, c_i) (just by comparing bits of m_i and c_i) that do not simultaneously satisfy both $Q_1(K, m_i) = m_i \oplus v_1$ and $P_3^{-1}(c_i \oplus K) \in X_3$. Thus, we can discard most pairs (m_i, c_i) which will suggest a wrong key (or suggest no key at all) with negligible computation.

We now assume that $|X_1| = |X_3| = S$. Similarly to the $2Round1KeyOpt$ algorithm, we choose X_1 as a subspace of values x in which the $n - \log(S)$ LSBs are zero (or any other constant). This implies that for any plaintext m_i, if $m_i \oplus K$ is one of the t_1 inputs that satisfy $P'_{1|X_1}(x) = v_1$, then the $n - \log(S)$ LSBs of K are equal to those of m_i. As for $x \in X_3$, we store the values of $P'_3(x) = x \oplus P_3(x)$, and set the additional condition that the $n - \log(S)$ LSBs of $P_3(x)$ are zero (or any other constant). In fact, during preprocessing, we do not evaluate $P_3(x)$ on $x \in X_3$, but rather evaluate $P_3^{-1}(y)$ for each $y \in Y_3$, where Y_3 contains all n-bit vectors whose $n - \log(S)$ LSBs are zero. Thus, we know that if $c_i \oplus K \in Y_3$, then the $n - \log(S)$ LSBs of K are equal to those of c_i. Combining the conditions on m_i and c_i, we know that a pair (m_i, c_i) will suggest a correct key in our algorithm only if the $n - \log(S)$ LSBs of m_i and c_i are equal.

Similarly to the $2Round1KeyOpt$ attack, the second optimization is to consider $\ell > 1$ outputs of P'_1 with a high in-degree (instead of just one), which allows us to reduce the data complexity of the attack. Our optimized algorithm $3Round1KeyOpt$ is described below, and Figure 2 illustrates its online part. Let S, D and ℓ be parameters.

Preprocessing:

PR1. Evaluate P'_1 on a subset of S inputs, X, such that the $n - \log(S)$ LSBs of each $x \in X$ are zero. Store the output values in a sorted list.

PR2. Traverse the sorted list and store the outputs $v_1, v_2, \ldots, v_\ell$ with the highest in-degrees. Denote the in-degrees of outputs $v_1, v_2, \ldots, v_\ell$ by $t_1, t_2, \ldots, t_\ell$, respectively.

PR3. Let Y_3 be the subspace of the $|S|$ n-bit vectors in which the $n - \log(S)$ LSBs are zero. For each $y \in Y_3$, store $P_3^{-1}(y) \oplus y = P'_3(P_3^{-1}(y))$ in a sorted list L_3 next to y.

Online:

O1. Ask for the encryption of D arbitrary plaintexts.

O2. For each plaintext-ciphertext pair (m_i, c_i), if the $n - \log(S)$ LSBs of m_i and c_i are not equal, discard it. Otherwise:
   (a) For $j \in \{1, 2, \ldots, \ell\}$:
      i. Assume that $Q_1(K, m_i) = m_i \oplus v_j \triangleq z_{ij}$ and calculate $P_2(z_{ij})$.
      ii. Look for the value of $P_2(z_{ij}) \oplus c_i$ in L_3. If there is no match: if $j < \ell$ return to Step O2.(a), otherwise ($j = \ell$) return to Step O2.
      iii. For each match of $P_2(z_{ij}) \oplus c_i$, obtain the value of y (such that $P_2(z_i) \oplus c_i = P_3^{-1}(y) \oplus y = P'_3(P_3^{-1}(y))$), and test the key suggestion $K' = y \oplus c_i$ by checking whether $Q_1(K, m_i) = m_i \oplus v_j$. If the test succeeds, return K', otherwise, if $j < \ell$ return to Step O2.(a), and if $j = \ell$ return to Step O2.

The time complexity of the preprocessing phase is S evaluations of P_1 and P_3^{-1}, and its memory complexity is $S+\ell$. The memory complexity of the online algorithm is also $S+\ell$. We denote by $\bar{t}$ the average value of $t_1, t_2, \ldots, t_\ell$, and thus there are $\bar{t}\ell$ values of x for which $Q_1(K,x) = x \oplus v_j$ for $j \in \{1,2,\ldots,\ell\}$. Consequently, in order to obtain a correct suggestion for the key, we require that $(\bar{t}\ell \cdot D \cdot S)/2^{2n} = 1$, implying that the data complexity of the attack is $D = 2^{2n}/(\bar{t}\ell \cdot S)$. We process a pair (m_i, c_i) (i.e., we do not discard it in step 2) with probability $S/2^n$, and for each such pair we perform ℓ evaluations of P_2 and for a $S/2^n$ fraction of those we also evaluate Q_1 (or P_1). The expected time complexity of the online algorithm is thus $\ell \cdot S/2^n \cdot D = 2^n/\bar{t}$ evaluations of P_2, $S/\bar{t}$ evaluations of P_1, and $2^{2n}/(\bar{t}\ell \cdot S)$ time required to generate the data.

Thus, the attack has about the same time complexity as the $2Round1KeyOpt$ attack, and for $\ell = 1$ it is more efficient than the $3Round1KeyBasic$ attack by a factor of about $2^n/S$.

Concrete Parameters For $n = 64$, let $S = 2^{60}$, i.e., $\lambda = 2^{60}/2^{64} = 2^{-4}$. Again, we use the formula $(2^n \cdot \lambda^t e^{-\lambda})/t! = (2^{64} \cdot 2^{-4t} e^{-1/16})/t!$ with $t = 8$, such that we expect at least $\ell = 2^{16}$ vertices with an in-degree of 8. With these parameters, the time complexity of the preprocessing phase is 2^{60} evaluations of P_1 (equivalent to about $2^{58.5}$ evaluations of the 3-round scheme), and its memory complexity is 2^{60}. The memory complexity of the online algorithm is 2^{60}, its expected data complexity is $2^{128}/(8 \cdot 2^{16} \cdot 2^{60}) = 2^{49}$ known plaintexts and its expected time complexity is $2^{64}/8$ evaluations of P_2 and $2^{60}/8$ evaluations of P_1, whose sum is equivalent to about $2^{59.6}$ time units (the time required to generate the data is negligible). Note that it is possible to reduce the data complexity further at the expense of increasing the time complexity by considering vertices of a lower in-degree.[6]

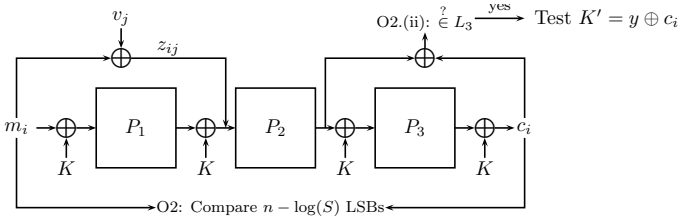

Fig. 2. The online algorithm of $3Round1KeyOpt$

[6] For example, we expect more than 2^{23} vertices with an in-degree of at least 7, and thus if we use only $2^{128}/(8 \cdot 2^{23} \cdot 2^{60}) = 2^{42}$ known plaintexts for the attack, the time complexity of the online algorithm slightly increases from $2^{59.6}$ to about $2^{59.8}$.

3 Applications to Step-Reduced LED

LED is a 64-bit block cipher designed for resource-constrained environments, proposed by Guo et al. at CHES 2011 [11]. The two main variants of LED are LED-64 (which supports 64-bit keys) and LED-128 (which supports 128-bit keys). The design of LED can be viewed as a special case of iterated EM schemes: LED-64 is in fact an 8-step iterated EM scheme[7] with one key and LED-128 is a 12-step iterated EM scheme with alternating keys K_1 and K_2. The inner permutations of LED are based on the AES design framework, however, since our attacks do not exploit any properties of these permutations, we do not specify them here and refer the reader to [11] for further details.

In the single-key model, the best attack published so far on reduced LED-64 breaks 2 steps of this cipher [12]. For LED-128, the largest number of attacked steps was 6 (see [15]). In this paper, we use our generic attacks in order to improve the data complexity of the attack on 6-step LED-128 from 2^{59} to 2^{45}, while keeping the time and memory complexities similar to the original attack. More significantly, we present the first single-key attacks which are faster than exhaustive search on 3-step LED-64 and on 8-step LED-128. The previous attacks on LED (which are in the single-key rather than in the related-key model) and our new attacks are summarized in Table 1. Note in particular that our new attack on 8-step LED-128 actually has a slightly better time complexity and requires about a thousand times less data than the best previous attack which could only be applied to 6 steps of LED-128, out of the full 12.

Reference	Cipher	Steps	Time	Data	Memory
[12]	LED-64	2	2^{56}	2^8 CP	2^{11}
This paper	LED-64	3	$2^{60.2}$	2^{49} KP	2^{60}
[12]	LED-128	4	2^{112}	2^{16} CP	2^{19}
[14]	LED-128	4	2^{96}	2^{64} KP	2^{64}
[15]	LED-128	4	2^{96}	2^{32} KP	2^{32}
[15]	LED-128	6	$2^{124.4}$	2^{59} KP	2^{59}
This paper	LED-128	6	$2^{124.5}$	2^{45} KP	2^{60}
This paper	LED-128	8	$2^{123.8}$	2^{49} KP	2^{60}

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

Table 1. Single-Key Attacks of Step-Reduced LED

[7] In the design of LED, the term "step" is used in order to describe what we refer to as a "round" of an iterated EM scheme. On the other hand, a "round" of LED is used in order to describe a smaller component of its internal permutation. Thus, in order to avoid confusion, we will use the term "step" in this section.

3.1 An attack on 6 Steps of LED-128

As was pointed out in [14, 15], it is easy to reduce $2r+2$-steps of LED-128 (with its alternating use of two keys) into an iterated EM scheme variant with one key by guessing K_1 and combining consecutive pairs of permutations (along with the XOR'ed key between them) into a single known permutation. In particular, [15] reduced 6-step LED-128 into a 2-step iterated EM, which was relatively easy to attack. Similarly, we guess K_1, and for each guess, we partially encrypt and decrypt the given plaintext-ciphertext pairs and remain with a 2-step iterated EM scheme with a single key (K_2). Thus, we can apply our 2-step iterated EM attack (presented in Section 2.1) for each guess of K_1. However, we note that the preprocessing phase of our $2Round1KeyOpt$ attack should be executed for each guess of K_1, and it is thus now a part of the online algorithm of the attack on LED-128. Moreover, the algorithm can no longer be performed in streaming mode, as we need to reuse each plaintext-ciphertext pair for each guess of K_1. The general framework of the algorithm is given below.

1. Ask for the encryption of D arbitrary plaintexts and store them.
2. For each value of K_1:
   (a) Apply the $2Round1KeyOpt$ attack (including the preprocessing steps) on the resultant scheme, with plaintext-ciphertext pairs $(P_1(m_i \oplus K_1), P_6^{-1}(c_i \oplus K_1))$. Test each returned key using another pair (m_j, c_j).

Using the parameters of our $2Round1KeyOpt$ attack (presented in Section 2.1), the expected data complexity of the attack is 2^{45} known plaintexts and its memory complexity is 2^{60} (required for preprocessing, which is now part of the online algorithm). We calculate the expected time complexity of the algorithm as follows: adding the preprocessing and online time complexities, the main procedure of the attack performed for each guess of K_1 requires about $2^{60.1} + 2^{60} \approx 2^{61.1}$ evaluations of 4 out of the 6 permutations, which is equivalent to about $2^{60.5}$ evaluations of the full scheme. Compared to this complexity, the partial encryption and decryption of each (m_j, c_j) pair, and the trial encryptions using (m_j, c_j) (performed on average once per guess of K_1) are negligible. Thus, the expected time complexity of the attack is about $2^{64+60.5} = 2^{124.5}$, which is about 11 times better than exhaustive search.

3.2 An Attack on 3 Steps of LED-64

We can attack 3-step LED-64 by directly applying $3Round1KeyOpt$ attack with $n = 64$, presented in Section 2.2. Thus, the preprocessing phase has a time complexity of about $2^{58.5}$ and memory complexity of 2^{60}. The online algorithm has a memory complexity of 2^{60}, data complexity of 2^{49} known plaintexts and time complexity of $2^{59.6}$. Since in this paper we consider the preprocessing time as part of the attack (i.e., we assume that we are trying to attack the scheme for the first time), the total time complexity of the algorithm is about $2^{60.2}$, which is about 14 times better than exhaustive search.

3.3 An Attack on 8 Steps of LED-128

We use the same framework of our 6 step attack on LED-128 in order to attack 8 steps of LED-128 (shown in Figure 3). Namely, we guess K_1, and for each guess, we partially encrypt and decrypt the given plaintext-ciphertext pairs and remain with a 3-step iterated EM scheme with a single key (K_2). We then apply our $3Round1KeyOpt$ attack (presented in Section 2.2) for each guess of K_1. Thus, the memory complexity of the attack is 2^{60} and its data complexity is 2^{49} known plaintexts. We calculate the expected time complexity of the algorithm as follows: adding the preprocessing and online time complexities, the main procedure of the algorithm performed for each guess of K_1 requires about $2^{58.5} + 2^{59.6} \approx 2^{60.2}$ evaluations of 6 out of the 8 permutations, equivalent to about $2^{59.8}$ evaluations of the full scheme. Thus, the expected time complexity of the attack is about $2^{64+59.8} = 2^{123.8}$, which is about 18 times better than exhaustive search.

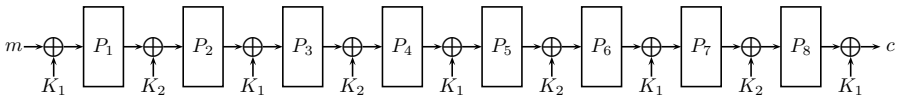

Fig. 3. 8-step LED-128

4 Attacks on 2-Round Iterated Even-Mansour with Independent Keys

The best known generic attack on 2-Round iterated EM with independent keys (see Figure 4) is a MITM attack. This attack is described in Appendix A and it requires 2^n memory and has a time complexity of about $2^{n+1.6}$ full cipher evaluations.

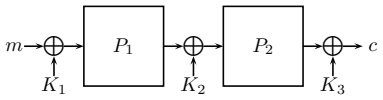

Fig. 4. A 2-round iterated EM with independent keys

In this attack, we use a property of the permutation P_i, which is shared by the keyed permutation $Q_i(K_i, K_{i+1}, x) = P_i(x \oplus K_i) \oplus K_{i+1}$ for any value of K_i and K_{i+1}: these permutations have the same difference distribution table. In order to demonstrate this, consider an entry with the value of t in the difference distribution table of P_i, and denote its input and output differences by

Δ_1 and Δ_2, respectively. Let us denote the t corresponding input-output pairs[8] by $((x_1, y_1), (x_1 \oplus \Delta_1, y_1 \oplus \Delta_2)), \ldots, ((x_t, y_t), (x_t \oplus \Delta_1, y_t \oplus \Delta_2))$. Then, the t input-output pairs $((x_1 \oplus K_1, y_1 \oplus K_2), (x_1 \oplus K_1 \oplus \Delta_1, y_1 \oplus K_2 \oplus \Delta_2)), \ldots, ((x_t \oplus K_1, y_t \oplus K_2), (x_t \oplus K_1 \oplus \Delta_1, y_t \oplus K_2 \oplus \Delta_2))$ correspond to the same entry in the difference distribution table of Q_i (i.e., the entry with input and output differences Δ_1 and Δ_2, respectively).

Using the property above, if we find an entry $[\Delta_1, \Delta_2]$ in the difference distribution table of P_i with a large value, then we can use a similar attack to the one given in [14] on 2-round iterated EM,[9] in order to break the scheme. However, our main observation is that we can find such an entry by preprocessing the public function P_i, which does not need to admit any special property in order to attack the scheme. Thus, our attack adds a preprocessing algorithm to the online algorithm of the attack of [14] (which assumes that we have an entry in the difference distribution table of P_i with a large value). In addition (as we will see later), in the case of independent keys, the basic attack of [14] is not better than exhaustive search, and we will need to add another non-trivial component to this attack. The details of our unoptimized attack $2Round3KeyBasic$ are given below, where S_1, S_2, D are parameters:

Preprocessing:

PR1. Choose an arbitrary input difference $\Delta_1 \neq 0$ and evaluate P_1 on S_1 arbitrary input pairs with input difference Δ_1. For each pair $(x, P_1(x)), (x \oplus \Delta_1, P_1(x \oplus \Delta_1))$, store the output difference $P_1(x) \oplus P_1(x \oplus \Delta_1)$ in a sorted list, next to x.

PR2. Traverse the sorted list and find the most common output difference Δ_2 (if there are several options for Δ_2, choose one arbitrarily). Keep only the entries of the list which correspond to pairs with the output difference of Δ_2 (assume that we have t such pairs). For each such entry, recalculate and store the full pair $(x, P_1(x)), (x \oplus \Delta_1, P_1(x \oplus \Delta_1))$.

PR3. Evaluate P_2 on S_2 arbitrary input pairs with input difference Δ_2. For each pair $(y, P_2(y)), (y \oplus \Delta_2, P_2(y \oplus \Delta_2))$, store the output difference $P_2(y) \oplus P_2(y \oplus \Delta_2)$ in a sorted list L_2, next to y.

Online:

O1. Ask for the encryption of D arbitrary input pairs with difference Δ_1.

O2. For each pair of plaintext-ciphertext pairs $((m_i^1, c_i^1), (m_i^2 = m_i^1 \oplus \Delta_1, c_i^2))$:
   (a) Search for the output difference $c_i^1 \oplus c_i^2$ in L_2, (if there is no match, discard the pair and return to Step O2).
   (b) For each match $(y, P_2(y)), (y \oplus \Delta_2, P_2(y \oplus \Delta_2))$, we have 2 candidates for K_3: $P_2(y) \oplus c_i^1$ and $P_2(y) \oplus c_i^2$. We also have $2t$ candidates for K_1: the candidates $x \oplus m_i^1$ and $x \oplus m_i^2$ for each of the t values of x. As each pair of values for K_1 and K_3 suggests a value for K_2, we have $4t$ suggestions of the full key to test using another plaintext-ciphertext pair.

[8] In this paper, we consider unordered pairs, i.e., $((x, y), (u, v))$ and $((u, v), (x, y))$ are considered the same pair.

[9] Although the attack of [14] was previously applied to 2-round iterated EM with one key, it can be adapted to work for the case of independent keys.

Similarly to our analysis of random functions, assuming that P_1 is a random permutation, then each entry in its difference distribution table is distributed according to the Poisson distribution [16].[10] This will allow us to easily determine the expected value of t and use it in order to analyze the expected complexity of our algorithm.

The memory complexity of the preprocessing phase is $max(S_1, S_2)$, and its time complexity is $2 \cdot S_1$ evaluations of P_1 and $2 \cdot S_2$ evaluations of P_2, or $S_1 + S_2$ evaluations of the full scheme. The memory complexity of the online algorithm is S_2. Using the birthday paradox, out of the D plaintext-ciphertext pairs evaluated in the online phase, at least $(D \cdot t)/2^{n-1}$ are expected to have a difference of Δ_2 after P_1 (note that we have 2^{n-1} unordered pairs with a given difference). Using the same argument, we expect that $(D \cdot t \cdot S_2)/2^{2(n-1)}$ of them will match the pairs evaluated for P_2 during proprocessing. Thus, we require that $(D \cdot t \cdot S_2)/2^{2(n-1)} = 1$, or $D = 2^{2(n-1)}/(t \cdot S_2)$ in order to find the key with high probability. Without going into the details of the time complexity analysis, note that we are using only two plaintext-ciphertext pairs to filter the key suggestions, tested in Step O2.(b). As we have $3n$ bits of key and $2n$ bits of filtering, we need to test at least 2^n keys in Step O2.(b), and thus the attack is not faster than the simple MITM attack on this scheme.

4.1 A Time-Optimized Attack on 2-Round Iterated Even-Mansour

In order to improve the attack, we need to add more filtering conditions, and thus we actually work on triplets, as described in the improved algorithm $2Round3KeyOpt$:

Preprocessing:

PR1. Choose an arbitrary input difference $\Delta_1 \neq 0$ and evaluate P_1 on S_1 arbitrary input pairs with input difference Δ_1. For each pair $(x, P_1(x)), (x \oplus \Delta_1, P_1(x \oplus \Delta_1))$, store the output difference $P_1(x) \oplus P_1(x \oplus \Delta_1)$ in a sorted list, next to x.

PR2. Traverse the sorted list and find the most common output difference Δ_2 (if there are several options for Δ_2, choose one arbitrarily). Keep only the entries of the list which correspond to pairs with the output difference of Δ_2 (assume that we have t such pairs). For each such entry recalculate and store the full pair $(x, P_1(x)), (x \oplus \Delta_1, P_1(x \oplus \Delta_1))$ in a list L_1.

PR3. Choose another non-zero input difference Δ_1'. For each value x stored in L_1, evaluate P_1 an additional time to obtain the pair $(x \oplus \Delta_1', P_1(x \oplus \Delta_1'))$. Store the (total of) additional t output differences $P_1(x) \oplus P_1(x \oplus \Delta_1')$ in a separate sorted list of differences, L_1'.

PR4. Evaluate P_2 on S_2 arbitrary input pairs with input difference Δ_2. For each pair $(y, P_2(y)), (y \oplus \Delta_2, P_2(y \oplus \Delta_2))$, store the output difference $P_2(y) \oplus P_2(y \oplus \Delta_2)$ in a sorted list L_2, next to y.

[10] However, we note that since we consider unordered pairs, then we have only 2^{n-1} possible pairs of a given difference, and each pair can attain (almost) all 2^n output differences

Online:

O1. Ask for the encryption of D arbitrary input triplets of the form m, $m \oplus \Delta_1$ and $m \oplus \Delta'_1$ (for D arbitrary values of m).
O2. For each pair of plaintext-ciphertext pairs $((m_i^1, c_i^1), (m_i^2 = m_i^1 \oplus \Delta_1, c_i^2))$:
   (a) Search for the output difference $c_i^1 \oplus c_i^2$ in the list L_2 (if there is no match, discard the pair and return to Step O2).
   (b) For each match $(y, P_2(y)), (y \oplus \Delta_2, P_2(y \oplus \Delta_2))$, compute the 2 candidates for K_3: $K'_3 = P_2(y) \oplus c_i^1$ and $K''_3 = P_2(y) \oplus c_i^2$.
   (c) Denote the third plaintext-ciphertext pair in the triplet by $(m_i^3 = m_i^1 \oplus \Delta'_1, c_i^3)$. Compute $y' = P_2^{-1}(c_i^3 \oplus K'_3)$ and $y'' = P_2^{-1}(c_i^3 \oplus K''_3)$.
   (d) Search L'_1 for the four possibilities of the third difference obtained at this stage: $y' \oplus y$, $y' \oplus \Delta_2 \oplus y$, $y'' \oplus y$, $y'' \oplus \Delta_2 \oplus y$ (if there is no match, discard the pair and return to Step O2).
   (e) Test the $4t$ suggestions of the full key using (m_i^3, c_i^3). If the test succeeds, return the key.

The time and memory complexities of the preprocessing phase are similar to those of the $2Round3KeyBasic$ attack (the additional t evaluations of P_1 and t units of storage are negligible). Using the calculation done for $2Round3KeyBasic$, the online algorithm requires $D = 2^{2(n-1)}/(t \cdot S_2)$ plaintext-ciphertext triplets. For each processed triplet, we expect to find a match in L_2 with probability $S_2/2^n$. For each such matched triplet, we need to compute $P_2(y)$ (in order to compute K'_3 and K''_3) and evaluate P_3^{-1} twice in order to compute y' and y''. Once we do so, the probability of a match in L'_1 in Step O2.(d) is proportional to $t/2^n$. This is a negligible probability, and thus we can neglect the complexity of the trial encryptions in Step O2.(e). Thus, the online time complexity (without counting the data) is about $3 \cdot D \cdot S_2/2^n = 0.75 \cdot 2^n/t$ evaluations of P_2, or $0.375 \cdot 2^n/t$ evaluations of the full scheme.

The data complexity of the attack is D triplets, or $3D$ chosen plaintexts. However, we can easily reduce it to $2D$ by requesting encryptions of structures containing the messages m, $m \oplus \Delta_1$, $m \oplus \Delta'_1$ and $m \oplus \Delta_1 \oplus \Delta'_1$. Each such structure of 4 plaintexts contains two triplets which we can exploit, implying that the data complexity of the attack is indeed $2D$. If we add the time to generate the data to the time complexity, we get that the total time complexity of the online attack is about $2D + 0.375 \cdot 2^n/t$ evaluations of the full scheme.

4.2 Applications to Full AES^2

AES^2 is a 128-bit block cipher presented at Eurocrpyt 2012 [5]. The cipher is a 2-round iterated EM construction, where each of the public permutations P_1 and P_2 is based on an invocation of full AES-128 with a pre-fixed and publicly known key. The designers of the scheme claim that its security is 2^{128}. However, the best attack known to the designers (as claimed in [5]) is the MITM attack presented in Appendix A, and based on our analysis, it has a slightly higher time complexity of $3 \cdot 2^{128} \approx 2^{129.6}$ and a memory complexity of 2^{128}.

In order to attack AES^2, we use our $2Round3KeyOpt$ attack with $S_1 = 2^{124}$ and $S_2 = 2^{125.4}$. This implies that the memory complexities of both the preprocessing and online phases is $2^{125.4}$. The time complexity of the preprocessing phase is $S_1 + S_2 = 2^{124} + 2^{125.4} \approx 2^{125.9}$ evaluations of the full scheme. Using the formula $(2^n \cdot \lambda^t \cdot e^{-\lambda})/t!$ with $\lambda = 2^{124}/2^{128} = 1/16$ and $t = 18$, it is easy to check that we expect to find at least 10 entries in the difference distribution table with a value of 18 (we need only one). Plugging in these values into the formula $D = 2^{2(n-1)}/(t \cdot S_2)$, we obtain $D \approx 2^{124.4}$, implying that the data complexity of the attack is $2^{125.4}$ chosen plaintexts. The time complexity of the online attack is $2D + 0.375 \cdot 2^n/t \approx 2^{125.6}$, and adding the preprocessing time, the total time complexity of the algorithm is about $2^{125.9} + 2^{125.6} \approx 2^{126.8}$. This is better than the $2^{129.6}$ time complexity of the MITM attack by a factor of about 7, and clearly violates the 128-bit security claimed for AES^2 in [5]. We also note that the memory complexity is improved from 2^{128} to about $2^{125.4}$, however the data complexity is greatly increased to $2^{125.4}$.

5 Conclusions

In this paper we considered several schemes which are based on the iterated Even-Mansour scheme, and improved their best known attacks. For the recommended values of n our attacks are between 7 and 20 times faster than exhaustive search, but they differ from other improvements of exhaustive search since their improvement factor is about $n/\log(n)$, which increases to infinity as n grows. In particular, we described the first attack on the full AES^2, and improved the number of steps which can be attacked in the well known LED-128 block cipher from 6 to 8. Even though our attacks are not likely to be practically significant, they indicate that block ciphers based on the EM scheme with one key should have at least 4 rounds, regardless of how strong we make the internal permutations.

References

1. Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the Indifferentiability of Key-Alternating Ciphers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 531–550. Springer, 2013.
2. Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.
3. Alex Biryukov and David Wagner. Slide Attacks. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.
4. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.

5. 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 Pointcheval and Johansson [17], pages 45–62.
6. Joan Daemen. Limitations of the Even-Mansour Construction. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT*, volume 739 of *Lecture Notes in Computer Science*, pages 495–498. Springer, 1991.
7. Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In Pointcheval and Johansson [17], pages 336–354.
8. Shimon Even and Yishay Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. *J. Cryptology*, 10(3):151–162, 1997.
9. Philippe Flajolet and Andrew M. Odlyzko. Random Mapping Statistics. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, 1989.
10. Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block Ciphers That Are Easier to Mask: How Far Can We Go? In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2013.
11. 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.
12. Takanori Isobe 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.
13. Rodolphe Lampe, Jacques Patarin, and Yannick Seurin. An Asymptotically Tight Security Analysis of the Iterated Even-Mansour Cipher. In Wang and Sako [18], pages 278–295.
14. Florian Mendel, Vincent Rijmen, Deniz Toz, and Kerem Varici. Differential Analysis of the LED Block Cipher. In Wang and Sako [18], pages 190–207.
15. Ivica Nikolić, Lei Wang, and Shuang Wu. Cryptanalysis of Round-Reduced LED. In *FSE*, 2013. To appear in Lecture Notes in Computer Science.
16. Luke O'Connor. On the Distribution of Characteristics in Bijective Mappings. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 360–370. Springer, 1993.
17. David Pointcheval and Thomas Johansson, editors. *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*. Springer, 2012.
18. 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 A Meet-in-the-Middle Attack on 2-Round Iterated Even-Mansour with Independent Keys

Unlike iterated EM schemes with one key, where the time complexity of exhaustive search is 2^n, 2-Round iterated EM with independent keys has $3n$ key bits

(see Figure 4) and thus straightforward exhaustive search has a time complexity of 2^{3n}. It is very easy to improve the time complexity of the attack to 2^{2n} by noticing that $K_3 = P_2(P_1(m \oplus K_1) \oplus K_2) \oplus c$, and thus guessing K_1 and K_2 suffices in order to compute K_3. A further simple improvement is to use a meet-in-the-middle attack in order to reduce the time complexity to about 2^n, at the expense of using about 2^n memory. Such a MITM attack on an iterated EM scheme with an arbitrary number of rounds is given in [5], and we describe it here for $r = 2$ (under the name $2Round3KeyMITM$) for the sake of completeness.

1. Obtain 3 known plaintext-ciphertext pairs — (m_1, c_1), (m_2, c_2), (m_3, c_3).
2. For each value of K_1, compute $\Delta_1 = P_1(m_1 \oplus K_1) \oplus P_1(m_2 \oplus K_1)$, and store (Δ_1, K_1) in a list sorted according to Δ_1.
3. For each value of K_3:
   (a) Compute $\Delta_3 = P_2^{-1}(c_1 \oplus K_3) \oplus P_2^{-1}(c_2 \oplus K_3)$, and search for Δ_3 in the sorted list.
   (b) For each match, test the key K_1, $K_2 = P_1(m_1 \oplus K_1) \oplus P_2^{-1}(c_1 \oplus K_3)$, K_3 using (m_3, c_3), and if the tests succeeds return the key.

The memory complexity of the algorithm is 2^n, required for storing the list in Step 2. The time complexity of Step 2 is $2 \cdot 2^n$ evaluations of P_1. We expect a single match in Step 3.(a) for a value of K_3, and hence the expected time complexity of Step 3 is $2 \cdot 2^n$ evaluations of P_1 and 2^n evaluations of the full cipher. The expected time complexity of the attack is thus $3 \cdot 2^n \approx 2^{n+1.6}$. We note that it is possible to trade some memory at the expense of time. However, in this paper we are mainly interested in the attack with the lowest time complexity.

B A Low-Memory Attack on 2-Round Iterated Even-Mansour with One Key

In this section, we present another attack on 2-round iterated EM with one key. The attack is different from all the other attacks described in this paper, since it does not exploit vertices with a high in-degree in random functions. In fact, the attack does not assume any property of the public permutations, and thus is expected to work even for those which are highly non-random. The advantage of the attack compared to the attacks of Section 2.1, is that it requires significantly less memory during preprocessing (if fact, the preprocessing computation of the attack is negligible). On the other hand, it requires more data than the data-efficient attack of of Section 2.1, and in addition it requires chosen plaintexts (rather than known plaintexts).

In order to simplify our notation, let v be a n-bit vector and X a set of n-bit vectors. We define the operation $v \oplus X \triangleq \{v \oplus x | x \in X\}$. We now describe a generalization of the filtering technique that we used in our optimized attack in Section 2.1: consider an arbitrary message m and an affine subspace X of dimension d_1, which is defined with a set of $n - d_1$ linear equations. If $m \oplus K \in X$, then $m \oplus K$ satisfies the $n - d_1$ linear equations which define X. Similarly to the

observation used in Section 2.2, if we know m, then we know the value of these linear equations on K. More generally, given an affine subspace V of dimension d_2, if there exists a message $m \in V$ such that $m \oplus K \in X$, then the intersection of the subspaces $K \oplus V$ and X is non-empty, and this imposes $n - d_1 - d_2$ linear equations on K. Our attack is based on this generalized filtering technique. Let S and D be parameters, such that $S + \log(S) < n$:

Preprocessing:

PR1. Evaluate P'_1 on the subspace of inputs X of size S, which contains all the vectors whose $n - \log(S)$ LSBs are set to zero, and the $\log(S)$ MSBs attain all S possible values. Store the S outputs $v_1, v_2, \ldots, v_S$. Denote by V the linear vector space of dimension (at most) S spanned by $v_1, v_2, \ldots, v_S$.

PR2. Compute and store the (minimum of) $n - \log(S) - S$ linear equations imposed on K assuming that $(K \oplus V) \bigcap X \neq \emptyset$. Denote the computed matrix by A.

Online:

O1. Ask for the encryption of D structures (affine subspaces of dimension S) of plaintexts, defined by $m \oplus V$, where m is an arbitrary message.

O2. For each structure $m_i \oplus V$:
   (a) Assume that $((m_i \oplus K) \oplus V) \bigcap X \neq \emptyset$, and use m_i to compute the values of the linear equations of A on K.
   (b) For each $m_i^j \in m_i \oplus V$, compute $z_i^j \triangleq P_2(m_i^j)$, and use it to compute the value of the linear equations of A on z_i^j. Store these 2^S values in a sorted list L_1, next to z_i^j.
   (c) For each ciphertext c_i^j in the structure, compute the value of the linear equations of A on $c_i^j \oplus K$ (their value on K is known from Step O2.(a)). Search for matches of this value in L_1.
   (d) For each match associated with c_i^j and z_i^ℓ (for some value of ℓ), obtain a suggestion for the key $K' = c_i^j \oplus z_i^\ell$, and test it using one plaintext-ciphertext pair. If the test succeeds, return the key.

In order to understand the attack, assume that indeed $((m_i \oplus K) \oplus V) \bigcap X \neq \emptyset$ for some m_i (as assumed in Step O2.(a)), i.e. $(m_i \oplus K \oplus v) \in X$ for some $v \in V$. Then, we know that $P'_1(m_i \oplus K \oplus v) \in V$, and thus $Q'_1(K, m_i \oplus v) = P'_1(m_i \oplus K \oplus v) \in V$, i.e., $Q_1(K, m_i \oplus v) = m_i \oplus v \oplus v'$ for some $v' \in V$. Since V is a linear subspace, it implies that $Q_1(K, m_i \oplus v) \in m_i \oplus V$. As we compute P_2 for all $m_i^j \in m_i \oplus V$ in Step O2.(b), we will get a match in step 2.(c) with the ciphertext of our plaintext $m_i \oplus v$, and this match will suggest the correct key.

The time complexity of the preprocessing phase is S evaluations of P_1 in addition to some linear algebra of complexity of about n^3 bit operations in Step PR2 (which can be performed be Gaussian elimination). As calculated shortly, these computations are negligible compared to the computations performed in the online phase of the attack. The memory complexity of the preprocessing phase is also proportional to S, with the addition if about n^2 bits required

in order to store A. Again, this storage is negligible compared to the storage required by the online algorithm.

The memory complexity of the online algorithm is about 2^S, required to store L_1 (note that we can work on the ciphertexts of each structure in streaming mode, and thus we do not need to store them in memory). The attack finds the correct key once $((m_i \oplus K) \oplus V) \bigcap X \neq \emptyset$ for some m_i. Since $|X| = S$, we need about $2^n/S$ data for this to occur with high probability, i.e., we require $D \approx 2^n/(S \cdot 2^S)$ structures of 2^S plaintexts. For each such structure, we perform 2^S evaluations of P_2. In order to calculate the expected number of trial encryptions performed in Step O2.(d) for each structure, we notice that we match two lists of size 2^S (although only L_1 is explicitly stored in memory). The number of matched bits is (at least) $n - \log(S) - S$, and thus the expected number of trial encryptions per structure is $2^{2S-n+\log(S)+S} = 2^{3S+\log(S)-n}$.

If we assume that $S = n/4$, then the time complexity of the trial encryptions is negligible compared to the complexity of evaluations of P_2, which is equivalent in total to about $0.5 \cdot 2^n/S$ encryptions. Adding the additional $2^n/S$ encryptions required to prepare the data, the total time complexity of the attack is equivalent to about $2^{n+0.6}/S$ encryptions.

We note that there are several possible extensions to this algorithm which include a more complicated preprocessing phase. In particular, we can combine this attack with the techniques of Section 2.1 by looking for subsets X whose elements have a large in-degree in the associated graph of P_1'. However, we do not describe these extensions in this paper.

Concrete Parameters For $n = 64$, let $S = 16$. Then, the memory complexity of the attack is 2^{16}, its expected data complexity is 2^{60} chosen plaintexts, and its expected time complexity is about $2^{60.6}$.

C Extending the attacks to Iterated Even-Mansour Schemes with Linearly-Dependent Keys

It is important to note that all of our attacks on 2-round and 3-round EM schemes with one key can be generalized to iterated EM schemes with linearly-dependent keys. Let us denote the first key as $K_1 = K$, and thus $K_2 = L_2(K)$, $K_3 = L_3(K)$, and in 3-round iterated EM, $K_4 = L_4(K)$, where L_2, L_3, L_4 are linear mappings. In order to attack such schemes, all of the steps in our algorithms which are based on the equality of the keys should be modified to handle the case where the keys are linearly dependent. This can be easily done using simple linear algebra without increasing the complexity of the algorithms. For example, our attacks on iterated EM schemes with one key are based on the property $x \oplus Q(K, x) = x \oplus K \oplus P(x \oplus K)$. In the more general scheme, we have $Q^*(K, x) = L(K) \oplus P(x \oplus K)$ (for some linear mapping L and permutation P), and it implies that $L(x) \oplus Q^*(K, x) = L(x \oplus K) \oplus P(x \oplus K)$. Thus, if we know that some values of the function $P''(z) \triangleq L(z) \oplus P(z)$ are more likely

than expected (by finding t-way collisions in this function), then we can predict the value of $Q^*(K, x)$ with a higher probability than expected, and exploit this property in the same way as in our attacks on schemes with one key.