Differential MITM attacks on SLIM and LBCIoT

Peter Grochal^{*} Martin Stanek[†]

Department of Computer Science Faculty of Mathematics, Physics and Informatics Comenius University

Abstract

SLIM and LBCIoT are lightweight block ciphers proposed for IoT applications. We present differential meet-in-the-middle attacks on these ciphers and discuss several implementation variants and possible improvements of these attacks. Experimental validation also shows some results that may be of independent interest in the cryptanalysis of other ciphers. Namely, the problems with low-probability differentials and the questionable accuracy of standard complexity estimates of using filters.

1 Introduction

We observe increasing applications of devices with limited resources, such as sensors, RFID tags, and IoT devices. When data confidentiality, integrity, or authenticity is required, cryptographic algorithms should be used. Constrained environments, whether it's memory, computation power, or battery, make conventional algorithms impractical in these scenarios. Lightweight cryptography aims to address this issue and provide cryptographic algorithms that can be implemented and operated on devices with limited resources. The importance of lightweight cryptography is also highlighted in a standardization effort in the ISO/IEC 29192 series, and the NIST standardization process [12].

There are many different proposals for lightweight cryptographic algorithms, and new ones are proposed every year [3, 6, 11]. Since these algorithms are optimized for resource consumption, achieving cryptographic resilience is challenging. Hence, it is essential to analyze them thoroughly.

Our contribution and related work

We study two lightweight block ciphers: SLIM [1], and LBCIoT [7]. These ciphers were analyzed using differential cryptanalysis [5], and various cryptanalytic attacks were recently tried in [10] – impossible differential, integral, etc. We apply differential meet-in-the-middle (MITM) cryptanalysis on SLIM and LBCIoT. This attack was recently proposed as an extension of classical differential attack and was successfully used to attack multiple block ciphers [2, 4, 8]. The main contribution of this paper:

^{*}pegro@protonmail.com

[†]martin.stanek@fmph.uniba.sk

- We present the first differential MITM attacks on SLIM and LBCIoT and compare them with the best-known attacks on these ciphers. Our attacks on 25 and 26-round LBCIoT are the best partial key recovery attacks on LBCIoT reported to date.
- We describe and experimentally verify a problem with low-probability differentials used for cryptanalysis. In this case, the classical differential and the differential MITM attacks do not work for a significant portion of encryption keys.
- We show that commonly used assumptions for complexity estimates of cryptanalytic attacks are not always justified. We confirm this observation experimentally for reduced variants of SLIM and LBCIoT.

The remainder of this paper is organized as follows. Section 2 introduces a notation and high-level overview of differential MITM cryptanalysis. We describe SLIM and LBCIoT in Section 3. The first part of Section 4 provides information about classical differential cryptanalysis of SLIM and LBCIoT, and the problems we identified for low-probability differentials. The main part of the section shows our differential MITM attack and its components: useful bits and deterministic bits. The last section contains the results from the experimental verification of the attack.

2 Preliminaries

Let $E, D : \{0,1\}^l \times \{0,1\}^n \to \{0,1\}^n$ be the encryption and decryption functions of a block cipher with the key length l and the block size n, i.e., for any key $k \in \{0,1\}^l$ and plaintext block $x \in \{0,1\}^n$ we have D(k, E(k, x)) = x. In the rest of the paper we assume E can be split into three consecutive transformations $E = E_{out} \circ E_m \circ E_{in}$, i.e., first encrypting a plaintext block with E_{in} , then E_m , and finishing with E_{out} . For iterated block ciphers, e.g., SLIM and LBCIoT, various splits are possible using subsequent rounds. In such case, we denote the number of rounds for the transforms by r_{in}, r_m , and r_{out} .

A difference of two *n* bit vectors x_1, x_2 is a bitwise xor of these vectors, denoted as $x_1 \oplus x_2$. We are interested in (sufficiently) high-probability differentials spanning the middle function $E_{\rm m}$. A differential $\Delta = (\alpha \to \beta)$ is a pair of input and output differences. The probability of the differential $\Delta = (\alpha \to \beta)$ is given by this formula:

$$\Pr[E_{\mathrm{m}}(k,x) \oplus E_{\mathrm{m}}(k,x \oplus \alpha) = \beta; \text{ for random } k \in \{0,1\}^{l} \text{ and } x \in \{0,1\}^{n}].$$
(1)

We show in Section 4.1 that for practical application of both differential attack and differential MITM attack it is important to distinguish whether (1) is calculated for a random or fixed k. Even for high-probability differentials we might observe large subset of keys, for which the probability is zero, resulting in attack failure.

Given a differential $\Delta = (\alpha \to \beta)$, we denote by k_{in}^{Δ} the set of key bits that are sufficient to compute a plaintext block \tilde{P} from any block P, such that $E_{\text{in}}(P) \oplus E_{\text{in}}(\tilde{P}) = \alpha$. Similarly, k_{out}^{Δ} is the set of key bits that are sufficient to compute a ciphertext block \tilde{C} from any block C, such that $E_{\text{out}}^{-1}(C) \oplus E_{\text{out}}^{-1}(\tilde{C}) = \beta$. See Figure 1 for visual representation of these concepts.

The notion of deterministic bits allows us to detect situations when a particular differential certainly did not occur, thus making the attack faster. Deterministic bits for a differential $\Delta = (\alpha \rightarrow \beta)$ and E_{out} are a subset of bits in a ciphertext block whose difference is constant, provided that the difference before E_{out} is β . We denote this difference δ^{Δ} .

Since the attacks usually employ a single differential, we omit writing Δ if it is obvious from the context, e.g., for k_{in}^{Δ} , k_{out}^{Δ} or δ^{Δ} .



Figure 1: Splitting a cipher for differential MITM attack

2.1 Overview of differential MITM cryptanalysis

Let us split E into three transformations $E_{\text{out}} \circ E_{\text{m}} \circ E_{\text{in}}$. Let $\Delta = (\alpha \to \beta)$ be a differential for E_{m} with probability 2^{-p} . The differential MITM attack is a chosen plaintext attack. The main idea of the attack is to find any correct pair of plaintexts P, \tilde{P} , with corresponding ciphertexts C, \tilde{C} , such that the differential Δ occurs for E_{m} . The successful search will reveal candidate key bits for k_{in} and k_{out} .

We find such correct pair P, \tilde{P} using MITM approach (a precise description with deterministic bits is provided in Algorithm 1 in Section 4.2):

- 1. For a fixed random P, we guess value i for $k_{\rm in}$, and compute \tilde{P} such that the difference of P and \tilde{P} after the transformation $E_{\rm in}$ results in α .
- 2. We ask for the ciphertext C corresponding to P and ciphertext \hat{C} corresponding to \tilde{P} . This is where chosen plaintext oracle is used. We store the pair (\hat{C}, i) using a hash table.
- 3. Independently, we guess value j for k_{out} , and compute \tilde{C} from C such that their difference after E_{out}^{-1} is β . Then, for each pair (\tilde{C}, i) found in the hash table, we get a candidate combination (i, j) for k_{in} , and k_{out} .

All candidate combinations have the property that the desired differences are achieved after $E_{\rm in}$ and $E_{\rm out}^{-1}$. This is certainly also true for any correct pair P, \tilde{P} . Hence, the correct key bits are among our candidates.

Since the probability of the differential Δ is 2^{-p} , we repeat the procedure 2^p times so that we can expect the differential to occur for the correct (i, j) values. There is a possibility to optimize the search procedure if k_{in} and k_{out} overlap.

The differential MITM cryptanalysis shares some similarities with a classical differential cryptanalysis, see [8] for a detailed discussion and comparison.

3 SLIM and LBCIoT

SLIM and LBCIoT are designed as lightweight 32-bit block ciphers aimed at RFID in case of SLIM [1], and IoT devices in case of LBCIoT [7]. We describe those parts of the ciphers that are relevant for the subsequent cryptanalysis.

3.1 SLIM

SLIM is a Feistel cipher with 32-bit block divided into left and right 16-bit halves. The cipher uses a 80-bit key to derive 32 round keys, each one 16 bits long. The round function of SLIM is depicted in Figure 2. The cipher begins with initializing the state with the plaintext block (L_0, R_0) , and follows by iterating the round function 32 times using corresponding round keys.



Figure 2: SLIM round function

The round function uses the following operations to transform the state (L_i, R_i) into (L_{i+1}, R_{i+1}) :

- Xor-ing round key K_i with a copy of R_i . Being a Feistel cipher, SLIM does not change R_i , which becomes L_{i+1} in the next round.
- Parallel substitution using a fixed 4×4 substitution box S:

x	0	1	2	3	4	5	6	7	8	9	А	В	С	D	Е	\mathbf{F}
S(x)	С	5	6	В	9	0	А	D	3	Е	F	8	4	7	1	2

• Permuting bits with a fixed permutation P; the proposal was not clear regarding the bit order, we follow implementation used in [5], where 0 refers to the least significant bit (the rightmost bit or, following Figure 2, the bit at the bottom):

- Xor-ing the result with L_i .
- Swapping the left and right halves of the state.

Remark. Since SLIM is a Feistel cipher, the decryption is performed using the same round function with round keys applied in reverse order. In order to make this work, the last round of the cipher should omit the final swap of halves. However, the proposal [1] does not mention this at all.

The key schedule of SLIM has the following structure: the 80-bit master key is divided into the first five 16-bit round keys, and the remaining round keys are computed by a deterministic algorithm. This algorithm was not clearly defined in [1]. Our analysis does not use any weaknesses of the key scheduling after the first five rounds. Therefore, we omit its description. If there are any exploitable deficiencies in the key scheduling, they can improve our attacks.

3.2 LBCIoT

LBCIoT shares some similarities with SLIM: 80-bit key, 32-bit block, 32 rounds, and similar internal components. It also splits the block into two 16-bit halves. Even though the round function of LBCIoT resembles a Feistel cipher, it is not – both halves are transformed in each round, see Figure 3.



Figure 3: LBCIoT round function

The round function uses the following operations to transform the state (L_i, R_i) into (L_{i+1}, R_{i+1}) :

• A cyclical rotation of R_i 's copy by 7 bits, followed by a parallel substitution using a fixed 4×4 substitution box S:

																\mathbf{F}
S(x)	0	8	6	D	5	F	7	С	4	Е	2	3	9	1	В	А

- Xor-ing the result with L_i and a round key K_i .
- Left and right halves are permuted using a fixed permutations P_1 and P_2 for the left half and the right half, respectively. Similarly to SLIM, the proposal was not clear

regarding the bit order. We choose the order compatible with differentials from [5]. In the following tables 0 refers to the least significant bit:

x	15	14	13	12	11	10	9	8	$\overline{7}$	6	5	4	3	2	1	0
$P_1(x)$	7	10	5	0	3	14	15	4	1	2	13	8	11	6	9	12
$P_2(x)$	9	6	14	8	5	13	0	3	12	1	10	2	11	15	7	4

• Swapping the left and right halves of the state.

The decryption round function must invert the encryption round. Starting with inverse permutations P_1^{-1} , P_2^{-1} . The inner transformation with rotation and substitutions is the same, and the order of round keys is reversed.

The key schedule of LBCIoT starts, similarly to SLIM, by dividing the 80-bit master key into the first five 16-bit round keys. The computation of other round keys is unclear from the proposal [7]. However, our attacks do not use any weaknesses of the key scheduling after the first five rounds.

4 Attacking SLIM and LBCIoT

4.1 Differentials and prior attacks

SLIM and LBCIoT were analyzed and attacked in [5] using a differential cryptanalysis. The authors found suitable differentials using CryptoSMT tool [9]. Tables 1 and 2 summarize a selection of their differentials that we use for our subsequent attacks, see Appendix A for a more extensive list of differentials. We have experimentally verified the probability of these differentials. More precisely, we have confirmed the probability of ciphertexts having the difference β , given a random key, and a random plaintext pair with difference α , see also (1) in Section 2.

r	α	eta	2^{-p}	r	α	eta	2^{-p}
11	4827 0080	0020 08b4	2^{-26}	16	0006 0400	0020 1000	2^{-26}
12	0b82 000a	0a00 801b	2^{-28}	17	0006 0400	0100 2040	2^{-28}
13	a208 a000	a000 b208	2^{-31}	18	6000 0040	0080 0000	2^{-30}
Г	able 1: Differ	centials for SI	LIM	Ta	ble 2: Differe	ntials for LB	CIoT

The authors of differential cryptanalysis of SLIM and LBCIoT claim [5] the following results, all attacking reduced versions of the ciphers:

- 13-round SLIM (using a 12-round differential): Reconstruction of 1-round subkey with probability 98.2% and time complexity proportional to 2^{31} encryptions.
- 14-round SLIM (using a 13-round differential): Using a full codebook as chosen plaintexts, the final round key can be obtained with probability 63.2% and time complexity proportional to 2^{32} encryptions.
- 18-round LBCIoT (using a 17-round differential): The attack reconstructs the final round subkey with probability 98.2% and time complexity proportional to 2^{31} encryptions.

• 19-round LBCIoT (using a 18-round differential): Time complexity is proportional to 2^{31} encryptions, and the final round key can be found with probability 63.2%.

4.1.1 The problem with low-probability differentials

The probability of differential defined as in (1) does not guarantee the desired property for each fixed key that we aim to reconstruct in the classical differential and differential MITM attacks. This situation is especially problematic for differentials with small probabilities (close to 2^{-n}), where selection of plaintexts is no more independent when almost entire plaintext space is exhausted.

Let us illustrate this problem on SLIM, where the authors [5] employ a 12-round differential with probability 2^{-28} , and a 13-round differential with probability 2^{-31} . We propose a simple experiment where a random key is tested on the entire plaintext space to find out, whether the desired difference occurs at least once. The results presented in Table 3 show that for a substantial portion of keys – each differential was tested with 10.000 random keys – the output difference was never observed. For such keys, the classical differential cryptanalysis and the differential MITM cryptanalysis can never succeed. The results of similar experiment for LBCIoT are presented in Table 4.

r	α	eta	2^p	% of keys
11	4827 0080	0020 08b4	2^{-26}	0%
12	0b82 000a	0a00 801b	2^{-28}	30.06%
13	a208 a000	a000 b208	2^{-31}	40.61%

Table 3: Percentage of tested keys for which the differential never occurs in SLIM

r	α	eta	2^p	% of keys
16	0006 0400	0020 1000	2^{-26}	0%
17	0006 0400	0100 2040	2^{-28}	0.01%
18	6000 0040	0080 0000	2^{-30}	5.95%

Table 4: Percentage of tested keys for which the differential never occurs in LBCIoT

4.2 Differential MITM attack

This section presents the differential MITM attack with deterministic bits, its components and complexity. We show how these are relevant for SLIM and LBCIoT, and what parameters we obtain when analyzing these ciphers.

4.2.1 Computation of k_{in} and k_{out} – useful bits

Before we explain the attack itself, we need to address how k_{in} and k_{out} are computed for given difference α and β , respectively. We want to identify *useful* bits. A key bit is called useful, if it cannot be omitted in k_{in} or k_{out} . More precisely, a bit (position) is useful for k_{in} if there exists a key k and a plaintext P, such that flipping that bit in the key k alters the result of $E_{\text{in}}^{-1}(k, (E_{\text{in}}(k, P) \oplus \alpha))$. Similarly, a useful bit (position) for k_{out} alters the result of $E_{\text{out}}(k, (E_{\text{out}}^{-1}(k, C) \oplus \beta))$, for some k and C, when the key bit is flipped.

A simple general procedure to find useful bits was proposed in [4, Appendix A]. Note that the algorithm is probabilistic, and may not detect some of the rarely-useful bits. Tables 5 and 6 show masks for useful bits for SLIM and LBCIoT, with various numbers of rounds considered for $E_{\rm in}$, $E_{\rm m}$, and $E_{\rm out}$. Let us remind that the number of rounds $r_{\rm m}$ for $E_{\rm m}$ determines the differential used in useful bits calculation. We use a single differential for each distinct r_m , see Appendix A.

Remark. Another option is to look at the structure of a cipher, and see which bits are guaranteed to not be useful. This is usually straightforward, but obtained sets of remaining bits might be larger than k_{in} and k_{out} obtained by the algorithm.

$r_{\rm in}$	$r_{\rm m}$	$r_{\rm out}$	$k_{\texttt{in}} \max$	$k_{\texttt{out}}$ mask	$ k_{\rm in} $	$ k_{\mathrm{out}} $
1	4	1	e000	f007	3	7
2	4	2	ffff e000	f007 ffff	19	23
1	6	1	0d07	efb7	6	13
1	8	1	f0db	0df0	10	7
2	11	3	ffff b7df	07fb ffff ffff	29	42
3	11	2	ffff ffff b7df	07fb ffff	45	26
3	13	2	ffff ffff fd07	fd07 ffff	42	26

Table 5: Useful bits in SLIM for selected cipher splits. Masks contain 1 for bits that are useful.

$r_{\rm in}$	$r_{\rm m}$	$r_{\rm out}$			$k_{\texttt{in}}$	mask	$k_{\texttt{out}}$	mask			$ k_{ m in} $	$ k_{ m out} $
2	8	2			a010	0000	0000	2021			3	3
3	8	3		bfbf	a010	0000	0000	2021	777b		17	15
2	10	2			a8d0	0000	0000	2021			6	3
2	14	2			a8d0	0000	0000	2121			6	4
3	17	3		bfff	a8d0	0000	0000	1658	ffff		21	22
4	17	4	ffff	bfff	a8d0	0000	0000	1658	ffff	ffff	37	38
4	18	4	ffff	efff	160b	0000	0000	2101	bdab	ffff	37	30

Table 6: Useful bits in LBCIoT for selected cipher splits. Masks contain 1 for bits that are useful.

4.2.2 Attack inputs and deterministic bits

Algorithm 1 shows a general pseudocode for differential MITM attack that uses deterministic bits. Let us discuss the inputs first. The cipher is split according to the given rounds numbers $r_{\rm in}$, $r_{\rm m}$, and $r_{\rm out}$. We additionally expect an $r_{\rm m}$ -round differential $\Delta = (\alpha, \beta)$ with probability 2^{-p} . The number of input rounds $r_{\rm in}$ and the difference α determine the set of useful bits $k_{\rm in}$; similarly, $k_{\rm out}$ is determined by $r_{\rm out}$ and β .

Algorithm	1	Differential	MITM	attack	with	deterministic bits
-----------	---	--------------	------	--------	------	--------------------

Inputs: $r_{\text{in}}, r_{\text{m}}, r_{\text{out}}$ – number of rounds how cipher is split into $E_{\text{in}}, E_{\text{m}}$, and E_{out} , respectively $\Delta = (\alpha, \beta) - r_{\text{m}}$ -round differential with probability 2^{-p} $k_{\text{in}}, k_{\text{out}}$ – set of key bits for given Δ , r_{in} , and r_{out} , respectively δ – difference of deterministic bits for given r_{out} and β κ – positive integer Output: set of candidates – key bits for k_{in} and k_{out} 1: Candidates = \emptyset \triangleright Initialize mutliset for candidate keys

2: **repeat** $\kappa \cdot 2^p$ times $H = \emptyset$ \triangleright Initialize the hash table 3: $P \in_R \{0,1\}^n$ ▷ Choose a random plaintext 4: C = E(P) \triangleright Query the oracle 5:for each guess i for $k_{\rm in}$ do 6: Compute \widetilde{P} from P and i, such that $E_{in}(P) \oplus E_{in}(\widetilde{P}) = \alpha$. 7: $\hat{C} = E(P)$ \triangleright Query the oracle 8: Let \mathscr{D} and $\widehat{\mathscr{D}}$ be the deterministic bits of C and \widehat{C} respectively. 9: if $\mathscr{D} \oplus \widehat{\mathscr{D}} = \delta$ then \triangleright Filter on deterministic bits 10: $H[C] \leftarrow i$ \triangleright Store in the hash table 11: \triangleright Guess for k_{out} only if some suitable i exists 12: if $H \neq \emptyset$ then for each guess j for k_{out} do 13:Compute \widetilde{C} from C and j, such that $E_{out}^{-1}(C) \oplus E_{out}^{-1}(\widetilde{C}) = \beta$ 14:for each $i \in H[C]$ do \triangleright (*i*, *j*) is consistent with differential Δ 15:Candidates.add((i, j)) \triangleright Add (i, j) to the multiset 16:17: **return** the most frequent (i, j) values from Candidates

Next input is δ , a difference of deterministic bits for given r_{out} and β . We use a simple algorithm to find the deterministic bits. It takes a few random ciphertext blocks. For each ciphertext C and all possible values of key bits in k_{out} , it computes the corresponding ciphertext C', such that C and C' have difference β before E_{out} . A deterministic bit must have a constant difference of its values for each pair (C, C'). A set of all constant differences forms the resulting δ . The complexity of this algorithm is exponential in $|k_{\text{out}}|$. Table 7 shows positions of deterministic bits (in ciphertext) and the difference δ in SLIM and LBCIoT for selected r_{out} and β .

Remark. When a bit is not deterministic for E_{out} and β , the algorithm usually recognizes this situation fast. A non-constant bit difference in any corresponding (C, C') is a definitive proof.

The last input parameter of Algorithm 1 is κ . We repeat the main cycle for $\kappa \cdot 2^p$ random plaintexts, in order to expect the differential Δ to occur κ times for the correct values of key bits in $k_{\rm in}$ and $k_{\rm out}$. Although the attack requires the differential to occur only once, multiple occurrences, for $\kappa > 1$, help the correct key bits to appear more frequently in the *Candidates* set. This is important if only the most frequent candidates are used for further attack. For practical evidence, see the experiments in Section 5.

	eta	$r_{\rm out}$	deterministic bits \mathscr{D}	$ \mathscr{D} $
SLIM	1000 a008	1	10100000000100000001.00.0.0	25
	1000 a008	2	00001.00.0.0	9
	8000 1d48	1	000111010100100011	18
	9000 02d0	1	00000010110100000100000.0.	24
	0020 08b4	2	00.10	5
	a000 b208	2	.001.00	5
LBCIoT	2000 0a40	2	00000.000000000	14
	0080 0000	2	000000100.00000100000.	21
	0020 1000	2	010000000.0001.00100	19
	0020 1800	2	0100000100.00000100	18

Table 7: Deterministic bits for SLIM and LBCIoT. \mathscr{D} contains '' if the bit is not deterministic, otherwise it shows the difference for the deterministic bit.

4.2.3 Attack description

The algorithm consists of two main steps. First, for each guess i for k_{in} bits, we compute \tilde{P} from P, such that α occurs after E_{in} assuming our guess is correct. This can be done by computing in constant time $\tilde{P} = E_{in}^{-1}(E_{in}(P) \oplus \alpha)$. The knowledge of k_{in} bits is sufficient, since k_{in} contains the useful bits, and all other key bits can be set arbitrarily for this calculation. Afterwards, we use the deterministic bits as a $|\mathcal{D}|$ -bit filter to detect some incorrect guesses, and thus lower the number of candidate entries for i in H.

The second step is executed only when H is not empty, i.e., when at least one i passed the deterministic bits check in the first step. Our experiments in Section 5 confirm that for SLIM and LBCIoT this check significantly reduces the overall number of second steps performed.

We search for the "correct" guess j for k_{out} bits in the second step. A correct guess means we can find a pair (i, j) that is consistent with the differential Δ . We compute \tilde{C} from C, such that β occurs before E_{out} : $\tilde{C} = E_{out}(E_{out}^{-1}(C) \oplus \beta)$, where the knowledge of useful k_{out} bits is sufficient for this calculation. A consistent pair (i, j) is recognized when $\tilde{C} = \hat{C}$. Therefore, we perform the lookup $H[\tilde{C}]$ to collect matching i values.

Remark. A consistent pair (i, j) for k_{in} and k_{out} does not imply that by encrypting P and P the differential Δ occurred. We just know, that using these key fragments we get difference α after E_{in} , starting from the plaintexts, and difference β before E_{out} , starting from the ciphertexts. Certainly, the correct values are among these consistent pairs.

4.2.4 Complexity of the attack

We discuss time, space and data complexity of the proposed attack. For each of the $\kappa \cdot 2^p$ iterations, the expected number of candidates added is the number of all possible candidate pairs $|H| \cdot 2^{|k_{\text{out}}|}$ reduced by the application of the *n*-bit filter, where \hat{C} and \tilde{C} must match on all *n* bits. Moreover, the expected size of *H* is $2^{|k_{\text{in}}|}$ reduced by the application of the $|\mathscr{D}|$ bit filter.

To calculate the effect of these filters, we assume the uniformity of the filters, which is a

standard assumption used in similar analyses [2, 4, 8]. More precisely, we assume a uniform distribution of ciphertexts \hat{C} and \tilde{C} for incorrect (i, j) pairs. As a consequence, a (\hat{C}, \tilde{C}) pair passes the *n*-bit filter with probability 2^{-n} . Similarly, a random \hat{C} passes the $|\mathscr{D}|$ bit filter with probability $2^{-|\mathscr{D}|}$. Hence, the expected size of *H* is $2^{|k_{in}|-|\mathscr{D}|}$, and the expected number of candidates added in one iteration is $2^{|k_{in}|-|\mathscr{D}|+|k_{out}|-n}$.

Remark. The uniformity assumption for the filters lowers the complexity of the attacks, it works "in our favor". The real ciphers can behave differently. We show the experimental evaluation of the attacks for SLIM and LBCIoT in Section 5.

Time complexity. The complexity depends on the iterations in two for-loops, i.e., $|k_{in}|$, $|k_{out}|$, and on the number of candidates added, since we need to build each one candidate from (i, j). Therefore, the time complexity is¹

$$\mathcal{T} = \kappa \cdot 2^p \cdot (2^{|k_{\rm in}|} + 2^{|k_{\rm out}|} + 2^{|k_{\rm in}| - |\mathscr{D}| + |k_{\rm out}| - n}).$$

Space complexity. The complexity depends both on |H|, and on the overall number of candidates stored in the multiset *Candidates*. We do not take into account that many candidates added are already present in the multiset, and require no additional space. The space complexity is therefore bounded from above:

$$\mathcal{S} < 2^{|k_{\rm in}| - |\mathscr{D}|} + \kappa \cdot 2^p \cdot 2^{|k_{\rm in}| - |\mathscr{D}| + |k_{\rm out}| - n}$$

Data complexity. We query the oracle for the entire codebook and cache the results – increasing the space complexity by 2^n – in case when the default attack would query the oracle more times than 2^n . Otherwise, we can trivially calculate the data complexity. Hence, the data complexity is

$$\mathcal{D} = \min\{\kappa \cdot 2^{p+|k_{\mathrm{in}}|}, 2^n\}.$$

Remark. Deterministic bits allow us to detect candidates for which the differential cannot occur, and reject them in advance. Hence, we can use the complexity estimate with deterministic bits even for an attack variant that does not use them. Moreover, we expect that using deterministic bits does not affect the number of candidates. It is because each ciphertext \hat{C} , which has the difference with C on the deterministic bits different from δ , cannot match any ciphertext \tilde{C} computed in the second for loop, since we know that all \tilde{C} are such that their difference with C on the deterministic bits δ .

4.3 Comparison with previous attacks

We denote by $DM(r_{in}, r_m, r_{out})$ the differential MITM attack on $(r_{in} + r_m + r_{out})$ -round cipher split into r_{in} -round E_{in} , r_m -round E_m , and r_{out} -round E_{out} . In our case, the attack is applied on SLIM or LBCIoT, and the split carries information on various component of the attack:

- Differential $\Delta = (\alpha, \beta)$ for $r_{\rm m}$ rounds according to Tables 1 and 2.
- Useful bits k_{in} and k_{out} for given Δ and cipher split, based on Tables 5 and 6.
- Deterministic bits according to Table 7.

¹We do not consider any potential performance gains from the $H \neq \emptyset$ test in this estimate.

We use complexity estimates from Section 4.2.4 for differential MITM attacks. Table 8 shows the best attacks of each type on SLIM and LBCIoT, including classical differential attacks [5], and the best known linear attack on SLIM [10]. The differential MITM attack on LBCIoT is the best attack according to our knowledge.

						bits r	ecovered
cipher	rounds	attack	${\mathcal T}$	S	\mathcal{D}	$ k_{ m in} $	$ k_{ m out} $
SLIM	13^{*}	differential [5]	2^{31}	2^{12}	2^{31}	-	12
	14^{*}	differential [5]	2^{32}	2^{12}	2^{32}	-	12
	16	$\mathrm{DM}(3,11,2)$	$\kappa\cdot 2^{71}$	$\kappa\cdot 2^{65}$	2^{32}	45	26
	18^{*}	$\mathrm{DM}(3,13,2)$	$\kappa\cdot 2^{73}$	$\kappa\cdot 2^{67}$	2^{32}	42	26
	19	linear $[10]$	$2^{64.4}$	2^{38}	2^{32}	-	36
LBCIoT	18	differential [5]	2^{31}	2^{8}	2^{31}	does r	not work ^{\dagger}
	19	differential [5]	2^{32}	2^{3}	2^{31}	does r	not work †
	25	DM(4, 17, 4)	$\kappa\cdot 2^{71}$	$\kappa\cdot 2^{71}$	2^{32}	37	38
	26^{*}	DM(4, 18, 4)	$\kappa\cdot 2^{67}$	$\kappa\cdot 2^{65}$	2^{32}	37	30

^(*) the attack fails for a substantial portion of keys, see Section 4.1.1

 $^{(\dagger)}$ see explanation in the text

Table 8: Selected attacks on reduced versions of SLIM of LBCIoT

Classical differential attack works by decrypting the final round on both \hat{C} and \hat{C} and asking whether the difference is β . In case of LBCIoT, the final round key is simply XOR-ed with the left half, and thus the difference is the same for all keys. Hence, no candidate will be more frequent then any other, i.e., we extract 0 bits of information. In other words, there are 0 useful bits for LBCIoT if $r_{\text{out}} = 1$.

Remark. The problem can certainly be fixed by using shorter differential and decrypting two final rounds. We do not analyze this variant, since the proposed differential MITM attacks work for more cipher rounds.

4.4 Extending the attack to full key recovery

The key schedule of both SLIM and LBCIoT has the following structure, the 80-bit master key is divided into the first five 16-bit round keys, and the remaining round keys are computed by a deterministic algorithm. Therefore, in our attacks, bits in $k_{\rm in}$ correspond to specific bits in the master key.

A simple approach to full key recovery is to guess the remaining $80 - |k_{\rm in}|$ bits of the master key, run the key scheduling, and filter out incorrect keys using known $k_{\rm out}$ bits. We can also consider skipping the filtering part and use only the master keys with prescribed $k_{\rm in}$. Resulting keys are then verified by checking correctness of encryption on a few known plaintext-ciphertext pairs. The success of this approach depends on the number of candidates m and the number of unique candidates for the $k_{\rm in}$ part (let us denote this number by $m_{\rm in}$; trivially $m_{\rm in} \leq m$ and $m_{\rm in} \leq 2^{|k_{\rm in}|}$). We call an attack successful, if it is faster than brute-force attack trying all 2^{80} keys.

- Only $k_{\rm in}$ is used, no filtering: The attack is successful if $m_{\rm in} < 2^{|k_{\rm in}|}$, since we have $m_{\rm in} \cdot 2^{80-|k_{\rm in}|} < 2^{80}$ candidates for the master key.
- Both $k_{\rm in}$ and $k_{\rm out}$ are used: The expected number of keys that requires final verification is $m \cdot 2^{80-|k_{\rm in}|-|k_{\rm out}|}$. We can run the key scheduling for $m_{\rm in} \cdot 2^{80-|k_{\rm in}|}$ master keys and filter out incorrect keys using corresponding $k_{\rm out}$ values. Again, the attack is successful if $m_{\rm in} < 2^{|k_{\rm in}|}$. Note, that if there is a significant complexity imbalance between key scheduling and encryption, it still might be useful to perform the attack, even if $m_{\rm in}$ is close or equal to $2^{|k_{\rm in}|}$.

Remark. Assuming a cipher with independent round keys, the attack is trivially successful. There is nothing to filter on, since both k_{in} and k_{out} bits correspond to specific bits in the master key.

An alternative approach to full key recovery is to iteratively attack a reduced round version of the cipher. The choice of the attacks depends on many details, such that the probability of the differentials or $|k_{in}| + |k_{out}|$ value, i.e., the number of already known bits of the key. Rather than searching through the space of all combinations of the attack's variants to find the optimal mix, we recommend to find multiple differentials for a fixed number of rounds to be able to attack as many bits of each round key as possible, preferably all.

5 Experiments

We implemented several variants of the differential MITM attack to experimentally validate their performance and various assumptions they make. This section presents the most interesting observations. All variants were run 100 times for random keys and the presented numbers are averages of obtained results. Exceptions are mentioned explicitly when relevant.

The attacks are computationally demanding, therefore our experiments cover variants that are computationally feasible, i.e., with reduced number of rounds.

5.1 Candidates

The number of candidates for k_{in} and k_{out} dictates the complexity of subsequent extension of the attack to full key recovery. Ideally, we get only few candidates. The *Candidates* multiset can contain multiple occurrences of a candidate. An interesting question is whether we can focus on the most frequent candidates and ignore the other candidates.

Table 9 shows a simple statistic for candidates, most frequent candidates, unique candidates, total number of candidates, and theoretical expectation of the total number of candidates based on the estimate from Section 4.2.4. We can make the following observations:

- Theoretical estimates that use the standard and commonly used assumptions of filter uniformity, are completely off for small attacks. This affects overall complexity estimates for these attacks as well. It remains an open problem whether the estimates hold for more rounds.
- Collecting and proceeding with only unique candidates, compared to trying each one as it is discovered in the algorithm (including duplicates), reduces the complexity moderately (up to factor 2). However, the importance of not repeating candidates increases with increasing κ , since κ is positively correlated with the total number of candidates.

- A much more promising heuristic is to focus only on the most frequent candidates, since the number of most frequent candidates is significantly lower. Again, increasing κ should help even more.
- Using longer differentials (increasing r_m) is more efficient than increasing k_{in} or k_{out} when attacking SLIM or LBCIoT.

cipher	attack	most frequent ^(*)	unique	total	expected
SLIM	$\mathrm{DM}(1,6,1)$	2.1	75.5	127.1	$\kappa \cdot 2^{-19}$
	$\mathrm{DM}(2,4,2)$	10.6	10021490.7	10067302.4	$\kappa\cdot 2^7$
	$\mathrm{DM}(1,8,1)$	4.7	103.2	157.2	$\kappa\cdot 2^{-21}$
LBCIoT	$\mathrm{DM}(2,8,2)$	1.1	5.6	21.2	$\kappa \cdot 2^{-35}$
	$\mathrm{DM}(2,10,2)$	1.1	15.2	42.4	$\kappa \cdot 2^{-28}$
	DM(2, 14, 2)	2.2	26.2	70.4	$\kappa \cdot 2^{-18}$

(*) when the occurrence of the correct key is at least 2

Table 9: Number of candidates	(experiments with $\kappa =$	7))
-------------------------------	------------------------------	----	---

Increasing the parameter κ

The size of the set of the most frequent candidates seems to be negatively correlated with their multiplicity. It supports the theory, that the greater the occurrence of the correct key, the "more difficult" it is for a random candidate to occur that many times.

The value of κ controls how many times we expect our differential in the attack. In our experiments, with $\kappa = 7$, we expected 7 occurrences of the differential. However, we got from 1 to 17 occurrences. Increasing κ to 14 changed this range to the minimum value 4 and the maximum value 25.

We observed few attacks where the set of most frequent candidates was the entire set of candidates, i.e., all had occurrence 1. In such case, using most frequent candidates instead of all candidates is irrelevant. Hence, we count the average over those attacks in which the most frequent candidates have multiplicity at least 2. Note that this occurred in 98.8% of attacks with $\kappa = 7$, and always for $\kappa = 14$.

5.2 Deterministic bits

The main motivation for deterministic bits is to skip guessing k_{out} in when unnecessary. This happens when $H = \emptyset$, see line 12 of Algorithm 1. Table 10 shows experimental results which count the situations when the attack did not skip the second for-loop: the first column shows an average of 100 runs, the third column is the theoretical estimate for this quantity, which we compute as $\kappa \cdot 2^p \cdot 2^{\min(0,|k_{\text{in}}|-|\mathscr{D}|)}$, i.e., we reduce the number of iterations by the expected size of H, again, assuming the uniformity of \mathscr{D} -bit filter. The second column "out of" displays, for comparison, how many times was this condition tested, i.e., $\kappa \cdot 2^{-p}$. Let us discuss the main observations:

cipher	attack	recorded	"out of"	expected
SLIM	$\mathrm{DM}(1,6,1)$	11.8	28672	7.0
	$\mathrm{DM}(2,4,2)$	445.9	448	448.0
	$\mathrm{DM}(1,8,1)$	39.8	1835008	112.0
LBCIoT	$\mathrm{DM}(2,8,2)$	7.1	7168	0.1
	$\mathrm{DM}(2,10,2)$	12.1	114688	14.0
	$\mathrm{DM}(2,14,2)$	2693.2	29360128	7168.0

Table 10: $H \neq \emptyset$ (experiments with $\kappa = 7$)

- The hash table H is non-empty very rarely. Hence, when using deterministic bits, we often skip over the second for-loop.
- Deterministic bits seem to have no actual impact on the number of candidates, unique candidates, and even the most frequent candidates. Table 11 summarizes statistics where no deterministic bits are used and allows a direct comparison with Table 9.
- The previous observation confirms our expectation that the notion of deterministic bits allows us to better approximate the number of candidates regardless of whether they are implemented or not (since their sole purpose is to prematurely detect bad pairs).

cipher	attack	most frequent ^(*)	unique	total	expected
SLIM	$\mathrm{DM}(1,6,1)$	2.2	72.5	119.2	3.5
	$\mathrm{DM}(2,4,2)$	16.0	9047210.1	9102387.2	458752.0
	$\mathrm{DM}(1,8,1)$	4.2	104.6	166.4	56.0
LBCIoT	$\mathrm{DM}(2,8,2)$	1.1	5.6	21.7	$\kappa \cdot 2^{-16}$
	$\mathrm{DM}(2,10,2)$	1.2	14.4	39.8	$\kappa \cdot 2^{-9}$
	$\mathrm{DM}(2,14,2)$	2.1	27.8	72.1	$\kappa \cdot 2^0$

(*) when the occurrence of the correct key is at least 2

Table 11: Number of candidates without deterministic bits (experiments with $\kappa = 7$)

5.3 Identical bits

We noticed the following phenomenon: few bits were identical among all of the most frequent candidates. Based on this observation, we propose the following modification: The attack outputs just one candidate – a vector of the identical bits, and a mask informing where the identical bits are located. Table 12 presents the average number of identical bits in various attacks, and compares it with overall bits guessed in candidates, i.e., $|k_{\rm in}| + |k_{\rm out}|$.

If multiple differentials are known, we can repeat the attack for each differential, hopefully covering the majority of round key bits for the first r_{in} rounds and the last r_{out} rounds. The remaining few bits can be brute-forced.

We present the following observations:

- By using the identical bits, we often lose little information, but we may lose a lot when there are only few identical bits. Therefore, we propose the following test. If the number of most frequent candidates is much smaller than the number of candidates satisfying the identical bits $2^{|k_{in}|+|k_{out}|-l}$ (2 to the power of not identical bits), then the attack should output the exact set of candidates.
- We observe that the number of identical bits is negatively correlated with increasing number of most frequent candidates.
- By increasing κ , we often increase the number of identical bits, up to a limit. For attack DM(2, 4, 2) on SLIM, by changing κ from 7 to 14, the average number of identical bits changed from 37.8 to 39.7. However, increasing κ further did not increase the number beyond 40, even though $|k_{\rm in}| + |k_{\rm out}|$ is 42.
- Use of deterministic bits does not significantly change the number identical bits. In our experiments, the average changed by ± 0.2 . Table 12 uses algorithm with deterministic bits.

cipher	attack	identical bits	$ k_{\rm in} + k_{\rm out} $
SLIM	$\mathrm{DM}(1,6,1)$	15.9	19
	$\mathrm{DM}(2,4,2)$	37.8	42
	$\mathrm{DM}(1,8,1)$	11.7	17
LBCIoT	$\mathrm{DM}(2,8,2)$	5.9	6
	$\mathrm{DM}(2,10,2)$	8.8	9
	$\mathrm{DM}(2,14,2)$	7.9	10

Table 12: Number of identical bits (experiments with $\kappa = 7$)

6 Conclusion

We have applied the differential MITM attack, with improvements, on lightweight ciphers SLIM and LBCIoT. Firstly, we proposed that the attack returns only the most frequent candidates. We have shown, experimentally for small cases, that it can significantly lower the resulting number of candidates. Secondly, based on the results of our experiments, we proposed the notion of identical bits, i.e., instead of returning a set of candidates, we return one partial candidate consisting of such bits (and their values), which have the same value across all most frequent candidates. We believe that these modifications may prove useful in attacks on other ciphers. Our attacks on 25 and 26-round LBCIoT are the best partial key recovery attacks on LBCIoT reported to date.

We have also discussed the problem with low-probability differentials. There might be keys for which no plaintext pair can produce that differential. We have experimentally verified this for ciphers SLIM and LBCIoT. Indeed, for a differential with probability close to 2^{-n} , there is a significant percentage of keys such that the differential does not occur for any plaintext pair. Whether this occurs for other ciphers is left as an open problem.

Our experiments have shown a problem with the complexity implications of *l*-bit filters. It is often assumed that a random candidate passes through an *l*-bit filter with probability 2^{-l} . We have shown, that for small cases², these approximations are incorrect. Whether this problem is universal for attacks on other ciphers, and when exactly the assumption holds is left for further investigation.

Acknowledgment

The computational resources were provided by the project ITMS2014+:313021X329 – Advancing University Capacity and Competence in Research, Development and Innovation (ACCORD), co-financed by the European Regional Development Fund.

References

- Bassam Aboushosha et al. "SLIM: A Lightweight Block Cipher for Internet of Health Things". In: *IEEE Access* 8 (2020), pp. 203747–203757. DOI: https://doi.org/10. 1109/ACCESS.2020.3036589.
- Zahra Ahmadian et al. Improved Differential Meet-In-The-Middle Cryptanalysis. Cryptology ePrint Archive, Paper 2024/351. 2024. URL: https://eprint.iacr.org/2024/351.
- [3] Vijesh Bhagat et al. "Lightweight cryptographic algorithms based on different model architectures: A systematic review and futuristic applications". In: Concurrency and Computation: Practice and Experience 35.1 (2023), e7425. DOI: https://doi.org/10. 1002/cpe.7425.
- [4] Christina Boura et al. Differential Meet-In-The-Middle Cryptanalysis. Cryptology ePrint Archive, Paper 2022/1640. 2022. URL: https://eprint.iacr.org/2022/1640.
- Yen Yee Chan et al. "On the resistance of new lightweight block ciphers against differential cryptanalysis". In: *Heliyon* 9.4 (2023). ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2023.e15257.
- [6] Sumit Singh Dhanda, Brahmjit Singh, and Poonam Jindal. "Lightweight Cryptography: A Solution to Secure IoT". In: Wireless Personal Communications 112.3 (2020), pp. 1947–1980. DOI: https://doi.org/10.1007/s11277-020-07134-3.
- [7] Rabie A. Ramadan et al. "LBC-IoT: Lightweight Block Cipher for IoT Constraint Devices". In: Computers, Materials & Continua 67.3 (2021), pp. 3563-3579. ISSN: 1546-2226. DOI: https://doi.org/10.32604/cmc.2021.015519.

²those that we were able to verify experimentally

- [8] Ling Song, Qianqian Yang, and Huimin Liu. Revisiting the Differential Meet-In-The-Middle Cryptanalysis. Cryptology ePrint Archive, Paper 2023/1302. 2023. URL: https: //eprint.iacr.org/2023/1302.
- [9] Stefan Kölbl. CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives. URL: https://github.com/kste/cryptosmt.
- [10] Nobuyuki Sugio. "Bit-Based Evaluation of Lightweight Block Ciphers SLIM, LBC-IoT, and SLA by Mixed Integer Linear Programming". In: *IET Information Security* 2024.1 (2024), p. 1741613. DOI: https://doi.org/10.1049/2024/1741613.
- [11] Vishal A. Thakor, Mohammad Abdur Razzaque, and Muhammad R. A. Khandaker. "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities". In: *IEEE Access* 9 (2021), pp. 28177– 28193. DOI: 10.1109/ACCESS.2021.3052867.
- [12] Meltem Sonmez Turan et al. Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology. 2023. DOI: https://doi.org/10. 6028/NIST.IR.8454.

A Differentials for SLIM and LBCIoT

We present a comprehensive set of differentials found in [5]. We have verified all listed differentials experimentally.

				r	α	eta	2^{-p}
				3	0010 0000	2000 0a40	2^{-4}
				4	0020 0000	0000 0800	2^{-4}
				5	0002 0000	4000 0002	2^{-6}
				6	0002 0400	0000 2000	2^{-8}
1	\cdot α	β	2^{-p}	7	0002 0000	0000 0800	2^{-8}
ć	3 8d10 0400	0000 0d00	2^{-4}	8	0002 0000	0020 1000	2^{-10}
4	4 1000 b000	1000 a008	2^{-6}	9	2000 0040	0020 1000	2^{-12}
Ę	5 d804 0040	0040 d804	2^{-8}	10	0006 0400	0020 1000	2^{-14}
6	6 0208 4700	8000 1d48	2^{-12}	11	0006 0400	0120 3800	2^{-16}
7	7 09a6 001a	001a 4982	2^{-16}	12	2000 0040	0120 3000	2^{-18}
8	3 9024 0090	9000 02d0	2^{-18}	13	0006 0400	0000 0800	2^{-20}
Q) 0b82 000a	000a 0a82	2^{-21}	14	0006 0400	0020 1800	2^{-22}
1() 0020 00ъ0	0080 4823	2^{-24}	15	0006 0400	0000 0800	2^{-24}
11	4827 0080	0020 08b4	2^{-26}	16	0006 0400	0020 1000	2^{-26}
12	2 0b82 000a	0a00 801b	2^{-28}	17	0006 0400	0100 2040	2^{-28}
13	3 a208 a000	a000 b208	2^{-31}	18	6000 0040	0000 0800	2^{-30}
	Differentials for SLIM Differentials for LBCIoT						