Differential-MITM Attack on 14-round ARADI

Shibam Ghosh, Bastien Michel, María Naya-Plasencia

Inria, Paris, France
{shibam.ghosh, bastien.michel,maria.naya_plasencia}@inria.fr

Abstract. ARADI is a low-latency block cipher introduced by the U.S. National Security Agency (NSA) for secure and efficient memory encryption applications. In contrast to most ciphers proposed in the academic community, the design rationale for ARADI has not been publicly disclosed, limiting external evaluation to independent cryptanalysis. Several such analyses have already been published, with the most effective attacks to date reaching up to 12 out of 16 rounds. In this work, we present a differential meet-in-the-middle attack on ARADI that incorporates several new optimizations and dedicated techniques, enabling, for the first time, an attack extending to 14 rounds of the cipher.

1 Introduction

Securing sensitive computations within CPU-bound secure enclaves has become standard in modern processor architectures. Yet, safeguarding main memory (RAM) has proven challenging due to inherent physical and architectural limitations. This persistent vulnerability has made RAM a favored vector for adversaries conducting physical attacks. For example, the *Cold Boot* Attack [HSH+08] exploits the phenomenon of data remanence in RAM, enabling attackers to retrieve cryptographic keys and other confidential data long after the intended computation has finished. To address these risks, cryptographic memory protection mechanisms, and most notably memory encryption have been advocated. However, for such solutions to be viable within high-performance systems, the underlying cryptographic primitives must operate with negligible added latency.

This demand has sparked focused research into block ciphers that are meticulously engineered for *low-latency*. Ciphers such as PRINCE [BCG+12] and its evolution, PRINCEv2 [BEK+20], epitomize this trend toward ultra-efficient encryption. The field has also witnessed the emergence of tweakable block ciphers, which add operational flexibility and further expand the applicability of memory encryption. Noteworthy examples in this category include MANTIS [BJK+16], QARMA [Ava17], and QARMAv2 [ABD+23]. The drive for minimal latency has even extended beyond RAM encryption: ciphers like CHILOW [BDG+25] have been developed for low-overhead code encryption in embedded systems, while SCARF [CGL+23] enhances cache security through address randomization, mitigating side-channel threats. Aimed this landscape, the low-latency block cipher ARADI was introduced by the U.S. National Security Agency (NSA) [GMW24]. Released in tandem with its dedicated mode of operation, LLAMA, ARADI aims to

deliver both confidentiality and integrity protections for memory within modern computing systems. Its 128-bit block size and 256-bit key directly address the pressing demand for cryptographic mechanisms.

However, the ARADI design document provides little insight into the underlying design rationale and offers only a minimal, high-level security evaluation. This lack of detailed justification and limited scope of initial analysis have led to concerns about ARADI's resilience against a broad spectrum of cryptanalytic attacks. As a result, there remains a clear need for independent and comprehensive security assessments to substantiate the cipher's security properties.

1.1 Related Cryptanalysis on ARADI

The cryptographic community has subjected ARADI to rigorous security examination through various approaches. Bellini et al.[BFG+24] initiated this effort with the automatic tool CLAASP [BGG+23], exploring diffusion properties, and both differential and linear cryptanalysis up to 9 rounds, alongside impossible differentials (8 rounds) and zero-sum distinguishers (7 rounds). In subsequent work [BRRT24], they extended the zero-sum distinguisher to 8 rounds, and developed a 10-round (out of 16 rounds) key-recovery attack. They also observed a curious byte-wise equality pattern in 5-round cube sums, which was later theoretically explained by Kim et al.[KKL+24] using division property. Dunkelman et al. [DG25] further pushed the boundaries by extending the 8-round zero-sum distinguisher to propose a 12-round key-recovery attack.

Alternative cryptanalytic approaches have emerged in parallel. While Hu et al.[HJ25] proposed a related-key attack using impossible boomerang distinguishers, an impossible differential attack in the single-key setting was presented in [DE25]. Beyond classical cryptanalysis, Kwon et al. [KH24] examined ARADI's side-channel vulnerabilities, while Avanzi et al. [ADG24] identified a potential weakness in LLAMA (ARADI's mode of operation) related to variable-length initialization vectors — an issue subsequently addressed by the ARADI designers [GMW24]. This diverse cryptanalytic landscape continues to refine our understanding of ARADI's security boundaries.

1.2 Contributions

The extensive cryptanalysis approaches on ARADI as discussed above show that ARADI is an important and challenging target for cryptographic attacks. In this work, we investigate ARADI using a relatively recent cryptanalytic approach: the family of differential meet-in-the-middle (Differential-MITM) attacks. Originally introduced in [BDD+23], differential-MITM attacks are particularly powerful against ciphers where the key size exceeds the state size, a property shared by ARADI and other designs recently studied [DZW25,AKM+24,CSNS25]. Within this setting, in the attack on ARADI, Differential-MITM techniques can outperform classical differential attacks, in particular thanks to the use of initial structures.

It looks like a promising family of attacks and several optimizations have been proposed in the literature. While applying differential-MITM attack to ARADI, we

have developed new improvements and generalized strategies, which we detail in Section 3 and that are of independent interest. We expect these contributions to be broadly useful in future research on these attacks.

Our attack uses the differential distinguisher from [BFG+24]. We have refined its probability analysis and tailored it to our specific scenario. The methodology and findings are provided in Section4.1.

Bringing all these elements together, we have succeeded in mounting, for the first time, a differential meet-in-the-middle attack on 14 rounds of ARADI, as described in Section 4. A thorough comparison of our results with previous work is presented in Table 1[1].

Table 1. Comparison of results with prior work on the key-recovery attacks on ARADI.

Rounds	Type	Time	Data	Memory	Reference
10	Integral	2^{192}	2^{124}	2^{131}	[BRRT24]
10	Integral	2^{125}	2^{125}	2^{76}	[DG25]
10	Integral	$2^{117.6}$	$2^{117.6}$	2^{80}	[DG25]
11	Integral	2^{126}	2^{126}	2^{42}	[DG25]
11	Integral	$2^{153.3}$	$2^{118.6}$	$2^{131.5}$	[DG25]
12	Integral	$2^{156.5}$	$2^{126.6}$	$2^{135.5}$	[DG25]
13	Impossible *	2^{255}	$2^{121.47}$	2^{112}	[DE25]
14	Diff-MITM	$2^{247.57}$	2^{128}	$2^{178.58}$	Section 4
14	Diff-MITM	$2^{248.57}$	2^{126}	$2^{178.58}$	Section 4.8

2 Background

2.1 Specification of ARADI

This section presents a formal description of the ARADI cipher. ARADI is a Substitution-Permutation Network (SPN) that processes 128-bit data blocks under a 256-bit master key. The cipher's internal state can be naturally structured as a 4×32 binary matrix; more precisely, for a state x, the j-th bit of the i-th row is denoted by $x_i[j]$ ($0 \leq i \leq 3$ and $0 \leq j \leq 31$). Thus, the j-th column of the state $x[j]$ consists of $(x_0[j],,x_1[j],,x_2[j],,x_3[j])$, and the i-th row, x_i, consists

[1] We contacted the authors of [DE25] for clarification, as their result was outlined only briefly in a methodological paper. They confirmed that, under the provided parameters, their method achieves only a one-bit advantage.

of the bits $(x_i[0], , \ldots, , x_i[31])$, as shown in the following

$x_0[31]$	$x_0[30]$	$x_0[29]$	$x_0[28]$	$x_0[27]$	$x_0[26]$	$x_0[25]$	$x_0[24]$	$x_0[23]$	$x_0[22]$	$x_0[21]$	$x_0[20]$	$x_0[19]$	$x_0[18]$	$x_0[17]$	$x_0[16]$	$x_0[15]$	$x_0[14]$	$x_0[13]$	$x_0[12]$	$x_0[11]$	$x_0[10]$	$x_0[9]$	$x_0[8]$	$x_0[7]$	$x_0[6]$	$x_0[5]$	$x_0[4]$	$x_0[3]$	$x_0[2]$	$x_0[1]$	$x_0[0]$
$x_1[31]$	$x_1[30]$	$x_1[29]$	$x_1[28]$	$x_1[27]$	$x_1[26]$	$x_1[25]$	$x_1[24]$	$x_1[23]$	$x_1[22]$	$x_1[21]$	$x_1[20]$	$x_1[19]$	$x_1[18]$	$x_1[17]$	$x_1[16]$	$x_1[15]$	$x_1[14]$	$x_1[13]$	$x_1[12]$	$x_1[11]$	$x_1[10]$	$x_1[9]$	$x_1[8]$	$x_1[7]$	$x_1[6]$	$x_1[5]$	$x_1[4]$	$x_1[3]$	$x_1[2]$	$x_1[1]$	$x_1[0]$
$x_2[31]$	$x_2[30]$	$x_2[29]$	$x_2[28]$	$x_2[27]$	$x_2[26]$	$x_2[25]$	$x_2[24]$	$x_2[23]$	$x_2[22]$	$x_2[21]$	$x_2[20]$	$x_2[19]$	$x_2[18]$	$x_2[17]$	$x_2[16]$	$x_2[15]$	$x_2[14]$	$x_2[13]$	$x_2[12]$	$x_2[11]$	$x_2[10]$	$x_2[9]$	$x_2[8]$	$x_2[7]$	$x_2[6]$	$x_2[5]$	$x_2[4]$	$x_2[3]$	$x_2[2]$	$x_2[1]$	$x_2[0]$
$x_3[31]$	$x_3[30]$	$x_3[29]$	$x_3[28]$	$x_3[27]$	$x_3[26]$	$x_3[25]$	$x_3[24]$	$x_3[23]$	$x_3[22]$	$x_3[21]$	$x_3[20]$	$x_3[19]$	$x_3[18]$	$x_3[17]$	$x_3[16]$	$x_3[15]$	$x_3[14]$	$x_3[13]$	$x_3[12]$	$x_3[11]$	$x_3[10]$	$x_3[9]$	$x_3[8]$	$x_3[7]$	$x_3[6]$	$x_3[5]$	$x_3[4]$	$x_3[3]$	$x_3[2]$	$x_3[1]$	$x_3[0]$

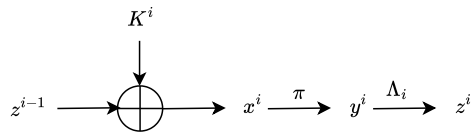
2.1.1 Round Function of ARADI: Each round of ARADI comprises three transformations: a substitution layer (SBox), a linear layer, and a round key addition. Specifically:

- The substitution layer is denoted π.
- The round-dependent linear layer is denoted Λ_i.
- The round key addition is denoted by τ^{k^i}, where k^i is the i-th round key and τ denotes XOR with the key.

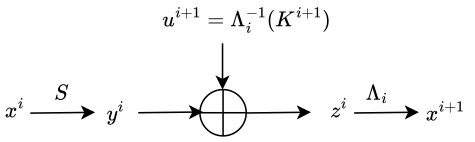
The i-th round can thus be expressed as $\mathcal{R}_i = \tau^{k^i} \circ \Lambda_i \circ \pi$. The cipher consists of 16 such rounds, and a final round key addition is performed after the last round. Consequently, the overall encryption can be written as:

$$\tau^{k^{16}} \circ (\Lambda_{16} \circ \pi) \circ \tau^{k^{15}} \circ \cdots \circ (\Lambda_1 \circ \pi \circ \tau^{k^1}) \circ (\Lambda_0 \circ \pi \circ \tau^{k^0}).$$

Throughout, we denote by x^i and y^i the input and output states of the i-th S-box layer, and by z^i the state subsequent to the linear layer in round i. This is summarized in the following



For analytic convenience, we occasionally employ an equivalent representation of the round function in which the order of the linear layer Λ_i and the subsequent round key addition $\tau_{k^{i+1}}$ is interchanged. In this case, the effective key is adjusted as $u^{i+1} = \Lambda_i^{-1}(k^{i+1})$, as summarized in the following



If we interchange in the final round, the resulting final operation is a linear layer, which may be omitted without loss of generality.

Table 2. The 4-bit SBox S and its inverse S^{-1} of ARADI.

S	0 1 2 3 4 D F 6 8 B 5 E C 7 A 9
S^{-1}	0 1 2 3 4 A 7 D 8 F E 9 C 5 B 6

Substitution Layer (π). The substitution layer applies a fixed 4-bit SBox S independently to each of the 32 columns of the state, in parallel. Explicit lookup tables for S and its inverse S^{-1} are given in Table 2.

Linear Layer (Λ). The linear layer Λ_i updates each row of the internal state by means of a round-dependent linear transformation. Specifically, at round i, the state x is mapped via:

$$\Lambda_i(x) = (L_j(x_0),\, L_j(x_1),\, L_j(x_2),\, L_j(x_3)), \text{ where } j = i \bmod 4.$$

Each L_j is an involutive linear map on $\mathbb{F}_2^{32}$. Writing a row as the concatenation $u\|l$ of two 16-bit halves, the map L_j is:

$$L_j(u\|l) = (u \oplus (u \lll a_j) \oplus (l \lll c_j))\|(l \oplus (u \lll a_j) \oplus (l \lll b_j)),$$

where $\lll$ denotes bitwise left rotation, and the offsets a_j, b_j, c_j are specified in Table 3.

Table 3. The rotation offsets of the linear layer of ARADI.

$j = i \bmod 4$	a_j	b_j	c_j
0	11	8	14
1	10	9	11
2	9	4	14
3	8	9	7

2.1.2 The Key Schedule of ARADI.

Key Schedule. The 256-bit master key K is regarded as eight 32-bit words, $\mathsf{K} = (\mathsf{K}_0, \ldots, \mathsf{K}_7)$. The key schedule state at round i is denoted $\mathsf{K}^i = (\mathsf{K}_0^i, \ldots, \mathsf{K}_7^i)$. Initially, the key state is loaded directly from the master key, i.e.,

$$\mathsf{K}^0 = \mathsf{K} = (\mathsf{K}_0^0, \mathsf{K}_1^0, \ldots, \mathsf{K}_7^0).$$

The 128-bit round keys K^i, for $i = 0, \ldots, 15$ and the final key K^{16}, are derived from the key schedule state by concatenating four 32-bit words as follows:

$$K^i = \begin{cases} \mathsf{K}_0^i||\mathsf{K}_1^i||\mathsf{K}_2^i||\mathsf{K}_3^i, & \text{if } i \bmod 2 = 0 \\ \mathsf{K}_4^i||\mathsf{K}_5^i||\mathsf{K}_6^i||\mathsf{K}_7^i, & \text{if } i \bmod 2 = 1. \end{cases}$$

The key schedule updates K^i using three steps:

Linear Transformations: At first, two invertible linear maps, M_0 and M_1, each operating on pairs of 32-bit words, are applied to the key state. These maps are defined as:

$$M_0(x, y) = ((x \lll 1) \oplus y, (y \lll 3) \oplus (x \lll 1) \oplus y$$
$$M_1(x, y) = ((x \lll 9) \oplus y, (y \lll 28) \oplus (x \lll 9) \oplus y$$

The map M_0 is applied to $\mathsf{K}_0^i||\mathsf{K}_1^i$ and $\mathsf{K}_4^i||\mathsf{K}_5^i$ while M_1 is applied to $\mathsf{K}_2^i||\mathsf{K}_3^i$ and $\mathsf{K}_6^i||\mathsf{K}_7^i$.

Word Permutation: Next, a permutation P is applied to the eight words K^i, defined as

$$\mathsf{P} = \begin{cases} (1,2)(5,6), & \text{if } i \bmod 2 = 0 \\ (0,3)(4,7), & \text{if } i \bmod 2 = 1 \end{cases}$$

Round Constant Addition: Finally, the step index i is `XOR`-ed as a round constant to the word K_7^i.

We note that `ARADI`'s key schedule is an affine transformation. Hence, all the round keys are an affine combination of the master key bits.

2.2 Differential Distinguisher

Differential cryptanalysis, originally introduced by Biham and Shamir [BS90], [BS91], is a foundational technique for evaluating the security of block ciphers. In this approach, an adversary seeks to identify a *differential characteristic*, that is, a sequence describing how an input difference propagates through each round of the cipher. An r-round differential characteristic is represented as $(a^0 \to a^1 \to \cdots \to a^r)$, where a^i denotes the difference after round i. For certain attacks, attention is focused not on the detailed sequence but solely on the probability that a specific input difference δ_{in} transitions to a given output difference δ_{out} after r rounds. This event defines a *differential* $\delta_{in} \to \delta_{out}$ with associated probability $p = \Pr(\delta_{in} \to \delta_{out})$. It is important to note that a single differential $\delta_{in} \to \delta_{out}$ may correspond to many distinct characteristics, each representing a possible trajectory of difference propagation. The probability of the overall differential is given by summing the probabilities of all such characteristics, that is,

$$\Pr(\Delta_{in} \to \Delta_{out}) = \sum_{a_1} \sum_{a_2} \cdots \sum_{a_{r-1}} \Pr[\Delta_{out} = a_r, ..., \Delta_{y_1} = a_1 | \Delta_{in} = a_0]. \quad (1)$$

This collection of characteristics is often referred to as a *cluster* for the differential $\delta_{in} \to \delta_{out}$. We discuss more on the differential characteristic used in our attack on ARADI and its clustering effect at Section 4.1.

2.3 Key-recovery with Differential Meet-in-the-Middle Attacks

Differential Meet-in-the-Middle (differential-MITM) attacks represent a recent and powerful extension of classical meet-in-the-middle strategies, specifically designed to leverage differential distinguishers for key-recovery purposes. This technique was first introduced at CRYPTO 2022 [BDD+23], where the authors demonstrated how to mount a differential-MITM attack utilizing a standard differential distinguisher. Furthermore, they provided initial insights into employing parallel partitioning, a traditional meet-in-the-middle approach, within the context of differential attacks to further enhance their effectiveness. Subsequent advancements presented at EUROCRYPT 2024 [AKM+24] illustrated that differential-MITM attacks can also be instantiated using truncated differential distinguishers. This latter work introduced several refinements to the attack methodology, notably the state-test technique, and a more sophisticated application of parallel partitioning. As a result, the differential-MITM technique has been successfully applied to a range of symmetric ciphers, including prominent designs such as AES, SKINNY, CRAFT, SIMON, SIMECK, and KATAN. In what follows, we first recapitulate the classical differential-MITM framework as originally proposed in [BDD+23]. We then discuss contemporary enhancements to this attack, including the state-test, as well as advanced structural techniques, as detailed in [BDD+23], [AKM+24].

Consider a generic r-round iterative block cipher $\mathcal{E}$ with an n-bit block size and a k-bit master key K. Suppose the adversary possesses a differential $\delta_{in} \to \delta_{out}$ over r_d middle rounds of $\mathcal{E}$, which holds with probability 2^{-p}. The essential idea of a differential-MITM attack, like classical differential key-recovery attack, is to treat the cipher as three concatenated components: r_{in} initial rounds, r_d middle rounds (the differential distinguisher), and r_{out} final rounds, so that $r = r_{in} + r_d + r_{out}$ and $r_{in} + r_{out} > 0$. We denote these as $\mathcal{E} = \mathcal{E}_{out} \circ \mathcal{E}_d \circ \mathcal{E}_{in}$, where $\mathcal{E}_{in}$ encompasses the first r_{in} rounds, $\mathcal{E}_d$ the middle r_d rounds (subject to the differential distinguisher), and $\mathcal{E}_{out}$ the last r_{out} rounds.

The attack proceeds by guessing certain subkey bits relevant to the outer rounds (either the initial or final set), enabling the attacker to partially encrypt or decrypt and thus to test if a given plaintext-ciphertext pairs satisfy the differential at the appropriate interface. In particular, for the first r_{in} rounds, let $\mathcal{K}_{in}$ denote the set of subkey bits necessary to compute the differences between two plaintexts $(P, \widetilde{P})$ in order to verify the presence of the input difference δ_{in} at the entry to the differential distinguisher. Similarly, for the last r_{out} rounds, let $\mathcal{K}_{out}$ denote the subkey bits required to compute the differences between the corresponding ciphertexts $(C, \widetilde{C})$ to verify the output difference δ_{out}. The sizes of these key spaces are denoted $k_{in} = |\mathcal{K}_{in}|$ and $k_{out} = |\mathcal{K}_{out}|$, respectively. This general differential-MITM attack strategy is illustrated in Figure 1.

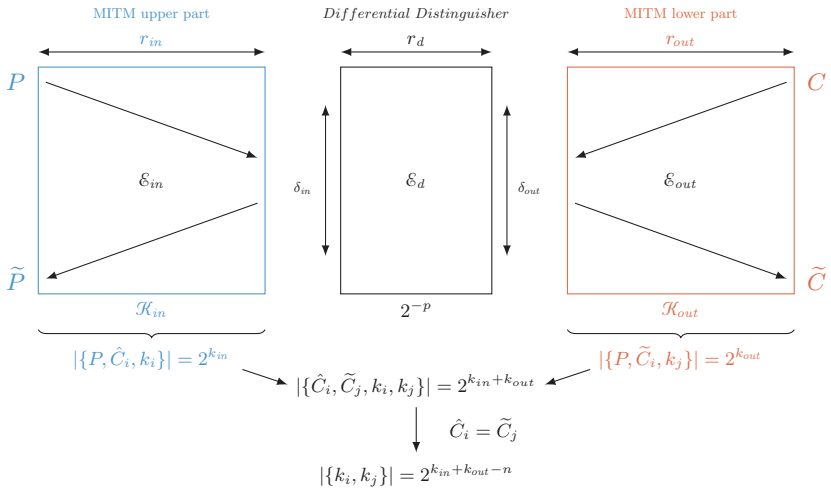


Fig. 1. High level description of the classic differential-MITM technique

Having established the basic framework for key-recovery, we now outline the main steps of the differential-MITM attack. The attack proceeds as follows: the adversary begins by choosing a plaintext P and obtaining its ciphertext $C = \mathcal{E}(P)$ via encryption oracle access. The attack then consists of three steps:

- **Upper Part:** For each possible guess k_i of the subkey bits $\mathcal{K}_{in}$ for the initial rounds, the attacker finds a plaintext $\widetilde{P}_i$ such that $\mathcal{E}_{in}(P) \oplus \mathcal{E}_{in}(\widetilde{P}_i) = \delta_{in}$, where $\mathcal{E}_{in}$ denotes the function corresponding to the first r_{in} rounds given k_i. The encryption of each $\widetilde{P}_i$ is then queried, and the resulting $\hat{C}_i = \mathcal{E}_{in}(\widetilde{P}_i)$, along with the associated key guess k_i, is stored in a hash table $\mathcal{H}$ as $\mathcal{H}(P, \hat{C}_i) = k_i$. In total, there are $2^{k_{in}}$ such candidates $(P, \hat{C}_i, k_i)$.
- **Lower Part:** Similarly, for each possible guess k_j of the subkey bits $\mathcal{K}_{out}$ for the final rounds, the attacker computes a ciphertext $\widetilde{C}_j$ derived from C, such that $\mathcal{E}_{out}^{-1}(C) \oplus \mathcal{E}_{out}^{-1}(\widetilde{C}_j) = \delta_{out}$, where $\mathcal{E}_{out}^{-1}$ denotes the inverse of the last r_{out} rounds under k_j. This yields $2^{k_{out}}$ candidates $(P, \widetilde{C}_j, k_j)$.
- **Matching:** As lower part candidates are generated, the attacker searches for collisions in the hash table $\mathcal{H}$ — that is, determines whether any $\widetilde{C}_j$ matches a stored index $\hat{C}_i$. Each collision yields a candidate key pair (k_i, k_j). These candidates can be expanded (if necessary), and are subsequently tested against additional plaintext-ciphertext pairs until the correct key is identified. The expected number of such key candidates, corresponding to the expected number of collisions between the two lists of size $2^{k_{in}}$ and $2^{k_{out}}$ over an n-bit state, is $2^{k_{in}+k_{out}-n}$.

Because the differential in $\mathcal{E}_d$ holds with probability 2^{-p}, this process must be repeated 2^p times, on average, to obtain a single pair (P, C) satisfying the differential distinguisher.

Complexity Analysis: The overall complexity of the differential-MITM attack is primarily determined by the sizes of the subkey spaces $\mathcal{K}_{in}$ and $\mathcal{K}_{out}$, with cardinalities k_{in} and k_{out}, respectively. To minimize redundant key guesses, it is advantageous to first resolve the bits common to both subkey sets, denoted $k_\cap = |\mathcal{K}_{in} \cap \mathcal{K}_{out}|$. By guessing these overlapping bits only once, we can optimize the memory complexity. The resulting time complexity is given by:

$$\mathcal{T} = 2^{p+k_\cap} \left(2^{k_{in}-k_\cap} + 2^{k_{out}-k_\cap} + 2^{k_{in}+k_{out}-2k_\cap-n}\right), \tag{2}$$

where p denotes the negative logarithm of the probability of the differential and n is the block size.

Depending on implementation choices, either the upper or lower tuples can be stored in memory, and this selection can be made to minimize memory usage. Accordingly, the memory complexity is:

$$\mathcal{M} = 2^{\min(k_{in}-k_\cap, k_{out}-k_\cap)}. \tag{3}$$

The data complexity is $\mathcal{D} = \min\left(2^n,\ 2^{p+\min(k_{in}-k_\cap,\, k_{out}-k_\cap)}\right)$, since, during the matching step, one must check whether $\widetilde{C}_j = \hat{C}_i$ for all relevant pairs.

We now discuss contemporary enhancements to this attack, including the *state-test*, as well as advanced structural techniques, as detailed in [BDD+23], [AKM+24].

State-test Technique. The state-test technique, originally introduced in[DSP07] for Meet-in-the-Middle attacks and subsequently applied to impossible differential attacks in [BNS14], seeks to lower time complexity by guessing a subset of internal *state bits* rather than subkey bits. This technique is recently used in differential-MITM attack in [AKM+24] and later improved in [MNPN25].

This approach reduces the number of key guesses required when evaluating whether candidate pairs follow the differential path. Let s_{in} and s_{out} denote the number of state-test bits employed in the upper and lower parts of the attack, respectively. In this refined framework, each candidate $\widetilde{P}_i$ (upper part) and each candidate $\widetilde{C}_j$ (lower part) must be generated not just for each key and plaintext guess, but also for every possible configuration of s_{in} and s_{out} state-test guesses. Consequently, the total number of candidates in each part increases by a factor of $2^{s_{in}}$ or $2^{s_{out}}$; however, if the internal state depends on more than $s_{in} + k_{in}$ or $s_{out} + k_{out}$ key bits, the overall set of candidates is reduced. The resulting time and memory complexities are given by:

$$\mathcal{T} = 2^p(2^{k_{in}+s_{in}} + 2^{k_{out}+s_{out}} + 2^{k_{in}+s_{in}+k_{out}+s_{out}-k_\cap-n}), \tag{4}$$

$$\mathcal{M} = 2^{min(k_{in}+s_{in}-k_\cap, k_{out}+s_{out}-k_\cap)}, \tag{5}$$

where the data complexity remains unchanged.

While the use of the state-test technique does reduce the number of key bits that are "directly" recovered, it simultaneously introduces additional constraints in the form of (possibly nonlinear) equations over the unknown key bits. Specifically, when a state-test is applied in the upper part of the differential-MITM attack, it yields an equation relating key elements not guessed in the upper phase,

potentially together with certain plaintext bits. Conversely, when the state-test is implemented in the lower part, the resulting equation involves key bits outside those guessed in the lower phase, along with bits from the ciphertext. Crucially, these equations encapsulate valuable information about the master key and can play a dual role in the cryptanalytic process. Not only can they be exploited to recover the full key in a post-processing stage — once sufficient candidate keys have been identified — but they can also be used as filters during the matching step to reduce the number of false positives. Leveraging these additional algebraic relations significantly enhances the power of the attack by providing further means of eliminating incorrect key candidates or directly solving for the correct key. In the context of our work, we make extensive use of the state-test technique in the cryptanalysis of `ARADI`, both to strengthen the attack and to improve its efficiency.

Parallel Partitioning Technique: The parallel partitioning technique, introduced in [BDD+23], represents an advancement in the application of differential-MITM attacks to block ciphers, enabling the inclusion of additional rounds within the attack. This method fundamentally broadens the scope of the attack by constructing a carefully chosen "structure", a subset of internal state values, that allows the attacker to efficiently manage and align the states at an intermediate point in the cipher and thus incorporate more rounds into the analysis. Initially, this framework was particularly suitable for ciphers whose round keys affect only a subset of the internal state. Subsequently, the technique was extended and refined to apply to ciphers without partial key addition, as detailed in [AKM+24]. In the following, we describe the approach adapted from [AKM+24], which is well suited for `ARADI`.

To implement parallel partitioning, the cipher is partitioned into four distinct segments, rather than three as in the classical differential-MITM setup. Specifically, an auxiliary component known as the *structure* ($\mathcal{E}_s$) is introduced. This structure covers r_s rounds of the cipher and is positioned before the upper part or after the lower part. For the sake of clarity and without loss of generality, we assume the structure precedes the upper part, so that the cipher $\mathcal{E}$ is decomposed as $\mathcal{E} = \mathcal{E}_{out} \circ \mathcal{E}_d \circ \mathcal{E}_{in} \circ \mathcal{E}_s$. We refer to the state at the end to the structure as X (see Figure 2).

The attack proceeds by fixing $F \leq n$ bits of the internal state within the structure, and the set of fixed bits are denoted as $\mathcal{S}^f$. The index f identifies a particular assignment to these F fixed bits. The bits are chosen such that, for any guess of the key components $\mathcal{K}_{in}$ relevant to the upper part, the adversary can compute corresponding F bits of the post-structure state X, denoted by X^f. Similarly, the selection ensures that, given any guess for the lower subkey $\mathcal{K}_{out}$, one can efficiently compute the F bits of the plaintext P, denoted P^f. For each f, the adversary enumerates all 2^s possible choices for the unfixed bits, where $s = n - F$ is the size of the structure, yielding 2^s complete values for X^f, and P^f. Then computes the ciphertexts $C^f = \mathcal{E}(P^f)$. The enhanced attack then unfolds as follows (cf. Figure 2):

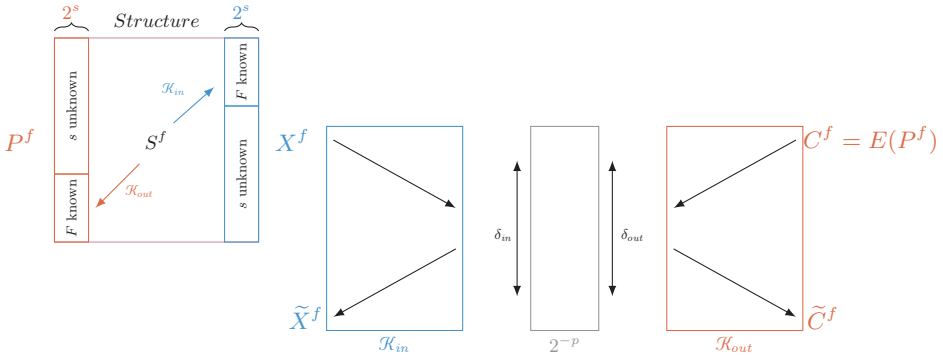


Fig. 2. High-level schematic of the parallel partitioning in differential-MITM attacks.

- **Upper Part:** For each X^f and each possible guess k_i of the subkey bits $\mathcal{K}_{in}$ pertaining to the upper rounds, the attacker determines $\widetilde{X}_i^f$ such that $\mathcal{E}_{in}(X^f) \oplus \mathcal{E}_{in}(\widetilde{X}_i^f) = \delta_{in}$. The resulting $2^{s+k_{in}}$ candidates $(X^f, \widetilde{X}_i^f, k_i)$ are stored in a table $\mathcal{H}$.
- **Lower Part:** Likewise, for each $C^f = \mathcal{E}(P^f)$ and each candidate key k_j for the lower rounds, the attacker computes a ciphertext $\widetilde{C}_j^f$ such that $\mathcal{E}_{out}^{-1}(C^f) \oplus \mathcal{E}_{out}^{-1}(\widetilde{C}_j^f) = \delta_{out}$. The attacker then decrypts $\widetilde{C}_j^f$ to obtain the corresponding plaintexts $\widetilde{P}_j^f$. This process produces $2^{s+k_{out}}$ tuples $(P^f, \widetilde{P}_j^f, k_j)$.
- **Matching:** The matching phase proceeds as in the classical methodology, but with an added refinement: for each pair $(X^f, \widetilde{X}_i^f, k_i)$ and $(P^f, \widetilde{P}_j^f, k_j)$, the adversary checks whether $\widetilde{X}_i^f$ and $\widetilde{P}_j^f$ are consistent with the F fixed bits of the structure. Each consistency yields a new candidate key pair (k_i, k_j). This process sieves 2^F triplet pairs, resulting in $2^{s+k_{in}+k_{out}-n}$ key candidates.

Due to the probability 2^{-p} that the differential holds in the middle rounds, this overall process must be repeated 2^{p-s} times on average. It is worth emphasizing that the matching (or filtering) step is now substantially more nuanced, as additional relationships (possibly unfixed) within the structure can be leveraged to further eliminate spurious key candidates. In our analysis of `ARADI`, we have demonstrated the practical benefit of such supplementary filters, denoted by a filtering factor $2^{\mathcal{A}}$ for the number of additional pairs excluded by new structural constraints. The time and memory complexity of this parallel partitioning differential-MITM attack are given by:

$$\mathcal{T} = 2^{p-s}(2^{s+k_{in}} + 2^{s+k_{out}} + 2^{k_{in}+k_{out}-k_\cap-F-\mathcal{A}}) \tag{6}$$

$$\mathcal{M} = 2^{min(s+k_{in}-k_\cap, s+k_{out}-k_\cap)} \tag{7}$$

Putting Everything Together: Bringing together all the previously discussed ideas, we now consider the comprehensive differential-MITM attack. This full

framework integrates the classical differential-MITM methodology with the state-test and parallel partitioning enhancements. An explicit description of the combined attack procedure is presented in Algorithm 1. The resulting time, memory and data complexities, are summarized as follows:

$$\mathcal{T} = 2^{p-s}\left(2^{s+k_{in}+s_{in}} + 2^{s+k_{out}+s_{out}} + 2^{k_{in}+s_{in}+k_{out}+s_{out}-k_{\cap}-F-\mathcal{A}}\right) \quad (8)$$

$$\mathcal{M} = 2^{min(s+k_{in}+s_{in}-k_{\cap}, s+k_{out}+s_{out}-k_{\cap})} \quad (9)$$

$$\mathcal{D} = \min\left(2^{n},\ 2^{p+\min(k_{in}-k_{\cap},\, k_{out}-k_{\cap})}\right) \quad (10)$$

Furthermore, there is a proved lower bound of the time complexity of this attack, as derived in [BDD+23], is :

$$\mathcal{T} \geq 2^{p+k-n} \quad (11)$$

Algorithm 1 Differential Meet-in-the-Middle

Output: Right key
Fix F bits of the set $\mathcal{S}^f$ inside the structure
for each guess k in $\mathcal{K}_{in} \cap \mathcal{K}_{out}$ **do**
    $\mathcal{H} \leftarrow \emptyset$ ▷ *table for the upper pairs*
    for each guess i in $\{\mathcal{K}_{in}/(\mathcal{K}_{in} \cap \mathcal{K}_{out}), s_{in}\}$ **do**
        Generate F fixed bits of X^f from $\mathcal{S}^f$ and (k, i)
        for each possible value Y of the $s = (n - F)$ remaining bits of X_f **do**
            Compute $\widetilde{X}^f$ from X^f, i and k. ▷ *partial encryption for pair generation*
            $\mathcal{H} = \mathcal{H} \cup \{X_f, \tilde{X}_f, i\}$
    for each guess j in $\mathcal{K}_{out}/(\mathcal{K}_{in} \cap \mathcal{K}_{out})$ **do**
        Generate F fixed bits of P^f from $\mathcal{S}^f$ and (j, k)
        for each possible value Z of the $s = (n - F)$ remaining bytes of P^f **do**
            $C^f \leftarrow \mathcal{E}(P^f)$ ▷ *Encryption oracle call*
            Compute $\widetilde{C}^f$ from C^f and (j, k). ▷ *partial decryption for pair generation*
            Compute $\widetilde{P}^f \leftarrow E^{-1}(\widetilde{C}^f)$ ▷ *Decryption oracle call*
            for each element $\{X^f, \widetilde{X}^f, i\}$ of $\mathcal{H}$ **do**
                if $\{X^f, \widetilde{X}^f, i\}$ and $\{P^f, \widetilde{P}^f, j\}$ are consistent through the structure **then** ▷ *Expect $F + \mathcal{A}$ filtering effect*
                    Complete (i, j, k) to retrieve the master key
                    Try the candidates against extra data ▷ *Full encryption*

3 New Generic Ideas for Optimising Differential-MITM Attacks

In the next section, we present a dedicated differential-MITM attack on ARADI that leverages two notable advancements: the state-test technique and an initial

1-round structure using parallel partitioning. Additionally, our approach exploits the linearity of the key schedule, enabling an efficient implementation of the matching step, but in order to make it work and to optimize it, we introduce several novel techniques, which serve as general improvements for differential-MITM attacks. We detail these generic strategies here as key methodological insights for future cryptanalyses; this exposition will also serve to streamline the subsequent description of our proposed attack.

3.1 Between Truncated and Classical Distinguishers for Differential-MITM

In [AKM+24], the generalization of differential-MITM attacks to accommodate truncated distinguishers was introduced. This naturally raises the question of whether a hybrid approach could be considered, one in which a subset of possible differences in the active nibbles at the input and/or output of the distinguisher is allowed. Exploring such hybrid strategies may, in certain scenarios, offer advantages in terms of attack complexity.

As a concrete example, consider the case of a classical single-path distinguisher, like it is the case for `ARADI`. Here, we could relax the conditions on the first or last round. Instead of requiring a unique input difference for the distinguisher ($|\delta_{in}| = 1$), with an associated probability $2^{-p_{up}}$ for that round, we could instead allow a subset of possible input differences of size $|\delta_{in}| = 2^{d_{in}} > 1$, resulting in a new probability $2^{-p'_{up}}$ for that transition. In general, $p'_{up} > p_{up}$, when choosing a single path, we consider the transition with the best probability, and typically $d_{in} \approx p'_{up}$. An analogous relaxation can be performed at the last round, with corresponding parameters d_{out}, p_{low}, and p'_{low}. Analyzing the impact of these modifications on the overall time complexity yields the following insights:

$$\mathcal{T} = 2^{p-\mathbf{p_{up}}-\mathbf{p_{low}}+\mathbf{p'_{up}}-\mathbf{d_{in}}} 2^{-s} (\mathbf{2^{d_{in}}} 2^{k_{in}+s_{in}+s} + \mathbf{2^{d_{out}}} 2^{k_{out}+s_{out}+s} + \mathbf{2^{d_{in}+d_{out}}} 2^{\bigcup_{k_{in},k_{out},s_{in},s_{out}} 2^{2s-n}}). \quad (12)$$

Since we have $d_{in} \approx p'_{up}$, the first term in the complexity expression will be multiplied by a factor of $2^{-p_{up}-p_{low}+d_{in}}$, which, in principle, reduces this term compared to an attack using a unique input difference δ_{in}. Similarly, the second term is adjusted by a factor of $2^{-p_{up}-p_{low}+d_{out}}$, and is likewise diminished. In contrast, the third term is scaled by $2-p_{up} - p_{low} + d_{in} + d_{out} \approx 2-p_{up} - p_{low} + p'_{up} + p'_{low}$; and since both $p'_{up} > p_{up}$ and $p'_{low} > p_{low}$, this factor exceeds one, meaning the third term actually increases. We can therefore conclude that applying this technique can only beneficial when the dominant complexity comes from either the first or second term, corresponding to the upper or lower part of the attack.

This analysis is based on the single-path setting. If clustering effects are considered, as is the case in the attack on `ARADI`, the probability computations must be refined carefully. While we then use corrected probabilities in place of

the parameters p_{up}, p_{low}, p'_{up}, and p'_{low}, the underlying effect on the complexity remains analogous to the reasoning presented for the single-path scenario, as will be discussed in the following section.

3.2 On Leveraging Distinguisher Rounds for Enhanced Filtering and Key Recovery

A well-known lower bound on the complexity of differential-MITM attacks is given by 2^{p+k-n}, where k is the number of key bits involved. To further improve the attack's complexity, we propose explicitly incorporating certain rounds of the distinguisher into the key-recovery phase. By verifying these rounds during the attack, the effective lower bound on complexity can be reduced to $2^{p'+k-n}$ with $p' < p$, since the probability for the verified rounds no longer contributes to the overall attack complexity. This approach offers a second important benefit: as candidate keys are proposed during the attack, the values (or relations) assigned to the key bits and state variables involved in the additional distinguisher rounds can also be deduced. Notably, due to dependencies introduced by the key schedule, this may enable even further filtering of key candidates. In our attack on `ARADI`, we implement this strategy for the first round of the distinguisher. However, we believe this technique has broader applicability and should be considered as a general method for strengthening differential-MITM attacks.

It is worth noting that, if clustering effects are present, i.e., if probability computations for the distinguisher include multiple characteristics within these rounds, then applying this technique can be more nuanced and potentially less effective. To address this in our analysis, we compute the distinguisher probability with the first round excluded from the clustering calculation: we fix the differences in the first and second rounds, and only then consider clustering. This adjustment, which does not significantly alter the overall probability, is explained in Section 4.1 and greatly simplifies the practical application of our proposed technique.

3.3 Recovering the Full Key with State-test Equations

For ciphers with a linear key schedule, the difficulty recovering the full key using state-test equations can vary depending on the amount of direct key information available. If the number of directly guessed key bits exceeds the size of the key, it is straightforward to filter out incompatible candidates. However, if the direct guesses cover fewer bits than the key size, it becomes less clear how to efficiently exploit the redundant information embedded in the state-test equations. In the initial differential-MITM attack on CRAFT [AKM+24], an ad hoc approach was proposed: after matching the upper and lower parts, additional filtering based on the remaining state-test equations was required. Later work [MNPN25] introduced a triangulation method for handling this step.

Building on the first idea, we have developed a method that leverages the amount of directly available key information during the attack to efficiently reformulate the remaining state-test equations. Let rs denote the number of these remaining bit-equations, and let r be the number of newly defined remaining unknown variables after performing Gaussian elimination. Using this reduction, each of the rk final unknown key bits can be expressed as a Boolean function of the r remaining unknowns and of the already determined key bits. The structure of this resulting system may sometimes allow for a direct and simple solution; however, even in the less favorable cases (as encountered with `ARADI`), there exists a general strategy for solving it optimally, that is, with a computational cost determined by the actual number of solutions, using precomputation. Specifically, one can precompute all possible instances of the remaining bit-equations as functions of the unknown variables. The required computational resources for this approach are determined by:

- the number of possible equations for the rk remaining key bits, as each previously guessed keybits will determine a different system
- the various possibilities for the bit variables already determined through guessed key bits and known data, denoted as nv (as in the CRAFT attack),
- all possible assignments for the r remaining unknown variables,
- the probability of satisfying the remaining rs bit-equations (i.e., $2^{-\mathsf{rs}}$).

Here, nv and rk are dictated by the specific structure of the state-test equations for the cipher in question, while the other quantities are defined by the attack parameters. In summary, this approach allows for efficient and comprehensive key recovery using state-test equations, either through direct reduction or optimized precomputation, depending on the algebraic structure of the remaining unknowns.

The complexity and the size of the precomputation becomes:

$$\mathcal{T}_{precomp} = 2^{\mathsf{nv}+\mathsf{rk}+\mathsf{r}-\mathsf{rs}}.$$

Once the matching phase yields candidate triplets, we can efficiently determine the remaining key bits by consulting the precomputed table: for each triplet, we use the $\mathsf{nv} + \mathsf{rk}$ bits derivable from the information obtained in the upper and lower parts of the attack to immediately retrieve the corresponding candidate values for the remaining key bits. This process ensures that the overall computational complexity is optimal and dictated solely by the actual number of possible solutions. To clarify this concept, we refer the reader to the detailed application in the final stage of the `ARADI` attack, discussed in the following section.

We believe that, whenever the key schedule is linear, yet more intricate than a simple bit permutation (or is absent, as in the case of CRAFT), a similar precomputation-based approach will be both effective and interesting, provided that the size of the precomputed table remains manageable compared to the overall complexity of the attack. In the particular case of `ARADI`, for instance, the overhead of such precomputation is negligible.

4 Attack on 14-round ARADI

Having established the general framework for the differential-MITM attack in Section 2.3 and several generic improvements in Section 3, we now turn to its application in the concrete setting of a 14-round key-recovery attack on ARADI. For this purpose, we employ the 9-round differential distinguisher introduced in[BRRT24], which is examined in detail in Section 4.1. We discuss the main attack in Section 4.2

4.1 Differential Distinguisher of ARADI

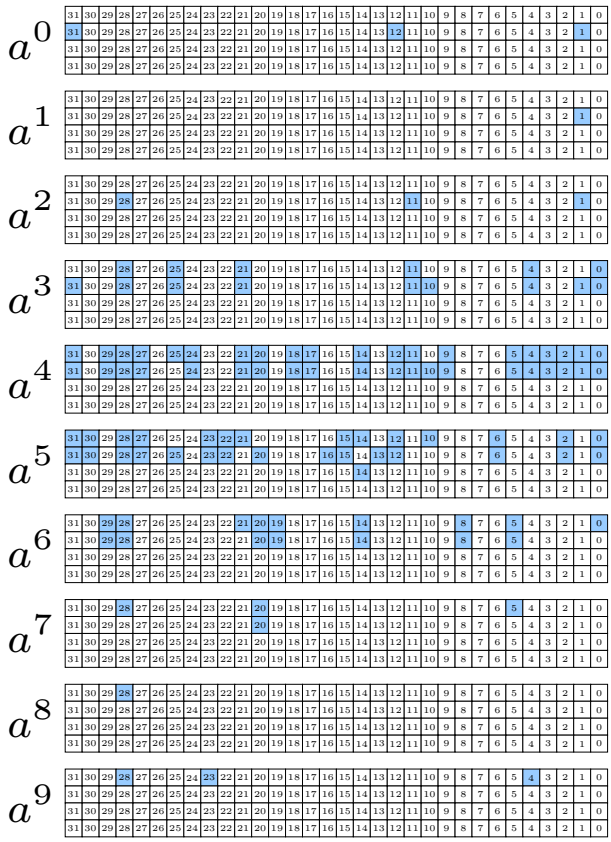


Fig. 3. The differential distinguisher for 9 rounds of ARADI. A colored square represents an active bit. Each state a^i is the state before round i.

In our key-recovery analysis, we leverage the 9-round differential characteristic of ARADI presented in [BRRT24], whose distinguisher (shown in Figure 3)

was initially estimated to occur with probability 2^{-137}. The authors of [BRRT24] further analyzed the clustering effect and ultimately reported a probability of $2^{-125.5}$ for the corresponding differential.

In this work, we revisit the computation of the differential probability, explicitly addressing the impact of clustering. Given an initial differential characteristic, one can estimate the differential's total probability by enumerating and aggregating the probabilities of all distinct characteristics within the cluster defined by the input and output differences. To facilitate this, we employ an automated SAT-based approach, which is adoption of cryptosmt tool [Ste] for ARADI.

By encoding the problem of finding valid differential characteristics as a boolean satisfiability instance, we are able to efficiently enumerate all characteristics leading from δ_{in} to δ_{out}. For each probability value 2^{-p}, all characteristics with this probability (as well as those with lesser probabilities such as $2^{-(p+1)}, \ldots$) are counted up to a practical computational limit. Summing the respective probabilities of all identified characteristics yields an estimate for the total differential probability. Applying this SAT-based methodology to the distinguisher, we observe a total probability of $2^{-124.65}$, slightly higher than the $2^{-125.5}$ reported in [BRRT24]. We attribute this discrepancy mainly to the threshold employed for small characteristic probabilities in our enumeration, as in our analysis, we included characteristics with probabilities down to 2^{-148}, while we get $2^{-125.5}$ when we count characteristic with probability up to 2^{-142}.

Input Difference	**Output Difference**	**Probability**
0x40000000000000000004000000000040	0x00008000080000000000000000080000	$2^{-124.65}$
0x40000000000000000004000000000040	0x00005000050000000000000000050000	$2^{-125.08}$
0x40000000000000000004000000000040	0x00007000070000000000000000070000	$2^{-125.99}$
0x40000000000000000004000000000040	0x0000A0000A000000000000000000A0000	$2^{-124.65}$
0x40000000000000000004000000000040	0x0000D0000D000000000000000000D0000	$2^{-125.99}$
0x40000000000000000004000000000040	0x0000F0000F000000000000000000F0000	$2^{-125.08}$

Table 4. The 9-round $(\mathcal{R}_{13} \circ \cdots \circ \mathcal{R}_4)$ differential distinguishers of ARADI.

There is, moreover, an additional motivation to revisit the clustering effect of this distinguisher. In our key-recovery attack, we consider several output differences for the same input difference, encompassing not only the original output difference from [BRRT24] but also additional candidates. Indeed, as described in Section 3.1, if instead of considering a classical differential-MITM distinguisher, we consider a hybrid between simple and truncated, where several output differences are possible, we are able to reduce the cost of the first terms of the time

complexity, which allows in some cases (as in our attack on ARADI) to balance out the terms and optimize the overall complexity. For this, instead of considering just one possible value for δ_{out}, we have considered several possible. In order to correctly take this into account when computing the empirical probability, we have launched our SAT-based tool for taking into account the clustering effect for each of this possible output differences.

The clustering effects associated with each output difference are summarized in Table 4. By taking into account all output differences and their corresponding clustering effects, we ultimately obtain an overall estimated probability of $2^{-122.55}$ for the differential distinguisher.

Furthermore, we have evaluated the probability of the distinguisher when the clustering does not include the first round, specifically, for an 8-round distinguisher with the input difference 0x00000000000000000000000000000040. The clustering effects corresponding to each output difference are summarized in Table 5, yielding an overall estimated probability of $2^{-116.99}$.

Input Difference	**Output Difference**	**Probability**
0x00000000000000000000000000000040	0x00008000080000000000000000080000	$2^{-119.01}$
0x00000000000000000000000000000040	0x00005000050000000000000000050000	$2^{-119.60}$
0x00000000000000000000000000000040	0x00007000070000000000000000070000	$2^{-120.48}$
0x00000000000000000000000000000040	0x0000A0000A0000000000000000A0000	$2^{-119.01}$
0x00000000000000000000000000000040	0x0000D0000D0000000000000000D0000	$2^{-120.48}$
0x00000000000000000000000000000040	0x0000F0000F0000000000000000F0000	$2^{-119.60}$

Table 5. The 8-round ($\mathscr{R}_{13} \circ \cdots \circ \mathscr{R}_5$) differential distinguishers of ARADI.

Validity of the Differential Distinguisher: We have examined the validity of the 9-round differential characteristics used in our work. Our main motivation stems from the *Hypothesis of Stochastic Equivalence* [LMM91], which asserts that the expected probability of a characteristic averaged over all keys closely matches the probability for almost all individual key choices. However, existing literature provides substantial evidence that this assumption does not always hold: the success of differential characteristics can indeed be highly key-dependent. As a result, verifying key dependence is crucial.

Two generic directions address this problem. Beyne and Rijmen [BR22] introduced the theory of quasidifferential trails, which track probabilistic linear equations governing the differences along a characteristic in the fixed-key setting. The overall probability of a differential characteristic can then be calculated as the sum over all compatible quasidifferential trails. Another approach is the tool AutoDiVer, introduced by Nageler et al. in [NGJE25], which estimates the space

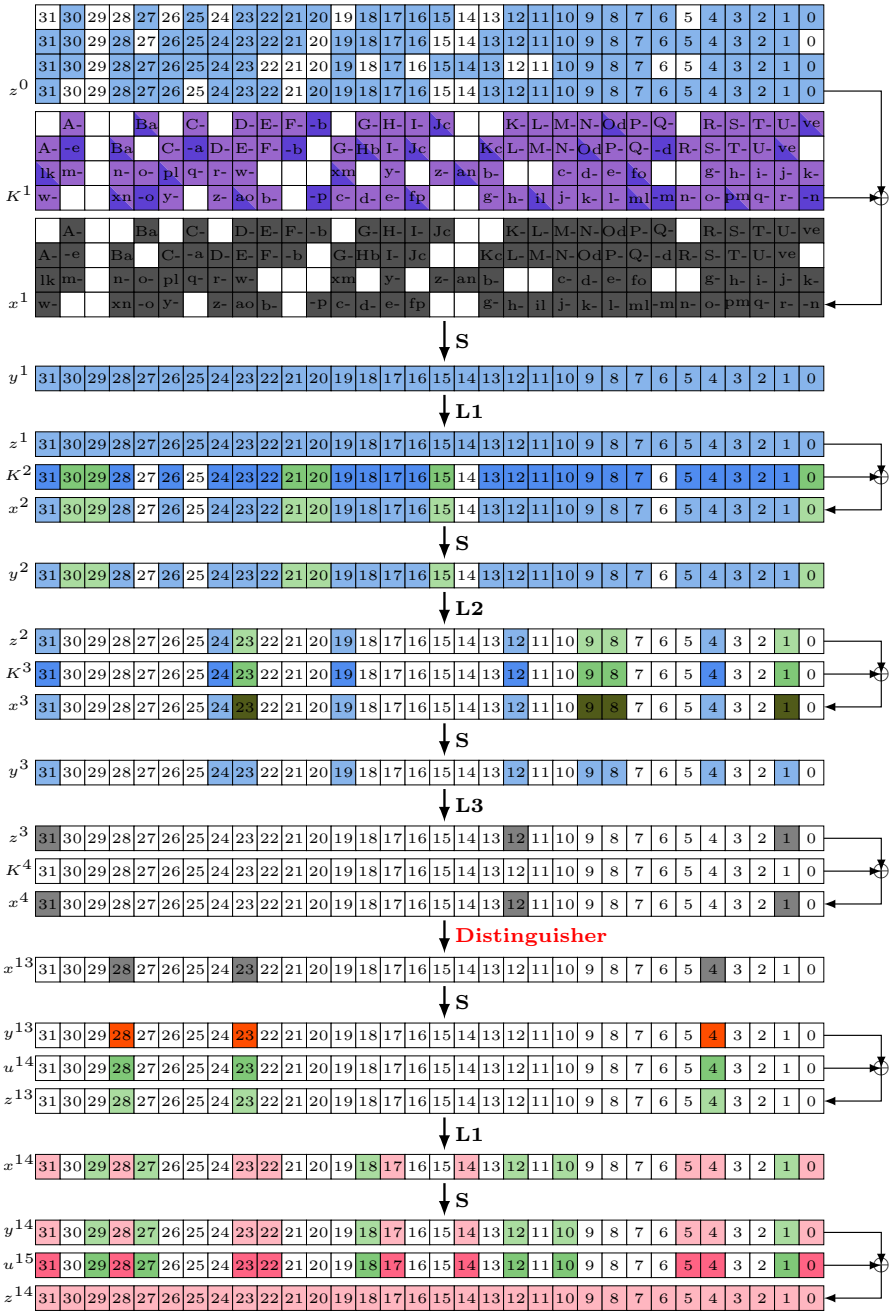


Fig. 4. Overview of the 14-round attack on ARADI. The range of the distinguisher, spanning from round $\mathcal{R}^4$ to $\mathcal{R}^{13}$, is indicated as (■). Key nibbles that must be guessed in the upper and lower parts are denoted by (■) and (■), respectively, while corresponding state values required in these parts are shown as (■) and (■). Nibbles subjected to state-testing in the upper and lower parts are designated as (■) and (■), respectively. Additionally, key nibbles and state nibbles that no longer need to be guessed due to the state-test technique are highlighted as (■) and (■), respectively. Importantly, the state nibbles highlighted as (■) are not *active* through difference propagation, so these values are not required.

of keys supporting valid characteristic pairs and deduces the relevant key conditions using SAT modeling, approximate model counting, and clause learning. Other works have developed cipher-specific analyses, such as the study of *plateau characteristics* for AES [DR07], key dependence in GIFT and SKINNY [PT22], and weak-key spaces for QARMAv1 [ADG25].

In our study, we initially explored the quasidifferential trail approach, but analyzing the 9-round characteristic proved complex. Therefore, we implemented `ARADI` in the AutoDiVer framework, which allowed us to identify 29 linear conditions on the round keys under which the characteristic is valid, as detailed in Appendix A. We also offer an analytical explanation for the emergence of these linear key-spaces in Appendix A, grounded in the specific design of `ARADI` linear layer. We believe these insights highlight interesting avenues for further investigation. In the worst case scenario, this would mean that our characteristic is valid for a well defined set of weak-keys, and the weak-key equations could simplify the key recovery part of the attack. Nevertheless, as we are considering clustering, we expect the class of keys for which the attack works to be much bigger, and we have preferred not to take this into account. The main lesson learned from our study is that the charactiristic is valid for a large class of keys, as we wanted to check.

4.2 Differential-MITM Attack

We now explain how to extend the previous distinguisher by 3 rounds in the beginning and 2 rounds in the end using differential-MITM approach. In our attack, the distinguisher is positioned to cover rounds 4 through 13; that is, we set $\mathcal{E}_d = \mathcal{R}^{13} \circ \cdots \circ \mathcal{R}^4$. To extend the attack, we prepend two rounds prior to the distinguisher ($r_{in} = 2$), and append two rounds after the distinguisher ($r_{out} = 2$). Additionally, we incorporate a 1-round structure before the upper part $\mathcal{E}_{in}$. In total, this configuration allows us to attack 14 consecutive rounds of `ARADI` taking away the first and the last rounds of the full construction (which makes the distinguisher fall in the right place regarding the ordering of the linear transformations), spanning from round 1 to round 15, such that

$$\mathcal{E} = \mathcal{E}_{out} \circ \mathcal{E}_d \circ \mathcal{E}_{in} \circ \mathcal{E}_s = (\mathcal{R}^{15} \circ \mathcal{R}^{14}) \circ (\mathcal{R}^{13} \circ \cdots \circ \mathcal{R}_4) \circ (\mathcal{R}_3 \circ \mathcal{R}_2) \circ \mathcal{R}_1.$$

It should be noted that, due to the design of `ARADI`'s linear layer — which repeats a different transformation every four rounds — the distinguisher can only commence at a round whose index is a multiple of four. We have also investigated distinguishers starting from other rounds; however, such alternatives offer either lower probabilities or do not yield improved key-recovery performance compared to our chosen configuration.

In our attack, besides the use of initial structures, we also take into account the state-test technique, which allows us to considerably reduce the complexity. We considered probabilistic extensions; however, in this particular case, they were not helpful for reducing complexity due to the specific shape of the extension and the properties of the linear layer. In addition, we apply our new ideas introduced in Section 3, which yield significant gains in the attack's efficiency.

We initiated our attack using fixed input and output differences within the distinguisher. As the computational bottleneck emerged from one of the two terms associated with either the upper or lower part of the attack, we refined our approach by allowing $2^{d_{out}} = 6$ output differences, i.e., $d_{out} \neq 1$. As discussed in Section 3, this adjustment can optimize the attack complexity by balancing the workload between the two phases. This consideration is consistent with our earlier probability calculations involving clustering for the distinguisher, and it directly informs the probability values used in our overall complexity analysis. Accordingly, the parameters for our attack are set as follows: $p = 116.99 + 6 = 122.99$, $d_{out} = 2.58$, $F = 54$, $s = (n - F) = 74$, $k_{in} = 88 + 20 = 108$, $k_{out} = 36 + 44 + 10 = 90$, $s_{in} = 16$, and $s_{out} = 12$. With these parameters, we now explain the various parts of the attack.

4.3 Initial structure:

We begin by detailing the application of the parallel partitioning technique, in which we fix specific `XOR` combinations of state variables rather than the state variables themselves. The primary motivation for this approach is to enhance filtering efficiency by leveraging properties of the key schedule (the detail will be further elaborated later in the text). Specifically, we construct a 1-round structure that covers the first round of the attack, in which we fix $F = 54$ selected `XOR` values involving various bits of x^1. By doing so, the attacker can enumerate $2^s = 2^{128-54} = 2^{74}$ possible assignments for the remaining state bits in x^1 when building the structure for the upper part of the attack. If, in the lower part, we guess the same set of 54 `XOR` values, but now for bits of K^1, we can compute equivalent `XOR` relations for the corresponding plaintext bits. This construction also allows us to generate a structure for the lower part of size 2^{74}, seamlessly aligning with the upper part.

There is a good reason for choosing exactly 54 `XOR` values to fix, as well as for the specific selection of these relations. This choice is motivated by our aim to maximize the number of key bits in the upper part that can be filtered using these fixed relations, by exploiting the linear structure of the key schedule. More concretely, among the 88 key bits from K^2 that must be guessed in the upper part (as discussed in detail in the subsequent subsection), we can, owing to linear relations imposed by the key schedule, filter these guesses based on 44 specific relations that connect these key bits to 44 of the 54 `XOR` values guessed in K^1. The same rationale applies for the 20 guessed key bits from K^3, which, via an additional 10 computable relations, further enhance the filtering process. The explicit equations connecting the relevant subkeys, where $K^i_j[l]$ denotes the bit at column l of row j in subkey i, are given as follows:

$$K^2_1[i] = K^1_0[i-1] \oplus K^1_1[i] \tag{13}$$

$$K^2_3[i] = K^1_2[i-9] \oplus K^1_3[i] \tag{14}$$

$$K^3_0[i] \oplus K^3_2[i] = K^1_0[i-6] \oplus K^1_1[i-3] \oplus K^1_0[i-4] \tag{15}$$

$$K^3_1[i] \oplus K^3_3[i] = K^1_2[i-24] \oplus K^1_3[i+4] \oplus K^1_2[i-5] \tag{16}$$

All of these relations are illustrated in Figure 4: each group of bits in K^1 sharing the same letter corresponds to a distinct guessed XOR relation. For instance, the value of $K_0^1[30] \oplus K_1^1[31]$ is guessed because both bits are marked with the letter 'A'. Letters positioned to the left indicate relations associated with K^2, whereas those on the right correspond to relations with K^3. As an example, $K_0^1[27] \oplus K_1^1[28]$ signifies a relation in K^2, while $K_0^1[27] \oplus K_1^1[28] \oplus K_1^1[25]$ signifies a relation in K^3.

Filtering the Pairs Via the Structure: After generating the triplets $(x^1, \widetilde{x}^1, i)$ for each of the $2^{k_{in}+s_{in}}$ upper-part key and state-test guesses i, and $(P, \widetilde{P}, j)$ for the $2^{k_{out}+s_{out}}$ lower-part guesses j, the attacker proceeds to filter candidate pairs $(\widetilde{P}, \widetilde{x}^1)$ by verifying their consistency with respect to the fixed bits imposed by the structure. This initial step eliminates a fraction 2^{-54} of the pairs, corresponding to the number of fixed bits. A further layer of filtering is achieved by ensuring that the differences between P and $\widetilde{P}$, as well as those between x^1 and $\widetilde{x}^1$, agree on the remaining (unfixed) bits; this step is expected to filter an additional 2^{-74} fraction of the pairs. Thus, in total, the filtering across both the fixed and unfixed portions yields the expected 2^{-128} reduction, consistent with the state size. Importantly, successfully matching these differences not only validates the consistency of the pair, but also allows full recovery of the remaining unknown parts of K^1. Consequently, for each matched triplet, the value of K^1 is completely determined, and, as shown previously, this also reveals half of the key bits in K^2 and K^3. These deduced relationships, stemming from the key schedule, enable even more filtering and further reduce the candidate space.

4.4 Key-Recovery Extension and State-test Technique

Once we have built the initial structure, we focus our differential-MITM attack on the subsequent two rounds preceding the distinguisher ($r_{in} = 2$) and the two rounds following it ($r_{out} = 2$). Given that the structure comprises 2^{74} possible values, the entire procedure must be performed $2^{p-74} = 2^{112.99-74} = 2^{48.99}$ times to achieve a reasonable probability of success. For each repetition, we generate the upper and lower sets of triplets, match them according to the structure and key-schedule constraints, and finally filter the matching pairs using all state-test conditions. This process yields a reduced candidate set for the master key. The following outlines each step of the attack in detail:

- **Upper Part:** To satisfy the distinguisher's input conditions, the attacker must guess 22 nibbles from K^2, 5 nibbles from K^3 (i.e., $k_{in} = 108$), and 4 nibbles from x^2 for state-testing (i.e., $s_{in} = 16$ bits). The specific bits involved in these guesses are:

  $K^2[1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 26, 28, 31]$,
  $K^3[4, 12, 19, 24, 31]$,
  $x^2[1, 8, 9, 23]$.

From each state x^1, these choices result in $2^{108+16} = 2^{124}$ possible values for $\widetilde{x}^1$.

- **Lower Part:** On the output side, the attacker must guess 9 nibbles from u^{15} (key guess) and 3 nibbles from y^{13} (state-test, so $s_{out} = 12$ bits). The relevant nibbles are:

$$u^{15}[31, 28, 23, 22, 17, 14, 5, 4, 0],$$
$$y^{13}[28, 23, 4].$$

  Additionally, 54 linear relations in K^1 are guessed, yielding $k_{out} = 36+54 = 90$. For every fixed set of the F `XOR`s, the lower part generates 2^{54} potential plaintexts based on the guessed K^1 values, and, for each associated ciphertext, considering 6 ($\approx 2^{2.58}$) possible output differences δ_{out}, the attacker constructs $2^{2.58+36+12} = 2^{50.58}$ possible values for $\widetilde{C}$.
- **Matching:** For each of the $2^{p-74} = 2^{48.99}$ repetitions, the procedure generates $2^{74} \cdot 2^{108+16} = 2^{198}$ upper triplets and $2^{74} \cdot 2^{2.58+54+36+12} = 2^{178.58}$ lower triplets. Recall that the structure naturally provides filtering over 128 bits (54 fix bits and 74 differences). Furthermore, 54 bits are filtered by exploiting key-schedule relationships between K^1 and K^2, as well as K^1 and K^3. This sequence of steps yields an overall filtering, giving $2^{48.99} \cdot 2^{198+178.58-128-54} = 2^{243.57}$ candidate matches.

At this stage of the attack, we have completely determined 128 key bits from K^1, as well as 44 independent bits from K^2, 10 independent bits from K^3, and 36 bits from u^{15}. Additionally, information from the state-test technique provides us with 16 bits in the upper state-test equations, and 12 in the lower ones. Let us first focus on the implications of the upper state-test equations, which are as follows:

$$x^2[1] = \mathbf{K^3[1]} \oplus S(z^0[29] \oplus \mathbf{K^2[29]}) \oplus y^1[1] \oplus y^1[8] \tag{17}$$
$$x^2[8] = \mathbf{K^3[8]} \oplus S(z^0[15] \oplus \mathbf{K^2[15]}) \oplus S(z^0[20] \oplus \mathbf{K^2[20]}) \oplus y^1[8] \tag{18}$$
$$x^2[9] = \mathbf{K^3[9]} \oplus S(z^0[0] \oplus \mathbf{K^2[0]}) \oplus S(z^0[21] \oplus \mathbf{K^2[21]}) \oplus y^1[8] \tag{19}$$
$$x^2[23] = \mathbf{K^3[23]} \oplus S(z^0[30] \oplus \mathbf{K^2[30]}) \oplus y^1[9] \oplus y^1[23], \tag{20}$$

where the unknowns are indicated in bold. Examining these equations more closely, we observe that 2 out of the 4 equations — specifically, equations(17) and(20) — involve only a single unknown column from K^3 and a single unknown column from K^2: the column 23 in Equation (20) involves unknown column 23 of K^3 and unknown column 30 from K^2; and the column 1 in Equation (17) that involves unknown column 1 of K^3 and unknown column 29 from K^2. In each of these columns, 2 bits are already determined by our knowledge of K^1. Thus, these two equations each present $2 + 2 = 4$ unknown bits, that will fix everything in the equation and will be verified with a probability of 2^{-4} each. Resolving these produces a unique solution for the 8 remaining bits influenced by these equations.

The other two state-test equations (Equation (18) and Equation (19)) each involve one unknown column from K^3 and two from K^2: column 9 in Equation (19) involves unknown column 9 from K^3 and unknown columns 21 and 0 from K^2, while column 8 in Equation (18) involves unknown column 8 from K^3 and unknown columns 15 and 20 from K^2. As before, with two bits per column predetermined by K^1, we have 2 unknown bits per column, which means that for each equation we have $2^{6-4} = 2^2$ solutions, while we are determining 6 additional bits.

Hence, in total, after processing these upper state-test equations, we are left with 2^4 solutions and an additional direct information on $4 + 4 + 6 + 6 = 20$ further key bits.

Consequently, we obtain $2^{p-74+194.58+4} = 2^{p+124.58} = 2^{247.57}$ key candidates for $218 + 20 = 238$ explicitly determined bits, with 12 equations from the lower state-test technique still to process. Notably, this approach avoids the exponential cost typically associated with guessing the remaining bits: the equations can be solved sequentially via efficient precomputations. For example, equations such as 17 or 20 can be solved by brute-forcing two bits that then uniquely determine the remaining two, effectively applying a 2^{-2} filter and yielding a single candidate in each case. The remaining two equations can be handled similarly, either via efficient enumeration at 2^4 cost per equation or by precomputing the number of solutions as a simplified instance of the general strategy described in Section 3.3, thus directly accounting for a 2^2 cost per equation.

4.5 Using the First Round of the Distinguisher in the Key Recovery

Using the idea we presented in Section 3.2 in order to filter further the kept candidates, we will consider the first round of the distinguisher. This will also recover more key information. We can check, out of the three active columns in that round, we only need to guess 5 additional keybits of information to determine everything in these three transitions : $K_1^3[31], K_3^3[31], K_1^3[19], K_3^3[19], K_1^3[1]$. In order to simplify this step, and as described in the previous section, we have fixed the output difference of this transition to not consider cluster, and only keep the transitions that generate the difference 4 in the three active columns, which happens with a probability of 2^{-6}. As there are 2^5 possibilities for the unknown keybits, and we have a probability of satisfying the conditions of 2^{-6}, by doing this we determine 5 additional keybits : $K_1^3[31], K_3^3[31], K_1^3[19], K_3^3[19], K_1^3[1]$, and we keep $2^{5-6} = 1/2$ of the previous candidates, so we have now $238+5 = 243$ keybits determined and 12 bits of state-test equations, and we have $2^{p-74+194.58+4-1} = 2^{p+123.58} = 2^{246.57}$ candidates. Please note that we do not have to pay the factor of 2^5 either, as we can solve the small independent equations again step by step using small precomputations that directly give us the solutions.

4.6 Recovering the Full Key: Using the Key-schedule and Final State-test Equations

At this point, we have 207 keybits from the upper round and 36 keybits from u^{15}. Given that the key schedule of `ARADI` is fully linear, these 36 key guesses for u^{15}

can be represented as linear equations in the $49 = 256 - 207$ unknown master key bits, forming a 36×49 coefficient matrix. By reducing this matrix to row-echelon form, the system can be further simplified, expressing all 36 equations in terms of just $13 = 49 - 36$ remaining variables. Thus, each unresolved bit appearing in the remaining state-test equations (21),(22), and(23) can be rewritten as a function of these 13 variables alone. The specific state-test equations on y^{13} are:

$$y^{13}[28] = u^{14}[28] \oplus S^{-1}(z^{14}[1] \oplus u^{15}[1]) \oplus S^{-1}(z^{14}[18] \oplus u^{15}[18]) \oplus x^{14}[28] \quad (21)$$

$$y^{13}[23] = u^{14}[28] \oplus S^{-1}(z^{14}[12] \oplus u^{15}[12]) \oplus S^{-1}(z^{14}[29] \oplus u^{15}[29]) \oplus x^{14}[23] \quad (22)$$

$$y^{13}[4] = u^{14}[4] \oplus S^{-1}(z^{14}[10] \oplus u^{15}[10]) \oplus S^{-1}(z^{14}[27] \oplus u^{15}[27]) \oplus x^{14}[4] \quad (23)$$

To solve these equations, we apply the precomputation technique described in Section 3.3. For this system, the relevant parameters are: $\mathsf{nv} = 36$, $\mathsf{rk} = 36$, $\mathsf{r} = 13$, and $\mathsf{rs} = 12$. Our precomputation therefore requires $2^{36+36+13-12} = 2^{73}$ entries, a table we construct and store in memory. In the final step of the attack, for each of the $2^{p+123.58}$ remaining candidates, we compute the known values for the $\mathsf{nv} + \mathsf{rk} = 36 + 36 = 72$ bits and use them to probe our precomputed table.

This process yields two possible assignments for the 13 unresolved bits, resulting in $2^{p+123.58+1} = 2^{122.99+124.58} = 2^{247.57}$ full 256 bits master key candidates.

4.7 Complexities of the Attack

We have a precomputation cost of 2^{73} in time and memory. The described attack recovers 256 bits of the master key. The final time complexity of this attack is :

$$\mathcal{T} = \mathcal{T}_1 + \mathcal{T}_2 + \mathcal{T}_3.$$

The first part of the attack is generating the upper and lower triplets in the differential-MITM attack, and keeping the pairs that provide a viable match:

$$\begin{aligned}\mathcal{T}1 &= 2^{122.99-74}(2^{108+16}2^{74} + 2^{2.58}2^{36+54+12}2^{74} + 2^{124+74}2^{104.58+74}2^{-128}2^{-54}) \\ &= 2^{246.99} + 2^{227.57} + 2^{243.57} \approx 2^{246.99}C_U\end{aligned}$$

where C_U is the cost of finding the upper triplets.

The second part performs the additional filtering of the matches regarding the state test equations of the upper part, the first round of the distinguisher, and the final state-test equations:

$$\mathcal{T}2 = 2^{243.57}2^4C_1 + 2^{246.5}C_2 + 2^{247.57}C_3,$$

where C_1 and C_2 represent simple computations of solving small systems of equations, and C_3 represents the cost of solving the linear system of the key schedule with 13 remaining variables, and checking the match with the precomputed table of size 2^{73}.

The last part deals with checking, for the remaining candidates, which is the correct one with two extra pairs of plaintext-ciphertext (so we have a probability of 2^{-256} of retaining a bad candidate).

$$\mathcal{T}3 = 2^{247.57} C_E + 2^{247.57-128} C_E,$$

where C_E is the cost of one 14-round `ARADI` encryption.

We can therefore approximate the overall time complexity

$$\mathcal{T} \approx 2^{247.57} C_E,$$

that is faster than brute force by a factor of $2^{8.42}$

As the attacker can choose the list to save between the triplets of the upper part and the triplets of the lower part, the memory complexity of this attack is :

$$\mathcal{M} = 2^{2.58} 2^{36+54+12} 2^{74} = 2^{178.58}$$

4.8 On the Data Complexity

The data complexity of the attack as described until now would be the whole code book. However, as demonstrated in [BDD+23], the data complexity of the attack can be further reduced by fixing a bits of the plaintext, thereby restricting the plaintexts to a subset of the full codebook. Specifically, by selecting P and all candidates $\widetilde{P}$ such that they share prescribed values on a specific bits, one effectively limits the working set to 2^{n-a} possible plaintexts. For this it is possible to precompute and store all the $\widetilde{C}$ associated to the 2^{n-a} restricted data that are being considered, and during the lower part of the attack, there is no need to call for decryptions, but directly check if it belongs to the set. While this reduction decreases the necessary data complexity, it also proportionally reduces the probability of finding a candidate such that the differential holds. In particular, although the chosen P always satisfies the a-bit constraint, each associated $\widetilde{P}$ satisfies it with probability 2^{-a}. Thus, the attack needs to be repeated an additional factor of 2^a, which is compensated with the fact that there is a smaller fraction of lower triplets kept, also determined by the same proportion, maintaing a same time complexity. Nevertheless, there needs to be enough available data to perform the attack, that is why the original condition proposed in [BDD+23] is that $p + a \leq n - a$.

We have realised that this technique, that was proposed as generic, cannot that easily be applied when using certain structures: if none of the fixed bits belongs to the data but to internal states, the conditions and the complexity formulas cannot stay the same. This is the case of our attack on `ARADI` . In order to reduce the data complexity below the whole code book, we have two options.

Option 1. Fixing anyway some bits of the plaintext: This will be translated into an asymmetric structure, that will have, in our case, a smaller size for the lower triplets, and also the probability of matching through the structure

will be reduced by 2^{-a}. This, added to the probability of having $\widetilde{P}$ falling in the same set, that also has probability 2^{-a}, modifies the conditions to $p+2a \leq n-a$. Regarding the time complexity, the first term of the time complexity ($\mathcal{T}_1$) of the differential-MITM attack will be multiplied by 2^{2a}, the second by 2^a and the third, the matching term, would stay the same. if we choose $a = 1$, the time bottleneck will become $\mathcal{T}_1 = 2^{248.99} C_U$, and the data

$$\mathcal{D} = 2^{128-1} = 2^{127}.$$

Option 2. Guessing keybits that allow to fix plaintext bits: In order to reduce the data complexity, we could consider that a bits out of the guessed bits of K^1 by the lower part are now instead guessed by the upper part. This allows to fix values of the structure in the plaintext. The inequality now stays the same as before: $p + a \leq n - a$. The time complexity will be affected by this. The common factor is increased by 2^a. The term associated to the upper part of the attack will be increased by the same amount, but in our case, it is also decreased by the same amount, as we can now directly apply a filtering conditions from the key-schedule (so this term is overall increased only by 2^a from the common factor). The term associated to the lower part is decreased by a factor 2^{-a} as we only keep the $\widetilde{P}$ that verify the a conditions, and by another additional factor of 2^{-a} as we are now guessing less bits here, so overall decreased by a factor of 2^{-a} when taking into account the common factor. The matching term has a less key-schedule conditions to apply, but it is also decreased by the second term having a smaller factor of 2^{-2a}, and taking into account the common factor it stays the same.

Overall, in our case, this option seems more interesting, as we have a less restrictive condition. We can choose $a = 2$, and the time complexity will be the same as in the first option $\mathcal{T} \approx 2^{248.99} C_U$, but the data complexity is now better:

$$\mathcal{D} = 2^{128-2} = 2^{126}.$$

Several trade-offs are possible.

5 Conclusion

In this work, we have presented the first dedicated cryptanalytic attack covering 14 out of 16 rounds of ARADI. Out attack uses differential-MITM attack, and exploits some new ideas, that not only enhance our specific attack but can actually be seen as generic improvements of this family of attack, and that we have generalized in the paper. Additionally, we have refined the estimation of differential probabilities and provided a more accurate understanding of the best existing distinguishers. We also demonstrated that the data reduction technique proposed in the original differential-MITM attack context does not directly translate when initial structures are involved, and have supplied corrected data complexity formulas in 4.8.

While the overall computational complexity of our attack remains high, our results indicate that 14-round ARADI does not achieve the full 256-bit security margin, highlighting a potential gap in the cipher's security. However, extending such cryptanalysis to the full 16 rounds of ARADI appears to remain a formidable challenge, likely necessitating fundamentally new ideas. It is noteworthy that the differential-MITM attack strategy currently surpasses all previously known approaches for this cipher.

Acknowledgements. This research is funded by the European Union via grants (ERC2023-COG, SoBaSyC, number 101125450) and (ERC starting researcher grant ReSCALE, number 101041545). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

References

ABD+23. Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The QARMAv2 Family of Tweakable Block Ciphers. *IACR Trans. Symmetric Cryptol.*, 2023(3):25–73, 2023.

ADG24. Roberto Avanzi, Orr Dunkelman, and Shibam Ghosh. A note on ARADI and LLAMA. Cryptology ePrint Archive, Paper 2024/1328, 2024.

ADG25. Roberto Avanzi, Orr Dunkelman, and Shibam Ghosh. Differential Cryptanalysis of the Reduced Pointer Authentication Code Function Used in Arm's FEAT_PACQARMA3 Feature. *IACR Trans. Symmetric Cryptol.*, 2025(1):380–419, 2025.

AKM+24. Zahra Ahmadian, Akram Khalesi, Dounia M'foukh, Hossein Moghimi, and María Naya-Plasencia. Improved differential meet-in-the-middle cryptanalysis. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I*, volume 14651 of *Lecture Notes in Computer Science*, pages 280–309. Springer, 2024.

Ava17. Roberto Avanzi. The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

BB93. Ishai Ben-Aroya and Eli Biham. Differential Cryptanalysis of Lucifer. In Douglas R. Stinson, editor, *Advances in Cryptology – Proceedings of CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 1993.

BCG+12. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga

Yalçin. PRINCE - a low-latency block cipher for pervasive computing applications - extended abstract. In *Advances in Cryptology – Proceedings of ASIACRYPT*, pages 208–225, 2012.

BDD+23. Christina Boura, Nicolas David, Patrick Derbez, Gregor Leander, and María Naya-Plasencia. Differential meet-in-the-middle cryptanalysis. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 240–272. Springer, 2023.

BDG+25. Yanis Belkheyar, Patrick Derbez, Shibam Ghosh, Gregor Leander, Silvia Mella, Léo Perrin, Shahram Rasoolzadeh, Lukas Stennes, Siwei Sun, Gilles Van Assche, and Damian Vizár. Chilow and chichi: New constructions for code encryption. In *Advances in Cryptology – Proceedings of EUROCRYPT*, volume 15601 of *Lecture Notes in Computer Science*, pages 212–243. Springer, 2025.

BEK+20. Dusan Bozilov, Maria Eichlseder, Miroslav Knezevic, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. PRINCEv2 - more security for (almost) no overhead. In *Proceedings of Selected Areas in Cryptography (SAC)*, volume 12804 of *LNCS*, pages 483–511. Springer, 2020.

BFG+24. Emanuele Bellini, Mattia Formenti, David Gérault, Juan Grados, Anna Hambitzer, Yun Ju Huang, Paul Huynh, Mohamed Rachidi, Raghvendra Rohit, and Sharwan K. Tiwari. CLAASPing ARADI: Automated analysis of the ARADI block cipher. Cryptology ePrint Archive, Paper 2024/1324, 2024.

BGG+23. Emanuele Bellini, David Gerault, Juan Grados, Yun Ju Huang, Mohamed Rachidi, Sharwan Tiwari, and Rusydi H. Makarim. CLAASP: a cryptographic library for the automated analysis of symmetric primitives. Cryptology ePrint Archive, Paper 2023/622, 2023.

BJK+16. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology – Proceedings of CRYPTO*, pages 123–153, 2016.

BNS14. Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and improving impossible differential attacks: Applications to clefia, camellia, lblock and simon. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 179–199. Springer, 2014.

BR22. Tim Beyne and Vincent Rijmen. Differential cryptanalysis in the fixed-key model. In *CRYPTO 2022*, volume 13509 of *LNCS*, pages 687–716. Springer, 2022.

BRRT24. Emanuele Bellini, Mohamed Rachidi, Raghvendra Rohit, and Sharwan K. Tiwari. Mind the composition of toffoli gates: Structural algebraic distinguishers of ARADI. Cryptology ePrint Archive, Paper 2024/1559, 2024.

BS90. Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference,*

Santa Barbara, California, USA, August 11-15, 1990, Proceedings, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

BS91. Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptol.*, 4(1):3–72, 1991.

CGL+23. Federico Canale, Tim Güneysu, Gregor Leander, Jan Philipp Thoma, Yosuke Todo, and Rei Ueno. SCARF - A Low-Latency Block Cipher for Secure Cache-Randomization. In *USENIX Security Symposium*, pages 1937–1954. USENIX Association, 2023.

CSNS25. Debasmita Chakraborty, Soumya Sahoo, Phuong Hoa Nguyen, and Santanu Sarkar. An automated model to search for differential meet-in-the-middle attack: Applications to andrx ciphers. *IACR Cryptol. ePrint Arch.*, page 1249, 2025.

DE25. Patrick Derbez and Marie Euler. Minimalist model for impossible differentials. Cryptology ePrint Archive, Paper 2025/1552, 2025.

DG25. Orr Dunkelman and Shibam Ghosh. Improved key-recovery attacks on ARADI. *IACR Cryptol. ePrint Arch.*, page 1227, 2025.

DR07. Joan Daemen and Vincent Rijmen. Plateau characteristics. *IET Information Security*, 1:11–17, 2007.

DSP07. Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved meet-in-the-middle attacks on reduced-round DES. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.

DZW25. Weiqing Deng, Jianing Zhang, and Haoyang Wang. Improved differential meet-in-the-middle cryptanalysis on SIMON and piccolo (full version). *IACR Cryptol. ePrint Arch.*, page 741, 2025.

GMW24. Patricia Greene, Mark Motley, and Bryan Weeks. ARADI and LLAMA: Low-latency cryptography for memory encryption. Cryptology ePrint Archive, Paper 2024/1240, 2024. https://eprint.iacr.org/2024/1240.

HJ25. Xichao Hu and Lin Jiao. Partial-guess, pre-sieve, greedy-search - new unified key recovery framework of impossible boomerang attacks: Full-round attack on ARADI. Cryptology ePrint Archive, Paper 2025/056, 2025.

HSH+08. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60. USENIX Association, 2008.

KH24. Donggeun Kwon and Seokhie Hong. Side-channel attack on ARADI. Cryptology ePrint Archive, Paper 2024/1987, 2024.

KKL+24. Sunyeop Kim, Insung Kim, Dongjae Lee, Deukjo Hong, Jaechul Sung, and Seokhie Hong. Byte-wise equal property of ARADI. Cryptology ePrint Archive, Paper 2024/1772, 2024.

LMM91. Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *EUROCRYPT '91*, volume 547 of *LNCS*, pages 17–38. Springer, 1991.

MNPN25. Dounia M'Foukh, María Naya-Plasencia, and Patrick Neumann. The state-test technique on differential attacks: a 26-round attack on CRAFT and other applications. Cryptology ePrint Archive, Paper 2025/1597, 2025.

NGJE25. Marcel Nageler, Shibam Ghosh, Marlene Jüttler, and Maria Eichlseder. Autodiver: Automatically verifying differential characteristics and learning key conditions. *IACR Trans. Symmetric Cryptol.*, 2025(1):471–514, 2025.

PT22. Thomas Peyrin and Quan Quan Tan. Mind your path: On (key) dependencies in differential characteristics. *IACR Transactions on Symmetric Cryptology*, 2022(4):179–207, 2022.

Ste. Stefan Kölbl. CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives. https://github.com/kste/cryptosmt.

A Weak-Key Space For 9-round Differential Distinguisher

The following 29 linear equations on the round keys must be satisfied in order to obtain a valid pair of states conforming to the distinguisher.

$$K_0^4[13] \oplus K_0^4[20] \oplus K_0^4[28] = 0$$
$$K_1^4[13] \oplus K_1^4[20] \oplus K_1^4[28] = 0$$
$$K_0^4[14] \oplus K_0^4[21] \oplus K_0^4[29] = 0$$
$$K_1^4[14] \oplus K_1^4[21] \oplus K_1^4[29] = 0$$
$$K_1^4[2] \oplus K_1^4[10] \oplus K_1^4[17] = 1$$
$$K_1^4[3] \oplus K_1^4[11] \oplus K_1^4[26] = 1$$
$$K_1^4[4] \oplus K_1^4[19] \oplus K_1^4[27] = 1$$
$$K_0^4[6] \oplus K_0^4[21] = 0$$
$$K_1^4[6] \oplus K_1^4[21] = 1$$
$$K_0^4[7] \oplus K_1^4[7] \oplus K_2^4[7] \oplus K_3^4[7] \oplus K_0^4[22] \oplus K_1^4[22] \oplus K_2^4[22] \oplus K_3^4[22] \oplus K_0^4[30] \oplus K_1^4[30] \oplus K_2^4[30] \oplus K_3^4[30] = 1$$
$$K_1^5[0] = 1$$
$$K_0^5[3] = 0$$
$$K_1^5[4] \oplus K_1^5[15] \oplus K_1^5[18] = 1$$
$$K_0^5[6] \oplus K_0^5[11] \oplus K_0^5[25] = 0$$
$$K_1^5[6] \oplus K_1^5[11] \oplus K_1^5[25] = 0$$
$$K_0^5[8] \oplus K_0^5[16] \oplus K_0^5[21] = 0$$
$$K_0^5[10] \oplus K_0^5[29] = 0$$
$$K_1^5[10] \oplus K_1^5[29] = 1$$
$$K_0^5[19] = 0$$
$$K_1^5[19] = 0$$
$$K_1^5[31] = 1$$
$$K_0^6[10] = 0$$
$$K_1^6[10] = 1$$
$$K_0^6[31] = 0$$
$$K_1^6[31] = 1$$
$$K_0^7[3] = 0$$
$$K_1^7[3] = 1$$
$$K_0^8[3] = 0$$
$$K_1^8[3] = 1$$

Characteristics that exhibit this type of key dependence are known as *conditional characteristics* [BB93]. In these equations, $K_i^r[j]$ denotes the j-th bit in the i-th row of the r-th round key. Moreover, we have investigated in detail the origins of this linear constraint space. The key reason lies in specific properties of the `ARADI` 's linear layer. From the structure of the linear layer, it is evident that each bit of the state z^r (which results from applying the linear layer Λ_{r-1} to y^{r-1}) depends on three bits of y^{r-1}. This dependency pattern provides insight into why such a system of equations arises. To explain this phenomenon, let us focus on the immediate SBox operation that follows the addition of the round key K^r. For any round r, and for each active nibble i, if the differential transition through the SBox is $a \to b$, then the following condition must hold:

$$(z^r[i] \oplus K^r[i]) \in \mathcal{B} = \{x \mid S(x \oplus a) \oplus S(x) = b\},$$

where S is the SBox and $\mathcal{B}$ is the set of values that generates the output difference b given the input difference a, sometimes called xDDT. Now, suppose that under the action of the linear layer, there exist four positions i_1, i_2, i_3, and i_4, and three output positions j_1, j_2, j_3, such that:

$$\begin{aligned}
z^r[j_1] \oplus K^r[j_1] &= K^r[j_1] \oplus y^{r-1}[i_1] \oplus y^{r-1}[i_2] \oplus y^{r-1}[i_3] \in \mathcal{B}_{j_1}, \\
z^r[j_2] \oplus K^r[j_2] &= K^r[j_2] \oplus y^{r-1}[i_1] \oplus y^{r-1}[i_4] \oplus y^{r-1}[i_3] \in \mathcal{B}_{j_2}, \\
z^r[j_3] \oplus K^r[j_3] &= K^r[j_3] \oplus y^{r-1}[i_2] \oplus y^{r-1}[i_4] \oplus y^{r-1}[i_3] \in \mathcal{B}_{j_3},
\end{aligned}$$

with each $\mathcal{B}_{j_k}$ corresponding to the valid input values for the SBox differential at nibble j_k. If we sum these three equations, the dependencies on the intermediate state bits y^{r-1} cancel out due to the structure of the additions, resulting in a new linear equation involving only the key bits $K^r[j_1]$, $K^r[j_2]$, and $K^r[j_3]$. The existence of such a linear relation depends on particular patterns within the sets $\mathcal{B}_{j_1}, \mathcal{B}_{j_2}$, and $\mathcal{B}_{j_3}$. For example, if $\mathcal{B} = \{0, 4, 8, 12\}$, the last two bits are always zero, yielding two additional linear equations on the involved nibbles of K^r. This structural property of the linear layer therefore directly results in the emergence of the observed set of key-dependent linear equations.