Improved Cryptanalysis of GIFT-64

Patrick Derbez[1], Baptiste Germon[1], Bastien Michel[2] and María Naya-Plasencia[2]

[1] Univ Rennes, Inria, CNRS, IRISA, France
patrick.derbez@inria.fr,baptiste.germon@inria.fr
[2] Inria Paris, France
bastien.michel@inria.fr,maria.naya_plasencia@inria.fr

Abstract. In this paper, we propose new differential attacks against the block cipher GIFT-64. First we demonstrate how the parallel matching algorithm proposed by Naya-Plasencia at CRYPTO'11 as an advanced list-merging algorithm can be leveraged to enhance differential attacks, overcoming a previously assumed bottleneck. By reducing the complexity of the pairs generation process whenever a non-linear filter is available, this approach enabled us to mount a new differential attack against 25-round GIFT-64 in the related-key setting.
Then we use the differential Meet-in-the-Middle cryptanalysis technique introduced by Boura et al. at CRYPTO'23 to improve the differential attacks recently proposed by Chang et al. at CT-RSA'25, leading to the best known attacks against GIFT-64 in the single-key setting, both in term of number of rounds and of complexity.

Keywords: symmetric cryptanalysis · GIFT · differential attacks · key-recovery · list merging algorithms · differential meet-in-the-middle attacks

1 Introduction

Differential cryptanalysis, introduced by Biham and Shamir in [BS91], is probably the most well studied and powerful class of cryptanalysis. Since its creation it has become mandatory for cryptographers to provide security criteria and design strategies to prevent a cipher from being vulnerable to this type of cryptanalysis. The core idea behind differential cryptanalysis is to study the propagation of an input difference throughout the encryption and to expose an unexpected propagation behaviour. Indeed, the first part of differential attacks is to exhibit a *differential* which is a couple of input/output difference where the probability that the input difference propagates to the output difference after some rounds of encryption is higher than the expected behaviour of a random permutation. Then, the attacker can add rounds before and after the *differential distinguisher* in order to mount a key-recovery attack. The first part (i.e. finding a high-probability differential) has received a lot of attention from the community ([FJP13, SHW+14, RGMS22, Knu95]) and is still an extensively studied topic. Researchers have developed really sophisticated approaches based on automated tools in order to find the highest probability possible for a differential on a given cipher. Yet, this may not lead to the best key-recovery attack.

Indeed, the key recovery step is a complex algorithmic problem that has been subject to many improvements and a lot of techniques have been developed to optimize this step. One can cite the use of *data structures* in [BS93] to lower the data complexity, the *early abort technique* of [LKKD08] to reduce the number of plaintext pairs to consider or *dynamic key guessing* from [WWJZ18, QHS16] to reduce the guessed key space. Unfortunately, it remains unclear how to find an optimal key-recovery strategy whether it is the pair generation that is the bottleneck or if the key guessing is too costly. Recently, Boura *et al.*

[BDD+24] have developed an automated tool to search for the best key guessing strategy which is a significant step toward the goal of having an efficient tool that would output the optimal key-recovery approach. But it only applies to a restricted class of ciphers and its generalization remains an open problem.

Additionally, a new key-recovery strategy, namely differential Meet-in-the-Middle (MitM), has been introduced in [BDD+23], later improved in [AKM+24, SYL23] and generalized in [SLY+24]. This approach aims at recovering the possible pairs of messages and associated key material through a meet-in-the-middle (MitM) procedure. In classical key-recovery of differential attack, the adversary first generates the pairs and then finds the correct partial keys for each pair, whereas in differential MitM attacks, he directly generates pairs of plaintexts along with the correct key, does the same for the ciphertexts and then search for a matching pair. Some variants of this attack use more advanced data structures that can cover one or several rounds such as in [AKM+24].

Lastly, Song *et al.* recent work [SLY+24] presented at Asiacrypt 2024 extends the concept of differential MitM to *rectangle attacks* and introduce the generic classical differential attack (`GCDA`) and generalized differential MitM attacks (`GDMA`) by considering an enhanced key guessing strategy. For example in the case of classical differential attacks, guessing some key bits before generating pairs can hugely improve the filtering part and therefore decrease the amount of pairs to deal with through the attack.

`GIFT-64`. `GIFT-64` is an instance of the `GIFT` family of lightweight block ciphers, designed by Banik et al. [BPP+17]. Since its proposal in 2017, it has received a lot of attention from the cryptanalysis community, as summarized in Table 1. The best known previous attacks in the single key setting reached 21 rounds, and the best related key attacks reached 26 rounds.

Our contributions Whereas we consider the classical differential attacks or the differential MitM attacks, it appears to us that those attacks can be improved and/or new trade-offs can be provided by considering non-linear filters. In the case of differential attacks, we will use this concept to filter the pairs retained for analysis during the key-recovery phase. For differential Meet-in-the-Middle attacks, we will consider more complex structures than the previous ones, covering two rounds in our case. We apply this ideas in particular to different existing attacks of `GIFT-64`, providing some of the best overall attacks, and in particular the first attack reaching 22 rounds in the single-key setting, as can be seen in Table 1.

Organization The rest of the paper is organized as follows. Section 2 recalls the principles behind both classical differential and differential MitM attacks, as well as a detailed description of the *parallel matching* algorithm. In Section 3, we apply the *parallel matching* algorithm to reduce the complexity of pairs generation in the former best known related-key attack on 25-round version of `GIFT-64`. In Section 4, we mount differential MitM attacks, each relying on a 2-round structure, against respectively 21 rounds and, for the first time, 22 rounds of `GIFT-64` in the single-key setting.

2 Preliminaries

In this section we recall how both differential attacks and differential meet-in-the-middle attacks are mounted against a block cipher. We also recall one of the efficient lists merging algorithms introduced in [Nay11].

Table 1: A summary of our results. Diff-MitM stands for differential MitM. RK stands for Related-Key and SK for Single-Key

Cipher	Rounds	Setup	Key space size	Time	Data	Memory	Type of Attack	Source
GIFT-64	18	SK	2^{128}	$2^{124.61}$	$2^{61.57}$	-	Differential-Linear	[WLHL24]
	19	SK	2^{128}	$2^{127.11}$	$2^{62.96}$	2^{60}	Linear	[SWW22]
		SK	2^{128}	2^{112}	2^{63}	2^{80}	Differential	[ZDY19]
	20	SK	2^{128}	$2^{112.68}$	2^{62}	2^{112}	Multi Diff	[CZD20]
		SK	2^{128}	$2^{101.68}$	2^{64}	2^{96}	Diff	[CZD20]
	21	SK	2^{124}	$2^{123.27}$	2^{64}	2^{112}	Multi Diff	[CWWH25]
		SK	2^{128}	$2^{121.66}$	2^{64}	2^{96}	Diff	[CZD20]
		SK	2^{124}	$2^{120.60}$	2^{64}	2^{96}	Diff	[CWWH25]
		SK	2^{124}	$2^{117.42}$	2^{64}	2^{96}	Diff	[CWWH25]
		SK	2^{124}	$2^{113.82}$	2^{59}	2^{90}	Diff MitM	Section 4
		SK	2^{124}	$2^{105.32}$	2^{59}	$2^{101.82}$	Diff MitM	Section 4
	22	SK	2^{124}	$2^{117.82}$	2^{64}	2^{55}	Diff MitM	Section 4
		SK	2^{124}	$2^{117.82}$	2^{61}	2^{61}	Diff MITM	Section 4
	23	RK	2^{128}	$2^{126.60}$	$2^{63.30}$	-	Boomerang	[LS19]
	24	RK	2^{128}	$2^{106.00}$	$2^{63.78}$	$2^{64.10}$	Rectangle	[JZZD21]
	25	RK	2^{128}	$2^{120.92}$	$2^{63.78}$	$2^{64.10}$	Rectangle	[JZZD21]
		RK	2^{120}	2^{107}	2^{51}	2^{49}	Differential	[BDD+24]
		RK	2^{120}	$2^{81.59}$	2^{51}	$2^{50.12}$	Differential	Section 3
	26	RK	2^{120}	$2^{123.23}$	$2^{60.96}$	$2^{102.96}$	Differential	[SWW21]
		RK	2^{120}	$2^{115.96}$	$2^{60.96}$	$2^{102.96}$	Differential	[BDD+24]
		RK	2^{120}	$2^{113.03}$	$2^{61.96}$	$2^{95.15}$	Differential	[CN25]
		RK	2^{120}	$2^{120.44}$	$2^{60.96}$	$2^{25.22}$	Differential	[CN25]

2.1 Classical differential attacks

In this part, we detail how differential attacks are mounted. First, a differential attack relies on the existence of a r_δ-round differential $(\delta_{in}, \delta_{out}) \in (\mathbb{F}_2^n)^2$, called a differential distinguisher, which has an associated probability of 2^{-p} such that $2^{-p} \gg 2^{-n}$ where n is the block size. In the context of key-recovery attacks, the attacker appends r_{in} rounds before the distinguisher and r_{out} rounds after. Let D_{in} be the set of differences such that δ_{in} propagates backward to D_{in} with probability 1 and $\mathcal{K}_{in}$ be the key bits involved in the r_{in} rounds. Similarly, δ_{out} propagates forward to D_{out} with probability 1 and we denote the key bits involved in the last r_{out} rounds by $\mathcal{K}_{out}$. An illustration of classical differential attacks is given in Figure 1. The goal for the attacker is to find the key that will lead the most pairs of plaintexts to follow the differential. To do so, the attacker follows several steps:

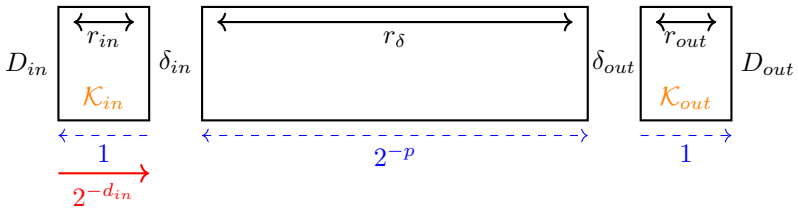

Figure 1: Illustration of differential attack

2.1.1 Acquiring data.

In order to exploit the distinguisher, the attacker must generates more than 2^p pairs that have a difference δ_{in} after r_{in} rounds. Usually, one uses *structures* of size $|D_{in}| = 2^{d_{in}}$ which are sets of plaintexts where $n - d_{in}$ bits are set to a fixed-value and the remaining d_{in} bits are unconstrained. As each structure contains $2^{d_{in}}$ plaintexts, one can generates roughly $2^{2d_{in}-1}$ pairs from one structure. Yet, the pairs formed may not lead to a difference δ_{in} after r_{in} rounds of encryption, in fact it only happens with probability $2^{-d_{in}}$. Consequently, each structure leads to $2^{d_{in}-1}$ pairs that have a difference δ_{in} after r_{in} rounds. Therefore, one must encrypt at least 2^s structures with $s = p - d_{in} + 1$. This leads to a data complexity $\mathcal{D}$ of at least $2^s \cdot 2^{d_{in}} = 2^{p+1}$ that is acquired with a time complexity of $2^{p+1} \cdot C_E$ where C_E denotes the cost of one encryption call.

2.1.2 Sieving pairs.

In the key recovery step, for each generated pair, the attacker will try to find all valid partial keys $k \in \mathcal{K}_{in} \cup \mathcal{K}_{out}$ such that the pair follows the differential. Therefore, reducing the number of pairs that will be tested is crucial. One common way of doing this is to build a hash table $\mathcal{H}$ indexed by the values of the $n - d_{out}$ inactive bits of each ciphertext. Indeed, if two ciphertexts do not share the same values on those bits then the corresponding pair does not belong to D_{out} and the pair cannot follow the differential. $\mathcal{H}$ is constructed on the fly during the data acquisition so it comes without additional complexity. Each entry of $\mathcal{H}$ contains approximately $2^{d_{in}-n+d_{out}}$ elements which leads to a total of $2^{2(d_{in}-n+d_{out})-1} \cdot 2^{n-d_{out}} = 2^{2d_{in}-n+d_{out}-1}$ generated pairs for a whole structure compared to $2^{2d_{in}-1}$ without the hash table. Overall, we will have to test $N = 2^s \cdot 2^{2d_{in}-n+d_{out}-1} = 2^{p+d_{in}-n+d_{out}}$ pairs in the key recovery step. Additionally, one can filter the pairs by considering non-linear constraints, we denote the cost of this operation for one pair by C_S and by $N' < N$ the number of pairs obtained after this filter. This will be discussed in more detail in Section 3.

2.1.3 Key guessing.

For a given pair (P, P') (and the corresponding ciphertext pair (C, C')), the attacker will search all possible values $k \in \mathcal{K}_{in} \cup \mathcal{K}_{out}$ such that respectively (P, P') and (C, C') have a difference δ_{in} and δ_{out} on the edges of the distinguisher. For each value of k, a counter is incremented when a valid pair is found and in the end the highest counter corresponds to the right key. We denote by C_{KR} the cost of finding such values for a given pair. The time complexity of this step is $N' \cdot C_{KR}$ and the memory complexity is negligible.

2.1.4 Complexity of a classical differential attack.

Finally, one can derive the following formulas for the time ($\mathcal{T}$), memory ($\mathcal{M}$) and data ($\mathcal{D}$) complexities of the attack:

$$\mathcal{T} = 2^{p+1} \cdot C_E + N \cdot C_S + N' \cdot C_{KR}, \quad \mathcal{M} = 2^{d_{in}}, \quad \mathcal{D} = 2^{p+1} \tag{1}$$

2.2 Differential meet-in-the-middle attacks

We will now detail another way to exploit a differential distinguisher to mount a key-recovery attack using an approach similar to MitM attacks for the key recovery part. The block cipher is divided into three parts: E_{u} (for the upper part), E_{d} (for the distinguisher), and E_{l} (for the lower part), such that $E = E_{\mathrm{u}} \circ E_{\mathrm{d}} \circ E_{\mathrm{l}}$. Both E_{u} and E_{l} correspond to the r_{in} and r_{out} rounds before and after the distinguisher. We denote by k_{in}, k_{out} and k_{inter} the cardinality of $|\mathcal{K}_{in}|$, $|\mathcal{K}_{out}|$ and $|\mathcal{K}_{in} \cap \mathcal{K}_{out}|$ respectively. We provide an illustration of the differential MitM technique in Figure 2.

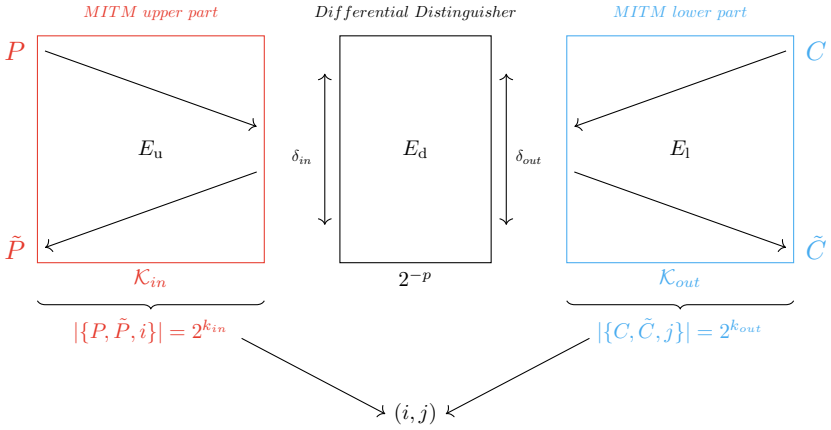

Figure 2: High level description of the differential MitM technique

For a given pair (P, C), we can generate $2^{k_{in}}$ $\tilde{P}$ plaintexts using E_u such that $E_u(P) \oplus E_u(\tilde{P}) = \delta_{in}$. Thus, for each element i of $\mathcal{K}_{in}$, we obtain a triplet $(P, \tilde{P}, i)$ that satisfies the distinguisher's input constraints after r_{in} rounds. Similarly, using E_l, we can generate for each j of $\mathcal{K}_{out}$, $2^{k_{out}}$ triplets $(C, \tilde{C}, j)$ such that $E_l^{-1}(C) \oplus E_l^{-1}(\tilde{C}) = \delta_{out}$.

A match is then made between the triplet lists $(P, \tilde{P}, i)$ and $(C, \tilde{C}, j)$, i.e., we check that $E(\tilde{P}) = \tilde{C}$ for the two considered triplets. In the case of equality, we keep the key pair (i, j) corresponding to the match, in the other case the key pair is rejected. There are $2^{k_{in}+k_{out}-k_{inter}}$ pairs to test and the probability of a match (i.e. the equality of two n-bit states) is 2^{-n}. The term $-k_{inter}$ comes from the fact that one should start by guessing the common bits of $\mathcal{K}_{in}$ and $\mathcal{K}_{out}$ to avoid considering triplets that differ on those bits.

As there exist pairs (P, C) that lead to no match one must repeat this process for 2^p distinct pairs (P, C) to ensure that at least one pair satisfies the differential distinguisher. Each part of the attack can be executed simultaneously, yielding the following time complexity:

$$\mathcal{T} = 2^p(2^{k_{in}} + 2^{k_{out}} + 2^{k_{in}+k_{out}-k_{inter}-n}) \tag{2}$$

An improvement to this method, called *parallel partitioning*, allows additional rounds r_s in the attack by adding a structure above E_u or below E_l. In the following, we will focus on the case where the structure is placed above E_u. Its initial state is P and corresponds to the final state of E_l, with its final state is called X_f and correspond to the first state of E_u. The attack process is then slightly different: we begin by fixing F bits in the structure. From these F fixed bits, we can use elements of $\mathcal{K}_{in}$ to compute F bits of X_f. Then, for the remaining 2^{n-F} values, we generate tuples $(X_f, \tilde{X}_f, i)$ in the same way as previously generated tuples $(P, \tilde{P}, i)$. Similarly, using the F fixed bits, we can use elements of $\mathcal{K}_{out}$ to compute F bits of P, and for the remaining 2^{n-F} possible values, we compute $C = E(P)$ and generate the tuples $(C_f, \tilde{C}_f, j)$ as in the original attack. For the matching phase, we can now filter the triplet pairs by verifying the consistency between $\tilde{X}_f$ and $\tilde{C}_f$ using the F equations coming from the structure. This results in an F-bit filtering. We can further enhance filtering by exploiting the linear equations from the unfixed part of the structure and the nonlinear equations from the attack, using precomputation tables, which can induce a time-memory trade-off. Let F_{add} denote this additional filtering. An illustration of this technique is given in Figure 3.

This process need to be repeated only $2^{p-(n-F)}$ times as each time we generate 2^{n-F} (X, C) pairs. At each repetition, the F fixed values are changed to generate new (X, C)

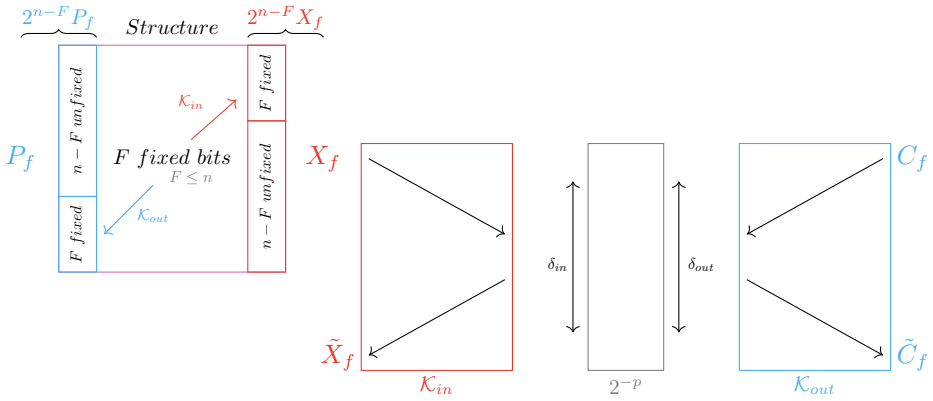

Figure 3: High level description of the differential MitM technique including structure

pairs that were never covered by previous iterations. The time complexity of this attack is

$$\mathcal{T} = 2^{p-(n-F)}\left(2^{n-F}2^{k_{in}} + 2^{n-F}2^{k_{out}} + 2^{2(n-F)+k_{in}+k_{out}-k_{inter}-F-F_{add}}\right)$$

which can be simplified as follows

$$\mathcal{T} = 2^{p}\left(2^{k_{in}} + 2^{k_{out}} + 2^{k_{in}+k_{out}-k_{inter}+n-2F-F_{add}}\right) \tag{3}$$

2.3 Efficient list merging algorithm: parallel matching

In this section, we describe an efficient list merging algorithm introduced in [Nay11], namely the *parallel matching*. Here, we keep the description as generic as possible but its use will be specified in the next section. Let $L_1, \ldots, L_M$ be M lists composed of uniformly distributed elements of $\mathbb{F}_2^l$. Let $T : (\mathbb{F}_2^l)^M \to \mathbb{F}_2$ be a boolean function. The goal is to efficiently obtain the list $L_{sol} \subseteq L_1 \times \cdots \times L_M$ with respect to the constraint $T = 1$ i.e. $L_{sol} = T^{-1}(\{1\}) \cap (L_1 \times \cdots \times L_M)$. In [Nay11], it was proven that every instance of such problem can be reduced to an equivalent problem with only two lists L_1 and L_2 of size 2^{l_1} and 2^{l_2}. From now on, we only consider this case where $M = 2$. Additionally, we are in the situation where the constraint T can be decomposed into z smaller constraints $t_1, \ldots, t_z$, so to say that there exists functions $t_i : \mathbb{F}_2^{2s} \to \mathbb{F}_2$, $f_i : \mathbb{F}_2^l \to \mathbb{F}_2^s$ and $f_i' : \mathbb{F}_2^l \to \mathbb{F}_2^s$ for all $i = 1, \ldots, z$ such that for all $(x_1, x_2) \in L_1 \times L_2$:

$$T(x_1, x_2) = 1 \Leftrightarrow \begin{cases} t_i(v_i, v_i') = 1 \\ v_i = f_i(x_1) \\ v_i' = f_i'(x_2) \end{cases} \quad \forall i \in \{1, \ldots, z\}$$

with s being an integer smaller than l. Typically, one can think of s as the size of a S-box, z the number of considered S-boxes, f_i and f_i' as functions extracting an s-bit word from a state and t_i as a condition on the values of extracted s-bit words. Let 2^{-p_1} be the probability that a random input $(\boldsymbol{x_1}, \boldsymbol{x_1'}) \in \mathbb{F}_2^{2s}$ verifies the first condition i.e. $t_1(\boldsymbol{x_1}, \boldsymbol{x_1'}) = 1$, likewise for $2^{-p_2}, \ldots, 2^{-p_z}$. The expected size of L_{sol} is $2^{-\sum_{i=1}^{z} p_i} \cdot 2^{l_1+l_2} =: P_T \cdot 2^{l_1+l_2}$.

2.3.1 Parallel matching.

The general idea of *parallel matching* is to compute in parallel two lists that will respectively contains elements that verify r and q conditions, with $z' = r + q$. Then, those lists are

merged together to obtain a list of elements of $L_1 \times L_2$ that verify z' conditions and then check the $(z - z')$ remaining conditions. Next, we describe the *parallel matching* algorithm more thoroughly in Algorithm 1.

Algorithm 1 Parallel matching algorithm

Input: Two lists L_1 and L_2 and a function T as previously described.
Output: The list $L_{sol} = T^{-1}(\{1\}) \cap L_1 \times L_2$.

1: From L_1, we build a hash table H indexed by $(v_1, \ldots, v_r)$.
2: Similarly, from L_2 we build a hash table H' indexed by $(v'_{r+1}, \ldots, v'_{r+q})$.
▷ *Each entry of H and H' respectively contains in average $2^{l_1 - rs}$ and $2^{l_2 - qs}$ elements.*
3: We build the list L_r of size $2^{2rs - \sum_{i=1}^{r} p_i} =: 2^{l_r}$ formed by all $(v_1, \ldots, v_r, v'_1, \ldots, v'_r)$ which satisfy $t_i(v_i, v'_i) = 1, \forall 1 \leq i \leq r$.
4: We build the list L_q of size $2^{2qs - \sum_{i=r+1}^{r+q} p_i} =: 2^{l_q}$ formed by all $(v_{r+1}, \ldots, v_{r+q}, v'_{r+1}, \ldots, v'_{r+q})$ which satisfy $t_i(v_i, v'_i) = 1, \forall r+1 \leq i \leq r+q$.
▷ *L_r and L_q respectively contains all possible solutions (not necessarily in $L_1 \times L_2$) to r and q conditions.*
5: With L_q and H' we build a new hash table H_q. For each $(\beta, \beta') \in L_q$, we add in H_q all the elements $(\beta, v'_1, \ldots, v'_z)$ of $H'[\beta']$. H_q is indexed by the values $(\beta, v'_1, \ldots, v'_r)$.
▷ *At this step, we have built elements of $\mathbb{F}_2^l \times L_2$ that verify q conditions out of the $(r+q)$ ones.*
6: **for each** $(\alpha, \alpha') \in L_r$ **do** ▷ *For all solutions to the r first conditions.*
7: **for each** $(v_1, \ldots, v_z) \in H[\alpha]$ **do** ▷ *For all elements of L_1 that are part of a solution to the r conditions.*
8: **if** $H_q[(v_{r+1}, \ldots, v_{r+q}, \alpha')] \neq \emptyset$ **then** ▷ *We found couples matching z' conditions*
9: **if** $(v_1, \ldots, v_z, v'_1, \ldots, v'_z)$ satisfies the remaining $(z - z')$ conditions **then**
10: $L_{sol} \leftarrow L_{sol} \cup \{(v_1, \ldots, v_z, v'_1, \ldots, v'_z)\}$
11: **return** L_{sol}

As proven in [Nay11], the time and memory complexities of the *parallel matching* algorithm are given by:

$$\mathcal{T}_{par} = 2^{l_r} + 2^{l_q} + 2^{l_1 + l_2 - \sum_{i=1}^{r+q} p_i} + 2^{l_1 + rs - \sum_{i=1}^{r} p_i} + 2^{l_2 + qs - \sum_{i=r+1}^{r+q} p_i} \tag{4}$$

$$\mathcal{M}_{par} = 2^{l_r} + 2^{l_q} + 2^{l_2} + 2^{l_2 + qs - \sum_{i=r+1}^{r+q} p_i} + 2^{l_1 + l_2 - \sum_{i=1}^{r+q} p_i} \tag{5}$$

The last term in $\mathcal{M}_{par}$ corresponds to the storage of L_{sol} which might not be necessary if we use the generated solutions on the fly.

Remarks. First, note that the lists L_r and L_q do not depend on the lists L_1 and L_2 and can thus be precomputed if the same parallel matching is applied on several pairs of lists (L_1, L_2). Second, it is common that if (a_1, a_2) is a solution of $T^{-1}(\{1\}) \cap L_1 \times L_2$ then (a_2, a_1) is a solution as well. By construction we can ensure that each row of H_q is sorted and therefore enumerate only the solution $(v_1, \ldots, v_z, v'_1, \ldots, v'_z)$ such that $(v_1, \ldots, v_z) \leq (v'_1, \ldots, v'_z)$. By selecting the lexical order, this saves a factor 2 in the size of L_r and in the final number of solutions, leading to the formula:

$$\mathcal{T}_{par} = 2^{l_r - 1} + 2^{l_q} + 2^{l_1 + l_2 - 1 - \sum_{i=1}^{r+q} p_i} + 2^{l_1 - 1 + rs - \sum_{i=1}^{r} p_i} + 2^{l_2 + qs - \sum_{i=r+1}^{r+q} p_i}.$$

3 Advanced sieving technique: application to 25-round GIFT-64

As explained in Section 2.1, the complexity of a differential attack is lower bounded by the number of pairs N that are generated from the data. For each of these pairs, the goal of the attacker is to construct all the possible tuples (P, P', k) where k is a value for $\mathcal{K}_{in} \cup \mathcal{K}_{out}$ that partially encrypt and decrypt the pair so that it reaches both the input and output of the differential. Whenever the average number of keys associated to a pair is smaller than 1, the number of pairs N can be the bottleneck of the whole differential attack. More precisely, generating the pairs is the bottleneck of a differential attack as soon as $N' \cdot C_{KR} < N$, meaning that the non-linear filter that can be applied on the initial N pairs should be higher than the cost of the key-recovery part associated to one pair. In this section we show how to use the parallel matching technique described Section 2.3.1 to generate the pairs in an efficient way, by directly handling non-linear filter and therefore outperforming the classical generation technique that relies on a hash table. To illustrate our technique we present an example on the block cipher `GIFT-64`.

Let us point out that, even though the related-key attack on 26 rounds of `GIFT-64` presented in [CN25] already used the parallel matching algorithm, the application there is very different to what we are proposing here. Their aim was to optimize the final key recovery steps of the attack, but the bottleneck given by the classic terms defined by the amount of data needed and the number of pairs to treat after the classical filtering was the same as classically, while in our case the aim is to reduce for the first time this bottleneck.

3.1 GIFT-64

`GIFT-64` is part of the `GIFT` family of lightweight block ciphers, designed by Banik et al. [BPP+17]. It features a 64-bit block size and a 128-bit key, with a structure composed of 28 rounds. As a classical Substitution-Permutation Network (SPN) cipher, its state is divided into 16 nibbles.

Each round begins with the XORing of the round key into the state. A distinctive aspect of `GIFT` is that the key is added only to specific bits of the state—precisely those at positions b such that $b \equiv 0 \pmod 4$ or $b \equiv 1 \pmod 4$. Next, a 4-bit S-box is applied in parallel to all nibbles, followed by a bit permutation. The S-box, along with its Difference Distribution Table (DDT) and the bit-permutation details of `GIFT-64`, can be found in Supplementary Material A. The 128-bit master key can be seen as eight 16-bit words stored in a register $K = k_7||k_6 \cdots ||k_1||k_0$. The key schedule begins by extracting the first 32 bits of the round key:

$$RK_r = k_1||k_0,$$

then the register K is updated:

$$k_7||k_6 \cdots ||k_1||k_0 \leftarrow k_1 \ggg 2||k_0 \ggg 12|| \cdots ||k_3||k_2.$$

In the following, we refer to the j-th bit of the i-th 16-bit word in the 128-bit master key K by $k_i[j]$ and $RK_r[16i + j]$ denotes the j-th bit of the i-th 16-bit word of the round key RK_r.

3.2 Previous attacks on 26-round GIFT-64

In [SWW21], Sun et al. presented a differential attack on 26 rounds of `GIFT-64` in the related-key setting. Their attack leveraged an 18-round related-key differential distinguisher with a probability of $2^{-p} = 2^{-58}$:

$$\texttt{0000 6000 0000 0600} \longrightarrow_{18r} \texttt{0000 0014 0000 0041}.$$

The corresponding difference in the 128-bit master key is defined as follows:[1]

0000 1400 0000 0000 0000 0000 0000 0000.

To extend this distinguisher into a full attack, the authors incorporated three key recovery rounds at the beginning and appended five additional key recovery rounds at the end. The differential propagation through the key recovery rounds is illustrated in Table 2. The i-th round key is added right after state O_i and there is no whitening round key. Since there are 8 bits with known difference on O_0, the adversary starts by asking for a structure of 2^{56} messages under the secret key such that all but those 8 bits take all the possible values on state O_0 and a similar structure of 2^{56} messages under the related key for which the value of the 8 constant bits is flipped. This allows to form 2^{112} pairs. Since we need $N = 2^{56+58} = 2^{114}$ pairs to get one following the differential, four such structures are necessary and therefore the data complexity would be $\varepsilon 2^{59} = 2^{60.96}$ chosen plaintext-ciphertext pairs for having a high success probability. The key-recovery step they proposed was not optimal and led to a time complexity of $2^{123.23}$. Boura *et al.* used their automatic tool in [BDD+24] to search for better key-recovery procedures, and the best one found had a time complexity of $2^{115.96}$ which is greater than N. In [CN25] the same distinguisher is used for building a better attack, improving the key recovery part using techniques from [BCF+21] by using some S-box properties.

We would like to point out that, despite the fact that the improvement techniques for key-recovery used in [CN25, BCF+21] might be useful to be combined with our new approach in other scenarios, in our `GIFT-64` attack application this could not improve the overall complexity. The reason is that the time complexity bottleneck is determined by the number of potential pairs to consider, and the remaining parts of the key-recovery step do not increase this complexity, as we will explain in the following.

Table 2: Differential attack on 26-round `GIFT-64` based on the 18-round related-key distinguisher from [SWW21].

S-box	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ΔI_0	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****
ΔO_0	****	****	****	****	1***	11**	*1**	****	*1**	****	1***	11**	****	****	****	****
ΔI_1	****	****	11**	****	****	****	11**	****	****	11**	****	****	****	11**	****	****
ΔO_1	...*	1...	.1..	..*.	...*	*...	.1..	..*.	1...	.1..	..*.	...*	*...	.1..	..*.	...*
ΔI_2	....	....	....	....	....	....	....	....	11**	*1**	....	....	....	....	11**	*1**
ΔO_2	....	....	....	....	....	....	....	....	.1..	..1.	....	....	....	....	.1..	..1.
ΔI_3	....	....	.11.	....	....	....	....	....	....	....	....	.11.	....	....	....	....
18-round related-key differential distinguisher																
ΔI_{21}	...1	.1..	....	....	....	....	....	....	.1..	...1	....	....	....	....	....	....
ΔO_{21}	****	***1	....	....	....	....	....	....	***1	****	....	....	....	....	....	....
ΔI_{22}	.**.	....	.**.	....	..*1	....	..**	....	*..*	....	*..1	....	**..	....	**..	....
ΔO_{22}	****	....	****	....	****	....	****	...	****	....	****	....	****	....	****	....
ΔI_{23}	.*.*	.*.*	.*.*	.*.*	*.*.	*.*.	*.*.	*.*.	.*.*	.*.*	.*.*	.*.*	*.*.	*.*.	*.*.	*.*.
ΔO_{23}	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****
ΔI_{24}	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****
ΔO_{24}	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****
ΔI_{25}	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****
ΔO_{25}	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****

3.3 Filtering wrong pairs

Actually, when the pairs are generated, they are filtered in order to remove the ones that cannot follow the differential. Indeed, the possible values and differences on state I_1

[1] In [SWW21], the least significant bit (LSB) was positioned on the left, differing from the original cipher's description [BPP+17]. Here, we adhere to the original notation, placing the LSB on the right.

are very restricted and this allows to decrease the number of pairs to deal with during the attack.

Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a non-linear function from n bits to m bits. Let A be a set of possible differences at the input of S and B a set of possible differences at the output of S. If one knows the exact values of plaintexts on the input of S then one can compute exactly the probability to satisfy the differential transition, we call this an *exact filter*. However, we might not know the exact values of the inputs due to previous key addition on some bits which renders the previous computation impossible without guessing those key bits. Still, we can compute an approximate filter, we refer to this by *compressed filter*. The *exact* and *compressed* filter are thus defined by:

Definition 1 (Exact/Compressed filters)**.** Let $K \subseteq \{0, \ldots, n-1\}$ be the subset of input bits where key material was added. Let $x \coloneqq (x_0, \ldots, x_{n-1}), y \coloneqq (y_0, \ldots, y_{n-1}) \in \mathbb{F}_2^n$. A pair (x, y) is a solution to the differential transition $A \to B$ if and only if $x \oplus y \in A$ and $S(x) \oplus S(y) \in B$. The compressed solution associated to this solution is a $(2n - |K|)$-bit tuple for which both x_i and y_i are stored if $i \notin K$ and only $x_i \oplus y_i$ is stored if $i \in K$. Let $L_{A,B}$ and $LC_{A,B}$ respectively be the set of solutions to $A \to B$ and the set of compressed solutions to $A \to B$. Then, the *exact filter* is computed by $|L_{A,B}|/(2^n \cdot |A|)$ and the *compressed filter* by $|LC_{A,B}|/(2^{n-|K|} \cdot |A|)$.

Example 1. Let us consider the S-box of `GIFT`. Let $A = \{0, \ldots, 15\}$ and $B = \{8\}$. In `GIFT-64`, the **SubCells** operation is preceded by a key addition on the two rightmost bits at each S-box. So instead of storing $(x_3, y_3, x_2, y_2, x_1, y_1, x_0, y_0)$ for each solution we only store the 6-bit word $(x_3, y_3, x_2, y_2, x_1 \oplus y_1, x_0 \oplus y_0)$ which is the associated compressed solution. There exists 16 solutions to the transition $A \to B$ and 14 associated compressed solutions. Therefore this differential transition has an *exact filter* of 2^{-4} and a *compressed filter* of $2^{-2.192}$. Here, guessing the key bits to use the exact filter instead of the compressed one is not worthwhile, as it would cost 2^2 for a filter gain of $2^{1.808}$.

3.4 New related-key attack on 25-round GIFT-64

On 26 rounds the number of pairs N is not the bottleneck, as the complexity of the attack is equal to $2N$. Yet, when considering only 25 rounds, by omitting the last round, the `KyRyDi` tool from [BDD+24] found that the complexity of the key-recovery step is $N \times 2^{-17.34}$ and therefore the bottleneck of the attack is the generation of the N pairs.[2] In the following, we describe an attack on 25 rounds that achieves a time complexity of $2^{89.59}$ while naively N would still be equal to 2^{114}. We thus show how to improve the complexity of a classical attack beyond the number of generated pairs, which was usually believe to be unreachable.

Filters on pairs In [BDD+24], authors looked at the filter on state I_1 coming from the restricted differences on state O_1. Over the 16 S-boxes, there are 4 S-boxes providing a filter of 2^{-0}, 4 other S-boxes providing a filter of $2^{-1.68}$, 6 S-boxes providing a filter of $2^{-1.83}$ and finally the 2 remaining S-boxes provide a filter of $2^{-2.19}$. This leads to a total filter of $2^{-22.08}$.

Generating pairs using parallel matching Let us use the parallel matching technique to generate the pairs required to perform the differential attack. Both the lists L_1 and L_2 are fulfilled with a full structure and are therefore of size 2^{56} each. We have 12 filters different from 2^{-0} and we use all of them in the algorithm so $z = z' = 12$. The components

[2] The file to generate the key-recovery against `GIFT` is already in the official repository of `KyRyDi` since it was used by their authors to mount the 26-round attack: https://gitlab.inria.fr/capsule/kyrydi

of the time complexity are $2^{l_r} = 2^{l_q} = 2^{2\times 6\times 4-11.04} = 2^{36.96}$, $2^{l_1+l_2-22.08} = 2^{89.92}$, $2^{l_1+6\times s-11.04} = 2^{l_2+6\times s-11.04} = 2^{68.96}$ with $s = 4$. Hence, the complexity of matching L_1 and L_2 under the constraints of the first layer of S-boxes is $2^{89.92}$ to generate the same amount of pairs. Since the attack requires 4 structures this procedure has to be performed 4 times, leading to a complexity of $2^{91.92}$.

Mounting the attack As we already exploited the filters to generate the pairs we re-ran the `KyRyDi` tool to find the best key-recovery procedure and it outputted an updated complexity of $2^{4.76}$ per pair. The same key is used between O_0 and I_1 and between I_{25} and O_{24} and the procedure starts by guessing it step by step to take advantage of the filter coming from I_{23} as quickly as possible. The resulting complexity of the attack is therefore $2^{91.92+4.76} = 2^{96.68}$ which is below the number of pairs that would have been generated through the classical pre-sieving technique. This highlights that, what we previously believed to be a bottleneck of differential attacks is not when using advanced techniques for the generation of pairs.

Improving the attack Actually, we do have more non-linear filter on pairs than the one coming from the first layer of S-boxes: we can go further by looking at the filter obtained through Super S-boxes. Indeed, two consecutive rounds of `GIFT`-64 can be represented as the parallel application of four Super S-boxes, each operating on 16-bit words and depending on 8 key bits. We therefore computed the filters associated to the four Super S-boxes from I_1 to O_2 as well as to the four Super S-boxes from O_{24} to I_{23}. As a result we obtained 2 filters of $2^{-8.07}$ and 2 of $2^{-7.7}$ for the upper Super S-boxes and marginal filters for the lower ones. As a consequence, we have a total filter of $2^{-31.54}$ and it can be fully used through the parallel matching algorithm. In this case, we only generate $2^{82.46}$ pairs with a complexity of $2^{82.46}$, however the cost per pair is increased from $2^{4.76}$ to $2^{7.13}$ and we thus end up with an attack with a lower complexity of only $2^{82.46+7.13} = 2^{89.59}$. The cost in memory of the parallel matching algorithm is $2^{58.12}$ (we ignore the last term as pairs can be considered on the fly) which is thereby the overall memory complexity of the attack.

Remarks In this improved version of the attack it seems that a part of the improvement comes from the fact that, by computing the filter on Super S-boxes, we actually compute the real probability of the differential from I_1 to O_2 since we do not use the assumption that rounds are independent anymore. It is also important to note that recently, [CHLW25] applied the quasidifferential framework from [BR22] and claimed that this related-key characteristic as a valid key space of $2^{-5} + 2^{-8} \approx 2^{-4.83}$. However, after running our own analysis based on the related-key extension proposed in [BDG25a] we found that the valid key space is at most 2^{-8}. Indeed, there are 256 quasidifferential trails with an absolute correlation of 2^{-58} which form a 8-dimensional vector space. This corresponds to 8 independent linear conditions and a key can only be valid is all those conditions are verified, leading to a key space of at most 2^{-8}. For the valid keys, the average probability is 2^{-50} even though the fixed-key probability ranges from 2^{-52} to $2^{-48.83}$. This means that our attack is actually a weak-key attack, able to attack only one key out of 256 but against the weak-keys it is on average $2^{58-50} = 2^8$ times faster than we just stated, leading to a total time complexity of $2^{81.59}$.

3.5 Application to other targets

We tried to apply the technique on other ciphers to improve previous differential attacks. Unfortunately, it seems that in most attacks the generation of the pairs is rarely the bottleneck of the time complexity. Still, we derive several conditions on both the cipher

and the attack parameters that should be fulfilled to maximize the efficiency of our parallel matching pairs sieving technique. First, the attack should filter pairs, in the sense that on average each candidate pair should suggest less than one key candidate. This can only happen in the case where the key bits involved are related to each other by many key schedule equations or when the cipher involves partial key addition layers as in `GIFT` or `SKINNY`. This is a necessary condition otherwise the key-recovery part would be more costly than the pairs generation process. The second condition is related to the number of candidate pairs. Obviously it should be higher than the data complexity, otherwise generating them would not be the bottleneck. But this is usually not enough to fully exploit the parallel matching algorithm. Indeed, both the terms $2^{l_1+rs-\sum_{i=1}^{r} p_i}$ and $2^{l_2+qs-\sum_{i=r+1}^{r+q} p_i}$ from the complexity formula of the algorithm should be lower than $2^{l_1+l_2}$ to expect any benefit from using the technique. Since s is always greater than p_i, both l_1 and l_2 should be sufficiently high to take into account as many filters as possible. Hence, the best situation would be a differential attack in which both the plaintext and the ciphertext have as many active bits as possible (to maximize $l_1 + l_2$) while both the state right after the first S-box layer and right before the last one have as many inactive bits as possible (to maximize the filters). In practice, differential attacks in this situation are quite rare since it would usually be much more efficient to arrange the inactive bits in order to minimize l_1 and l_2. But sometimes it is impossible and the recent differential attacks against both `PIPO` and `FLY` described in [KKK+25] look like good targets.

It is also important to mention that, whenever there are plenty of inactive bits after the first and before the last S-box layer, guessing key bits before starting the attack as suggested in [SLY+24] becomes very powerful as well. Actually this combines well with our technique since guessing key bits does increase the filters of related S-boxes and can therefore allow more granularity than guessing all the key bits involved in one S-box. Another interesting application of our advanced sieving could be to mount efficient data/time trade-off. An example on the blockcipher `SPEEDY` is proposed in Appendix B.

Overall, the conditions under which parallel matching sieving can outperform the best-known attacks appear to be restrictive, and in practice, the pair-generation process is not the main bottleneck in attacks aiming to minimize overall complexity. Nevertheless, we believe this improvement is valuable from a theoretical standpoint, demonstrating that a previously limiting step can be optimized, and relevant for the community, particularly for researchers developing automated attack search tools.

4 Improved single-key attacks on `GIFT-64`

In [CWWH25], Chang *et al.* reevaluated the probability of several differential distinguishers against both `GIFT-64` and `GIFT-128` through a fixed-key analysis based on the quasidifferential framework. From there, they proposed new differential attacks against both versions of `GIFT` and in particular were able to break up to 21 rounds of `GIFT-64` in the single-key setting for a weak-key space of 2^{124}.

Based on the same distinguisher, we were able to mount differential MitM attacks using a structure over two rounds, whilst only one round was added in previous works such as [AKM+24]. We therefore managed to improve the complexity of the 21-round attack and to provide for the first time an attack against 22 rounds in the single-key setting. We comprehensively describe the 22-round attack before explaining how the same ideas can be applied to improve the previous best attack on 21 rounds.

4.1 New attack on 22-round GIFT-64

We propose a new single-key differential MitM attack on GIFT-64 covering 22 rounds which is the best-known single-key attack against this cipher. Our attack is based on the 13-round differential distinguisher presented in [CZD20] :

$$(0000\ 0000\ 0000\ 0202) \xrightarrow[13r]{2^{-57.82}} (0000\ 0020\ 0000\ 0020).$$

In [CWWH25], the authors show that this distinguisher is not valid for all the keys but only in a weak key space W_1 of size 2^{124}. In W_1, they show that the probability of this distinguisher is $2^{-57.82}$. By considering 6 rounds before the distinguisher, the four conditions on the master key of W_1 are :

$$k_4[4] + k_4[12] = 0, k_4[0] + k_4[8] = 0 \quad (6)$$
$$k_0[5] + k_0[13] = 0, , \quad (7)$$
$$k_0[1] + k_0[9] = 0, \quad (8)$$

The differential MitM attack we propose involves 2 rounds of structure at the beginning of the attack, while, in the key-recovery, we prepended 4 rounds in the upper part and appended 3 rounds in the lower part. A detailed illustration of the attack is given in Figure 4 and Table 3 details the key elements that are guessed by each component of the attack.

Table 3: The key bits guessed during the 22-round attack for each side of the attack, the second column is the quantity of bits recovered by the attack for each sub-key.

Key	$\|k\|$	$\mathcal{K}_{in}$	$\mathcal{K}_{out}$	$\mathcal{K}_{in} \cap \mathcal{K}_{out}$
k_0	14	RK_0[9, 25], RK_4[13, 29]	RK_0[13, 15, 29, 31], RK_{20}[9, 11, 13, 15, 25, 27, 29, 31]	k_0[2, 10]
k_1	14	RK_0[4, 20], RK_4[12, 28]	RK_0[9, 22], RK_{20}[0, 2, 4, 6, 16, 18, 20, 22]	/
k_2	16	RK_1[1, 5, 19, 23]	RK_{21}[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31]	k_2[0, 2, 9, 11]
k_3	16	RK_1[10, 14, 24, 28]	RK_{21}[0, 2, 4, 6, 8, 10, 12, 14, 16, 20, 22, 24, 26, 28, 30]	k_3[5, 7, 12, 14]
k_4	16	RK_2[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31]	/	/
k_5	16	RK_2[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]	/	/
k_6	6	RK_3[17, 19, 21, 23]	RK_{19}[27, 31]	/
k_7	6	RK_3[16, 18, 20, 22]	RK_{19}[18, 22]	/
Σ	104	56	58	10

We now describe the different steps of our attack.

1. Fix the set $\mathcal{F}$ made of 56 bits of Z_0 and X_2 (gray bits in Figure 4).
2. Guess the 10 elements of $\mathcal{K}_{in} \cap \mathcal{K}_{out}$
3. *Upper part* : Generation of the $(X_2, \tilde{X}_2, i)_{\mathcal{F}}$ triplets.
   (a) Guess the $2^{56-10-2}$ elements of $\mathcal{K}_{in} \setminus (\mathcal{K}_{in} \cap \mathcal{K}_{out})$. The term 2^{-2} comes from the two equations in (6) of the weak key space W_1. Naively, we would have guessed all $k_4[0, 4, 8, 12]$ (corresponding to RK_2[1, 9, 17, 25] in Table 3) but we can only guess 2 of them and use the linear relations (6) to deduce the value of the remaining bits.

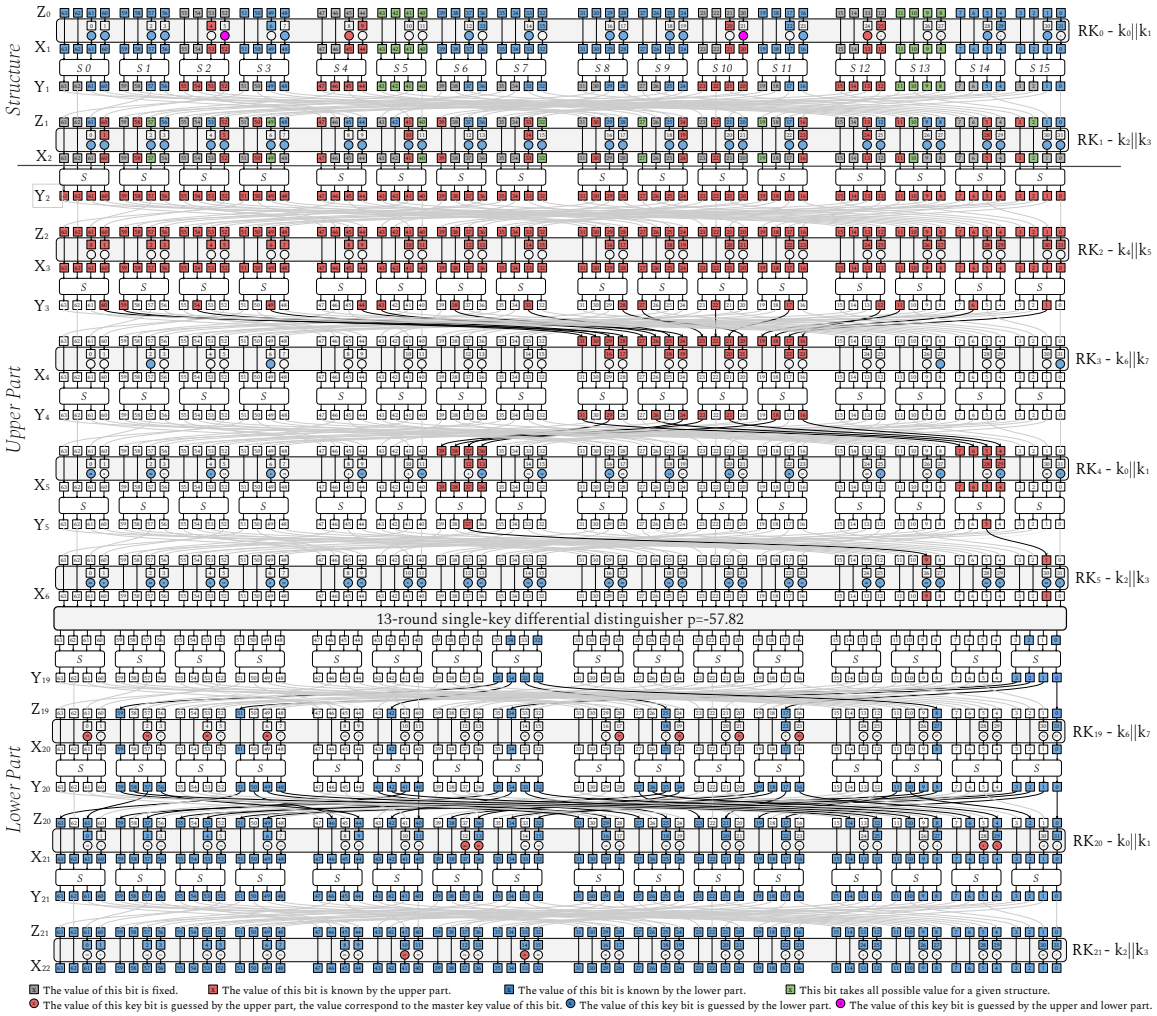

Figure 4: Illustration of the 22 rounds attack against GIFT-64

(b) For all the 2^{44} guessed values, compute 56 bits of X_2 from the 56 bits fixed in $\mathcal{F}$ and the known elements of $\mathcal{K}_{in}$.

(c) For all the possible values of the remaining 8 bits of X_2, compute all the triplets $(X_2, \tilde{X}_2, i)_{\mathcal{F}}$ such that after 4 rounds, X_2 and $\tilde{X}_2$ verify the constraints at the input of the distinguisher. Store all the triplets in a hash table of size 2^{8+44}.

4. *Lower part* : Generation of the $(Z_0, \tilde{Z}_0, j)_{\mathcal{F}}$ triplets

(a) Guess the $2^{58-10-1}$ elements of $\mathcal{K}_{out} \setminus (\mathcal{K}_{in} \cap \mathcal{K}_{out})$. Again, we can use (8) to guess only one bit of $k_0[1, 9]$ ($RK_{20}[19, 27]$ in Table 3) instead of two.

(b) For all the 2^{47} guessed values, compute 56 bits of Z_0 from the 56 bits fixed in $\mathcal{F}$ and the known elements of $\mathcal{K}_{out}$.

(c) For all the possible values of the remaining 8 bits of Z_0, request the encryption of $P = X_0$, obtaining $C = X_{22}$.

(d) Compute all the triplets $(X_{22}, \tilde{X}_{22}, j)_{\mathcal{F}}$ including the states after three rounds of forward propagation from the output of the distinguisher, such that, X_{22} and

$\tilde{X}_{22}$ verify the constraints at the output of the distinguisher for the value of $\mathcal{K}_{out} = j$. Request the decryption of $\tilde{X}_{22}$ and store all the triplets $(Z_0, \tilde{Z}_0, j)_{\mathcal{F}}$ in a hash table of size 2^{8+47}.

5. *Match* : Save in a hash table the pairs of triplets $\{(Z_0, \tilde{Z}_0, j), (X_2, \tilde{X}_2, i)\ \}_{\mathcal{F}}$, such that the 56 bits of $\tilde{X}_2$ and $\tilde{Z}_0$ involved in $\mathcal{F}$ are consistent with the values fixed in first step. We expect to filter 2^{56} possible pairs of triplets and thus to store a hash table of $2^{44+8+47+8-56=51}$ pairs of triplets for the considered set of values fixed in $\mathcal{F}$.

6. Repeat from the first step, by fixing the bits in $\mathcal{F}$ to a new distinct set of values. This is done $2^{57.82-8} = 2^{49.82}$ times to ensure that at least one pair (P, C) verifies the distinguisher.

The time complexity of the attack is :

$$\mathcal{T} = 2^{57.82-8}\ 2^{10}\Big(2^8 \cdot 2^{56-10-2} + 2^8 \cdot 2^{58-10-1} + 2^{8+44+8+47-56}\Big) \tag{9}$$

$$= 2^{111.82} + 2^{114.82} + 2^{110.82} \approx 2^{115.07} \tag{10}$$

The memory complexity of the attack :

$$\mathcal{M} = 2^{max(52,\ 55,\ 49)} = 2^{55} \tag{11}$$

The attack as described above requires the whole codebook, i.e a data complexity of 2^{64}. However, we can use the trick presented in [BDD+23] to find a new data/memory trade-off and force the messages to belong in some restricted portion of the codebook. The idea is to set a bits in the plaintexts to be constant (amongst the 16 bits in $\mathcal{F}$) and the constraint on a is that $57.82 + a$ should be lower than $64 - a$ to expect at least one pair to follow the differential and to respect the a conditions also on $\tilde{P}$. This thus leads to $a = 3$ in our case and a data complexity of 2^{61}. The memory complexity is also increased to 2^{61}, as we have to precompute and store this number of pairs (plaintext, ciphertext) verifying the a conditions, to know beforehand which ones will verify the a conditions and not having to decrypt a higher number.

$$\mathcal{D} = 2^{61} \tag{12}$$

Recover the full key. After the attack, 24 bits of the master key still haven't been recovered: 10 bits of k_7 (0, 2, 4, 5, 6, 7, 12, 13, 14, 15), 10 bits of k_6 (0, 1, 2, 3, 4, 5, 6, 7, 12, 14), 2 bits of k_1 (5 and 13) and 2 bits of k_0 (5, 13). To recover the full master key, the attacker can consider the values of the bits and differences around the S-boxes 5 and 13 (green values in Figure 4). The attacker then computes through the information contained within each pair of triplets, the values of the four bits at the output of the S-boxes, the values of the two left bits at the input and the differences of the two right bits of the S-boxes. In addition, the attacker can consider the equation (7) of the weak-key space between $k_0[5]$ and $k_0[13]$ and add one bit of information. By doing so, the attacker will filter $2^{6+6+1} = 2^{13}$ pairs of triplets and recover the value of the missing key bits of both k_0 and k_1. We now have $2^{110.82-13} = 2^{97.82}$ remaining triplets. With them, we can find the remaining 20 missing key bits of k_6 and k_7 by testing the pairs on extra an extra data (or on one of the pairs not verifying the differential with no extra data cost). The final time complexity will be $2^{97.82+20} = 2^{117.82}$. Note that we could have repeated the previous process one more time, reducing the number of recovered triplets appearing in both cases by 2^{-7}, with a slightly better time complexity of $2^{116.38}$, but with a much bigger memory complexity.

Remark. It is still unclear why the differential MitM cryptanalysis technique sometimes gives much better results than more classical differential attacks. In the original paper [BDD+23], authors claim that a condition is that the key size should be much higher than the state size. We agree with them on this point, as for instance we were unable to improve the results on `GIFT-128` through differential MitM attacks. However, it looks like the use of internal structures, which probably is the most powerful advantage of the technique, allows delaying the *guess* of the key material involved in the structure at the end of the attack. Because of the partial key addition in `GIFT`, these key bits are actually not guessed but deduced from the matching pairs since the entropy is low. Hence, we believe that an extra condition for the differential MitM technique to be efficient is that the key bits involved in the key recovery part are highly related to each other through the key schedule relations or that the cipher only involves partial key additions.

4.2 Improved 21-round attack on `GIFT-64`

We now propose a new single-key differential Meet-in-the-Middle attack on `GIFT-64` with better time and memory complexities than the best known attack against 21 rounds of this cipher. We first tried to extend the previous distinguisher over 13 rounds. Unfortunately, this did not work as considering only one round structure is not sufficient to determine enough key material through the attack and the final exhaustive search becomes too costly. Our attack is thus based on the same distinguisher as the attack on 22 rounds, without the last round :

$$0000\ 0000\ 0000\ 0202 \longrightarrow_{12r} 0000\ 0020\ 0000\ 0020.$$

Regarding the key space of the differential on 12 rounds, we found that the incompatibilities that reduce the key space to W_1 do not involve the last round-key. Thus, the differential on 12 rounds has the same weak key space W_1 as the previous 22-round attack and a probability of $2^{-53.82}$, as the last round had probability 2^{-4}.

The attack we mounted is described in Figure 5. As in the previously described attack, we consider 2 rounds of structure at the beginning of the attack, 4 rounds for the upper part and 3 rounds for the lower part. The key bits guessed during the attack are given in Table 4.

Table 4: The key bits guessed during the 21 rounds attack by each side of the attack, the second column is the quantity of bits recovered by the attack for each sub key.

Key	$\|k\|$	$\mathcal{K}_{in}$	$\mathcal{K}_{out}$	$\mathcal{K}_{in} \cap \mathcal{K}_{out}$
k_0	16	$RK_4[13, 29]$	$RK_{20}[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31]$	$k_0[2, 10]$
k_1	16	$RK_4[12, 28]$	$RK_{20}[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]$	$k_1[4, 12]$
k_2	3	$RK_1[3, 7, 31]$	/	/
k_3	3	$RK_1[30]$, $RK_5[26, 30]$	/	/
k_4	16	$RK_2[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31]$	$RK_{18}[3, 7]$	$k_4[1, 3]$
k_5	16	$RK_2[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]$	$RK_{18}[26, 30]$	$k_5[13, 15]$
k_6	12	$RK_3[17, 19, 21, 23]$	$RK_{19}[9, 11, 13, 15, 25 ,27, 29, 31]$	/
k_7	8	$RK_3[16, 18, 20, 22]$	$RK_{19}[0, 2, 4, 6, 16, 18, 20, 22]$	$k_7[8, 9, 10, 11]$
Σ	90	50	52	12

The framework of the attack is similar to the one used in the attack on 22 rounds previously presented. The equations in (7) and (8) are used to sieve 2^{-2} key guesses in

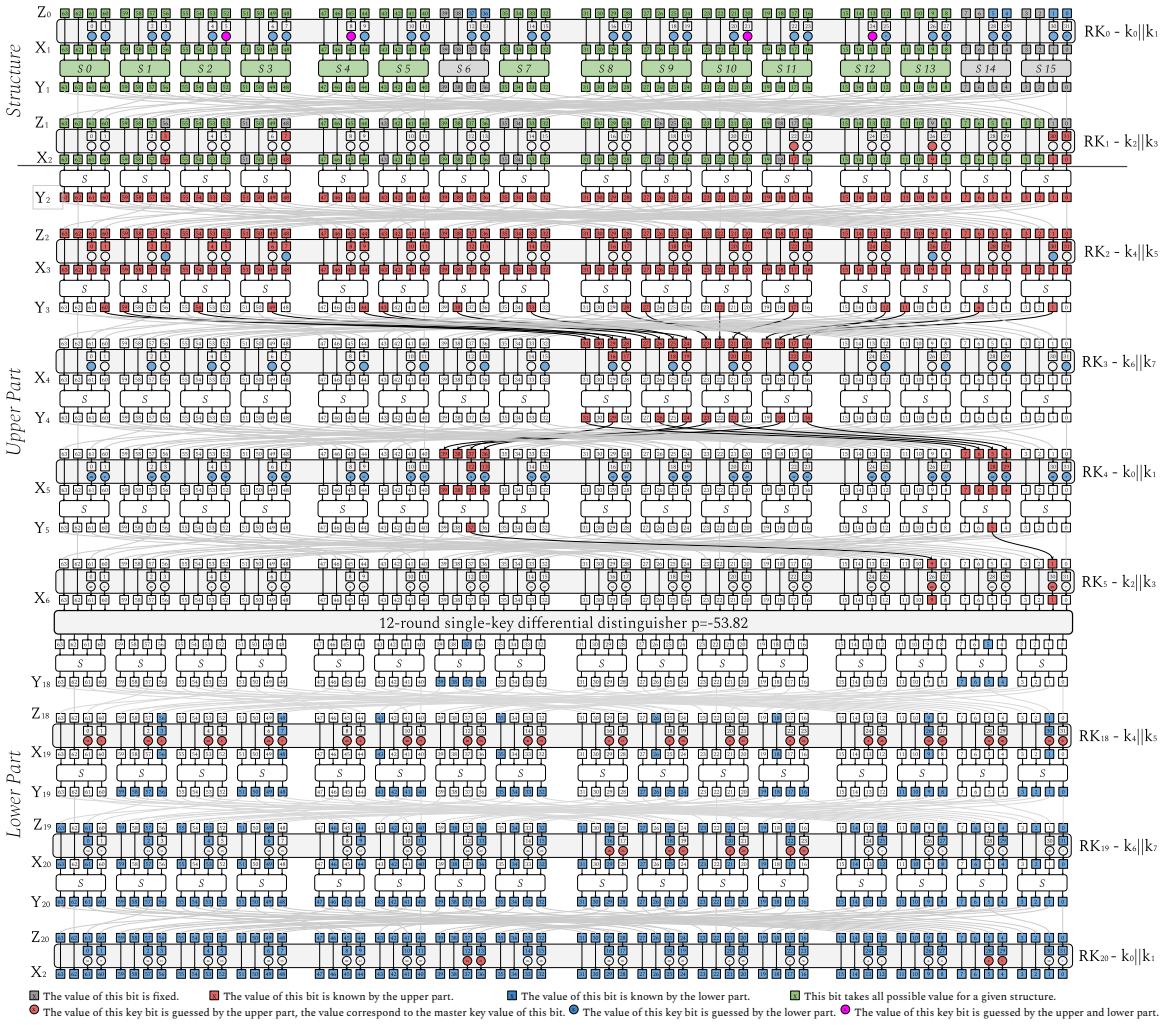

Figure 5: Path of the 21 rounds attack against `GIFT-64`

the lower part ($k_0[1, 5, 9, 13]$ are guessed by the lower part as $RK_{20}[3, 11, 19, 27]$). The two equations in (6) are used to sieve 2^{-2} key guesses in the upper part ($k_4[0, 4, 8, 12]$ are guessed by the upper part as $RK_2[0, 4, 8, 12]$). The structure used is composed of 12 fixed bits (grey bits in Figure 5) and 52 unfixed bits (green bits in Figure 5). The main difference is that, during the match step, the attacker can save in a hash table the pairs of triplets such that:

1. The 12 bits of $\tilde{X}_2$ and $\tilde{Z}_0$ involved in the fixed part of the structure are consistent. We expect to filter 2^{-12} possible pairs of triplets.

2. By considering the 13 non-fixed S-boxes in the structure (green S-boxes in figure 5) the attacker can compute, with the elements of the lower hash table $(Z_0, \tilde{Z}_0, j)_{\mathcal{F}}$, the output of the S-boxes and match with the upper triplets $(X_2, \tilde{X}_2, i)_{\mathcal{F}}$ on :

   (a) The values of the 2 left bits of each S-box for Z_0 and X_2 and for $\tilde{Z}_0$ and $\tilde{X}_2$. We expect a $2^{-2\times2\times13}$ filtering effect.

(b) The value of the differences $Z_0 \oplus \tilde{Z}_0$ and $X_2 \oplus \tilde{X}_2$ on the 2 right bits of each S-box. We expect a $2^{-2\times 13}$ filtering effect. During this filtering step, the attacker can also retrieve the value of the 26 key bits involved in the differences. If a pair of triplets is not discarded during this step, the attacker adds to the pairs of triplets the values of the key bits $k_3[0,1,2,3,4,5,6,7,8,9,10,12,14]$ and $k_2[0,2,4,5,6,7,8,9,10,11,12,13,14]$.

The time complexity of the attack is:

$$\mathcal{T} = 2^{53.82-52}\ 2^{12}(2^{52}2^{38-2} + 2^{52}2^{40-2} + 2^{88+90-12-52-26}) \tag{13}$$

$$= 2^{101.82} + 2^{103.82} + 2^{101.82} \approx 2^{104.32} \tag{14}$$

The memory complexity of the attack:

$$\mathcal{M} = 2^{max(88,\ 90,\ 88)} = 2^{90} \tag{15}$$

By applying the same method as the one described in the attack on 22 rounds:

$$\mathcal{D} = 2^{59} \tag{16}$$

Recover the full key. After the attack, $90 + 26 = 116$ bits of the master key are known by the attacker, and for them, we have $2^{101.82}$ triplet candidates. The attacker could recover the 12 last key elements by testing the $2^{101.82+12} = 2^{113.82}$ possibilities on an extra pair of data, making this the time bottleneck. Instead, it might be more interesting to repeat the procedure and store the candidate triplets in memory, filtering therefore by a factor of $2^{101.82-116} = 2^{-14.82}$. We recover then $2^{101.82-14.82} = 2^{87}$ candidate triplets, and the complexity of testing the final 12 bits of unknown key becomes 2^{99}. The time complexity bottleneck comes from repeating the previous procedure twice, becoming $2 \times 2^{104.32}$, and the memory is increased to store the first set of potential solutions of size $2^{101.82}$.

5 Conclusion

In this paper, we have proposed new improved attacks against the block cipher `GIFT-64`. We have increased the number of attacked rounds in the single-key setting, and improved other reduced-round attacks. For this, we have considered the use of non-linear filtering techniques, and in particular built more complex initial structures when mounting differential MitM attacks.

We also show that differential MitM can be the most efficient attack on `GIFT-64`. In particular, because of the already mentioned condition of having a key size bigger than the state as a good candidate for target of these attacks, we do not expect `GIFT-128` to have the best known attacks given by differential MitM ones, as after considering an interesting distinguisher, the margin for dealing with the key-recovery part would be too small.

Acknowledgements. This work has been supported by the French Agence Nationale de la Recherche through the OREO project under Contract ANR-22-CE39-0015, and through the France 2030 program under grant agreement No. ANR-22-PECY-0010 CRYPTANALYSE. It has been partially funded by the European Union (ERC-2023-COG, SoBaSyC). 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

[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, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 280–309. Springer, Cham, May 2024.

[BCF+21] Marek Broll, Federico Canale, Antonio Flórez-Gutiérrez, Gregor Leander, and María Naya-Plasencia. Generic framework for key-guessing improvements. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 453–483. Springer, 2021.

[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, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 240–272. Springer, Cham, August 2023.

[BDD+24] Christina Boura, Nicolas David, Patrick Derbez, Rachelle Heim Boissier, and María Naya-Plasencia. A generic algorithm for efficient key recovery in differential attacks - and its associated tool. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 217–248. Springer, Cham, May 2024.

[BDG25a] Christina Boura, Patrick Derbez, and Baptiste Germon. Extending the quasidifferential framework: From fixed-key to expected differential probability. *IACR Trans. Symmetric Cryptol.*, 2025(1):515–541, 2025.

[BDG+25b] Christina Boura, Patrick Derbez, Baptiste Germon, Rachelle Heim Boissier, and María Naya-Plasencia. A note on a recent attack against SPEEDY-7-192. Cryptology ePrint Archive, Paper 2025/1265, 2025. Note SPEEDY.

[BDG+25c] Christina Boura, Patrick Derbez, Baptiste Germon, Rachelle Heim Boissier, and María Naya-Plasencia. SPEEDY: Caught at last. Cryptology ePrint Archive, Paper 2025/890, 2025. Old version.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, Cham, September 2017.

[BR22] Tim Beyne and Vincent Rijmen. Differential cryptanalysis in the fixed-key model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 687–716. Springer, Cham, August 2022.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 2–21. Springer, Berlin, Heidelberg, August 1991.

[BS93] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 487–496. Springer, Berlin, Heidelberg, August 1993.

[CHLW25] Chengcheng Chang, Kai Hu, Muzhou Li, and Meiqin Wang. Related-key differential and boomerang cryptanalysis in the fixed-key model. *IACR Cryptol. ePrint Arch.*, page 402, 2025.

[CN25] Federico Canale and María Naya-Plasencia. Guessing less and better: improved attacks on GIFT-64. *Des. Codes Cryptogr.*, 93(3):787–822, 2025.

[CWWH25] Chengcheng Chang, Meiqin Wang, Wei Wang, and Kai Hu. Quasidifferential saves infeasible differential - improved weak-key key-recovery attacks on round-reduced GIFT. In Arpita Patra, editor, *Topics in Cryptology - CT-RSA 2025 - Cryptographers' Track at the RSA Conference 2025, San Francisco, CA, USA, April 28-May 1, 2025, Proceedings*, volume 15598 of *Lecture Notes in Computer Science*, pages 27–50. Springer, 2025.

[CZD20] Huaifeng Chen, Rui Zong, and Xiaoyang Dong. Improved differential attacks on gift-64. In Jianying Zhou, Xiapu Luo, Qingni Shen, and Zhen Xu, editors, *Information and Communications Security*, pages 447–462, Cham, 2020. Springer International Publishing.

[FJP13] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 183–203. Springer, Berlin, Heidelberg, August 2013.

[JZZD21] Fulei Ji, Wentao Zhang, Chunning Zhou, and Tianyou Ding. Improved (related-key) differential cryptanalysis on gift. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography*, pages 198–228, Cham, 2021. Springer International Publishing.

[KKK+25] Insung Kim, Seonggyeom Kim, Sunyeop Kim, Donggeun Kwon, Hanbeom Shin, Dongjae Lee, Deukjo Hong, Jaechul Sung, and Seokhie Hong. Towards optimal differential attacks on FLY and PIPO. Cryptology ePrint Archive, Paper 2025/837, 2025.

[Knu95] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *FSE'94*, volume 1008 of *LNCS*, pages 196–211. Springer, Berlin, Heidelberg, December 1995.

[LKKD08] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. Improving the efficiency of impossible differential cryptanalysis of reduced Camellia and MISTY1. In Tal Malkin, editor, *CT-RSA 2008*, volume 4964 of *LNCS*, pages 370–386. Springer, Berlin, Heidelberg, April 2008.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. *IACR TCHES*, 2021(4):510–545, 2021.

[LS19] Yunwen Liu and Yu Sasaki. Related-key boomerang attacks on gift with automated trail search including bct effect. In Julian Jang-Jaccard and Fuchun Guo, editors, *Information Security and Privacy*, pages 555–572, Cham, 2019. Springer International Publishing.

[Nay11] María Naya-Plasencia. How to improve rebound attacks. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 188–205. Springer, Berlin, Heidelberg, August 2011.

[QHS16] Kexin Qiao, Lei Hu, and Siwei Sun. Differential security evaluation of simeck with dynamic key-guessing techniques. In Olivier Camp, Steven Furnell, and Paolo Mori, editors, *Proceedings of the 2nd International Conference on Information Systems Security and Privacy, ICISSP 2016, Rome, Italy, February 19-21, 2016*, pages 74–84. SciTePress, 2016.

[RGMS22] Loïc Rouquette, David Gérault, Marine Minier, and Christine Solnon. And rijndael?: Automatic related-key differential analysis of rijndael. In Lejla Batina and Joan Daemen, editors, *AFRICACRYPT 22*, volume 2022 of *LNCS*, pages 150–175. Springer, Cham, July 2022.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, Berlin, Heidelberg, December 2014.

[SLY+24] Ling Song, Huimin Liu, Qianqian Yang, Yincen Chen, Lei Hu, and Jian Weng. Generic differential key recovery attacks and beyond. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VII*, volume 15490 of *LNCS*, pages 361–391. Springer, Singapore, December 2024.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Trans. Symm. Cryptol.*, 2021(1):269–315, 2021.

[SWW22] Ling Sun, Wei Wang, and Meiqin Wang. Improved attacks on gift-64. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography*, pages 246–265, Cham, 2022. Springer International Publishing.

[SYL23] Ling Song, Qianqian Yang, and Huimin Liu. Revisiting the differential meet-in-the-middle cryptanalysis. Cryptology ePrint Archive, Report 2023/1302, 2023.

[WLHL24] Shichang Wang, Meicheng Liu, Shiqi Hou, and Dongdai Lin. Differential-linear cryptanalysis of GIFT family and GIFT-based ciphers. *IACR Communications in Cryptology*, 1(1), 2024.

[WWJZ18] Ning Wang, Xiaoyun Wang, Keting Jia, and Jingyuan Zhao. Differential attacks on reduced SIMON versions with dynamic key-guessing techniques. *Sci. China Inf. Sci.*, 61(9):098103:1–098103:3, 2018.

[ZDY19] Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. Milp-based differential attack on round-reduced gift. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 372–390, Cham, 2019. Springer International Publishing.

A GIFT-64

We give the specifications of the S-box (Table 6) and of the bit-permutation (Table 5) used in GIFT-64. We also provide the Difference Distribution Table (DDT) of the S-box in Table 7.

Table 5: The bit-permutation of GIFT-64

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	17	34	51	48	1	18	35	32	49	2	19	16	33	50	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

Table 6: The S-box of GIFT

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	1	10	4	12	6	15	3	9	2	13	11	7	5	0	8	14

Table 7: The Difference Distribution Table (DDT) of the GIFT S-box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	2	2	0	2	2	2	2	2	0	0	2
2	0	0	0	0	0	4	4	0	0	2	2	0	0	2	2	0
3	0	0	0	0	0	2	2	0	2	0	0	2	2	2	2	2
4	0	0	0	2	0	4	0	6	0	2	0	0	0	2	0	0
5	0	0	2	0	0	2	0	0	2	0	0	0	2	2	2	4
6	0	0	4	6	0	0	0	2	0	0	2	0	0	0	2	0
7	0	0	2	0	0	2	0	0	2	2	2	4	2	0	0	0
8	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4
9	0	2	0	2	0	0	2	2	2	0	2	0	2	2	0	0
a	0	4	0	0	0	0	4	0	0	2	2	0	0	2	2	0
b	0	2	0	2	0	0	2	2	2	2	0	0	2	0	2	0
c	0	0	4	0	4	0	0	0	2	0	2	0	2	0	2	0
d	0	2	2	0	4	0	0	0	0	0	2	2	0	2	0	2
e	0	4	0	0	4	0	0	0	2	2	0	0	2	2	0	0
f	0	2	2	0	4	0	0	0	0	2	0	2	0	0	2	2

B Application to SPEEDY

In this section we describe new data/time trade-offs of for the recent attacks proposed by Boura *et al.* in both [BDG+25c] and [BDG+25b] against the block cipher SPEEDY introduced by Leander *et al.* in 2021 [LMMR21].

SPEEDY is an ultra-low-latency block cipher family with instances denoted SPEEDY, where r is the round count and n (a multiple of 6) the block size. The main variants—SPEEDY, SPEEDY, and SPEEDY—use 192-bit keys and blocks, differing only in rounds and claimed security.

The 192-bit state is arranged as a 32×6 bit matrix $x[i][j]$ with indices $0 \leq i < 32$, $0 \leq j < 6$; bit 0 is most significant, with arithmetic modulo 6 (rows) and 32 (columns).

Each round applies AddRoundKey, SubBox, ShiftColumns, a second SubBox then ShiftColumns and MixColumns, and finally AddRoundConstant. The plaintext initializes the state, and R_r is iterated $r \in \{5, 6, 7\}$ times.

Boura *et al.* proposed new attacks against the different versions of the SPEEDY family of block ciphers. They searched for the attacks with the best overall complexities and were in particular able to provide the first valid attack against the full version of SPEEDY-7-192. By applying our new pairs generation process based on parallel matching, we can propose similar attacks with lower data complexities and a contained increase of the time complexity. We take their attack against the full version of SPEEDY-7 to illustrate our technique. The attack is depicted in Figure 6, the purple bits are inactive in the original attack but active in our.

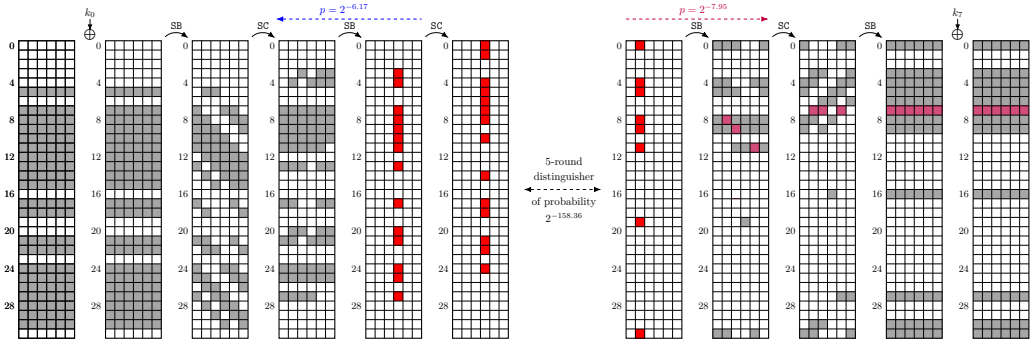

Figure 6: Attack against SPEEDY-7-192.

The probability p of the distinguisher together with the probabilistic extension is $2^{-6.17-158.36-7.95} = 2^{-172.48}$ against $2^{-173.53}$ if the purple bits are inactive as in the original attack. We thus need only $2^{173.48}$ messages to generate enough pairs so one follows both the probabilistic extension and the differential distinguisher. Because of the extra active bits, during the key-recovery we now have to deal with $2^{172.48+6} = 2^{178.48}$ pairs if they are generated without taking care of the non-linear filter coming from S-boxes of the first and last layers, and we verified with the KyRyDi tool that this is the bottleneck of the attack. We can use the parallel matching algorithm to exploit the filter of $2^{-1.54}$ on row 16 on the ciphertext side. Given one structure of $2^{21\times 6} = 2^{126}$ plaintexts, we can form $2^{126-20\times 6} = 2^{120}$ groups of 2^6 messages according to their value in the 120 inactive bits of the ciphertext. Instead of generating the $2^{6+5} = 2^{11}$ pairs for each of them, we apply the parallel matching algorithm to generate the pairs under the constraint of the filter on row 16. This gives a time complexity of $2^{6+5-1.54} + 2^{6-1-1.54} + 2^6 = 2^{9.61}$ to generate the $2^{9.46}$ possible pairs. As a consequence, the time complexity is only $2^{172.48-126} \times 2^{120+9.61} = 2^{176.09}$ which is more efficient than the classical approach.

The same technique can be directly applied to their attack against 4 rounds of SPEEDY-5 since it already has only 31 inactive rows in total and the data generation process is the bottleneck. However, the maximum filter is on row 21 on the ciphertext side and is only of $2^{-0.79}$. As a consequence the time complexity is locally of $2^{6+5-0.79}+2^{6-1-0.79}+2^6 = 2^{10.31}$ instead of 2^{11}, lowering the final time complexity of the attack from $2^{66.18}$ to $2^{65.49}$.

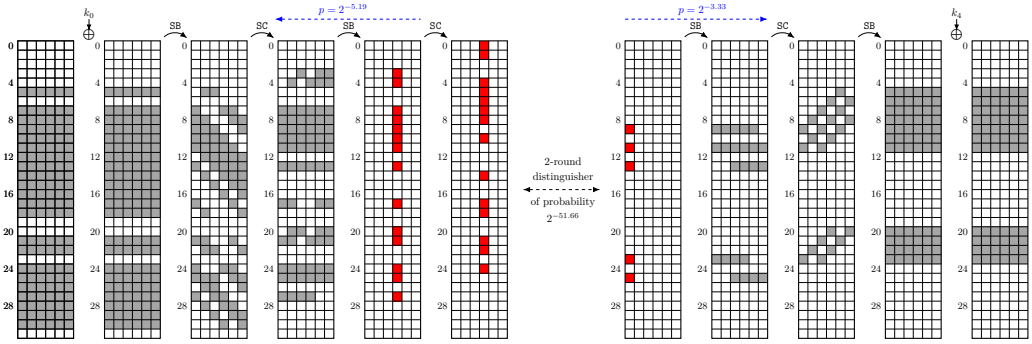

Figure 7: Attack against 4-round `SPEEDY-5-192` [BDG+25c].

After investigating the blockcipher `SPEEDY` and the different attacks proposed by Boura *et al.* it appears that the probabilistic extension is very cheap, in the sense that increasing by one the number of active rows only decrease by a factor 2 the data complexity. Furthermore the maximum filter per S-box is very low, which does not allow to exploit much of them with the parallel matching algorithm. It clearly appears that blockciphers with small S-boxes and partial key addition as in `GIFT` offer much more filter and are thus better candidates for this new pair-generation process.