New Attacks on Feistel Structures with Improved Memory Complexities

Itai Dinur[1], Orr Dunkelman[2,4,⋆],
Nathan Keller[3,4,⋆⋆], and Adi Shamir[4]

[1] Département d'Informatique, École Normale Supérieure, Paris, France
[2] Computer Science Department, University of Haifa, Israel
[3] Department of Mathematics, Bar-Ilan University, Israel
[4] Computer Science department, The Weizmann Institute, Rehovot, Israel

Abstract. Feistel structures are an extremely important and extensively researched type of cryptographic schemes. In this paper we describe improved attacks on Feistel structures with more than 4 rounds. We achieve this by a new attack that combines the main benefits of meet-in-the-middle attacks (which can reduce the time complexity by comparing only half blocks in the middle) and dissection attacks (which can reduce the memory complexity but have to guess full blocks in the middle in order to perform independent attacks above and below it). For example, for a 7-round Feistel structure on n-bit inputs with seven independent round keys of $n/2$ bits each, a MITM attack can use ($2^{1.5n}$, $2^{1.5n}$) time and memory, while dissection requires (2^{2n}, 2^{n}) time and memory. Our new attack requires only ($2^{1.5n}$, 2^{n}) time and memory, using a few known plaintext/ciphertext pairs. When we are allowed to use more known plaintexts, we develop new techniques which rely on the existence of multicollisions and differential properties deep in the structure in order to further reduce the memory complexity.

Our new attacks are not just theoretical generic constructions — in fact, we can use them to improve the best known attacks on several concrete cryptosystems such as CAST-128 (where we reduce the memory complexity from 2^{111} to 2^{64}) and DEAL-256 (where we reduce the memory complexity from 2^{200} to 2^{144}), without affecting their time and data complexities. An extension of our techniques applies even to some non-Feistel structures — for example, in the case of FOX, we reduce the memory complexity of all the best known attacks by a factor of 2^{16}.

Keywords: Cryptanalysis, block cipher, Feistel structure, dissection, meet-in-the-middle, splice-and-cut, CAST-128, DEAL.

⋆ The second author was supported in part by the Israeli Science Foundation through grant No. 827/12.
⋆⋆ The third author was supported by the Alon Fellowship.

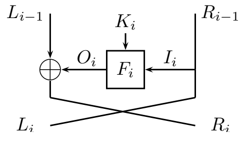


Fig. 1. The i'th Round of a Feistel Structure

1 Introduction

Feistel structures were first used in the design of DES [17], and had a major influence on the development of both the theory and the practice of cryptography (e.g., in the Luby-Rackoff [16] construction of pseudo random permutations and in the design of numerous block cipher proposals). In this paper we will primarily consider generic Feistel structures whose i-th round is depicted in Figure 1. They divide their n-bit blocks into two equal parts (L_i, R_i), use independent $n/2$-bit subkeys in their ℓ rounds, and have round functions F_i over $n/2$-bit inputs, outputs and subkeys which are perfect in the sense that they cannot be broken with attacks that are faster than exhaustive search. This choice of parameters allows us to consider any two consecutive rounds in a Feistel structure as a single round in a regular (non-Feistel) structure that has n-bit inputs outputs and subkeys. However, when we describe attacks on concrete schemes which have a Feistel structure, we will consider the relevant key and block sizes, and exploit some of the weaknesses of the actual round functions.

A major type of low-data attacks which can be applied to multi-round constructions is the Meet-In-The-Middle (abbreviated as MITM) attack, which was proposed by Diffie and Hellman [8] in 1977 as a method for cryptanalyzing double encryption schemes. It gained additional fame in 1985, when Chaum and Evertse [7] applied it to reduced-round variants of DES [17], and it is now considered as an essential part in any course in cryptanalysis. In the last few years, research of MITM techniques had expanded in diverse directions and numerous new extensions of the basic MITM appeared, including partial matching [4], probabilistic matching [14, 20], bicliques [3], sieve-in-the-middle [6], and many others. A more recent approach is the dissection attack, which was introduced by Dinur et al. [9] at CRYPTO 2012. Dissection can solve a wide variety of combinatorial search problems with improved combinations of time and memory complexities. In its cryptanalytic application, dissection significantly improved the time/memory tradeoff achievable by MITM attacks on multiple encryption schemes with more than 3 rounds.

The main difference between these two types of low-data attacks can be described in the following way: In basic MITM the adversary starts from the known plaintexts and ciphertexts at the endpoints, and works from both endpoints towards the middle by guessing some keys and building appropriate lookup tables. The equality of the pairs of values in the middle is used just as a filtering condition to identify the correct keys, and there is no need to know them in order to start the attack. In dissection attacks the adversary starts by guessing the relevant values in the middle, and works from the middle towards the endpoints. In fact, the knowledge of the middle values enables the adversary to break the cryptanalytic problem into two independent smaller problems with new known plaintext and ciphertext pairs at their endpoints, which can be solved recursively by another dissection, or with MITM at the leaves of the recursion tree.

When we compare the two types of attacks on a Feistel structure with an odd number of $\ell = 2r+1$ rounds, we notice that each one of them offers different advantages and disadvantages. The MITM attack can ignore the middle round by comparing only the $n/2$-bit half blocks which are not affected by this round in the Feistel structure. This enables the MITM attack to be more time-efficient in the case of Feistel structures, since it does not have to guess the middle subkey. Even though dissection is naturally more efficient than MITM, it loses in its time complexity in this case since it has to guess the full n-bit middle value in order to be able to encrypt and decrypt this guessed value through multiple rounds.

In this paper we present new techniques which enable us to combine the MITM and dissection approaches, along with additional ingredients, such as iterating over values that are not used later in the MITM attack, and using multi-collisions and differential properties in the middle of the Feistel structure. We first consider the case of Feistel structures with an odd number of rounds, and try to reduce the memory complexity of the most time-efficient attacks on them. We show that the memory complexity of the MITM attack on $\ell = 2r+1$ rounds for $r \geq 3$ can be reduced all the way from $2^{0.5rn}$ to about $2^{\lceil r/2 \rceil 0.5n}$ (like in dissection) without increasing the time complexity, at the expense of increasing the data complexity to about $2^{\lceil \frac{r-3}{(r+1)} \rceil 0.5n}$ known plaintexts. If no additional plaintexts are allowed, we are still able to reduce the memory, but only to about $2^{\lceil 2r/3 \rceil 0.5n}$. In particular, we can reduce the memory complexity of the standard MITM attack on 7-round Feistel from $2^{1.5n}$ to 2^n with no effect on the data and time complexities.[1]

A different goal is to reduce the time complexity of the most memory-efficient nontrivial attacks (in which the memory available to the adversary is restricted to $2^{0.5n}$, which makes it possible to store in memory all the possible values of a single half-block or a single subkey, but not more). Here, we assume that the round functions can be inverted efficiently when their subkey is given. In [9], Dinur et al. considered low-memory dissection attacks on general (non-Feistel) structures, in which the available memory is restricted to 2^n (where n is the length of a single block or a single subkey). They defined the *gain* of a dissection attack of time complexity T over a standard MITM attack with 2^n memory (whose time complexity is $2^{(\ell-1)n}$ for ℓ rounds) by $(\ell-1) - \log_2(T)/n$. Then, they computed the sequence of round numbers ℓ for which the gain of the best dissection attack increases by one — {4,7,11,16,22,29,37,...}. We use our techniques to compute a similar sequence of round numbers ℓ of Feistel structures for which the gain

[1] We alert the reader that similarly to [9], we concentrate on the asymptotic complexity and ignore small logarithmic factors in r and n.

(over MITM, whose time complexity is $2^{(\ell-2)0.5n}$) increases by one — it turns out to be $\{5,10,15,22,29,38,47,\ldots\}$, and the asymptotic complexity of an attack on ℓ-round Feistel using a minimal amount of $2^{0.5n}$ memory turns out to be $2^{0.5n(\ell-2-\sqrt{2\ell}+o(\sqrt{\ell}))}$. In particular, we present an attack on 5-round Feistel with time complexity of 2^n and memory complexity of $2^{0.5n}$ (compared to $(2^n, 2^n)$ or $(2^{1.5n}, 2^{0.5n})$ which are the best that can be obtained by previous attacks).

To deal with an even number of rounds without having to guess the extra key, we show that our algorithms can be combined with a recent algorithm presented by Isobe and Shibutani [11] at Asiacrypt 2013. This algorithm extends MITM attacks on Feistel structures by one round, at the expense of increasing the time complexity by $2^{0.25n}$ and using $2^{0.25n}$ chosen plaintexts. As a result, we obtain an attack on a Feistel structures with $\ell = 2r$ rounds (for $r \geq 4$) that requires $2^{(0.5r-0.25)n}$ time, about $2^{(0.5\lceil r/2\rceil-0.25)n}$ memory, and $\max\{2^{0.25n}, 2^{\lceil\frac{r-4}{r}\rceil 0.5n}\}$ chosen plaintexts. Alternatively, we can use only $2^{0.25n}$ chosen plaintexts like in [11], with $2^{(0.5r-0.25)n}$ time and about $2^{(\lceil 2(r-1)/3\rceil 0.5+0.25)n}$ memory. In particular, we reduce the memory complexity of Isobe and Shibutani's attack [11] on 8-round Feistel structures from $2^{1.75n}$ to $2^{1.25n}$, with no effect on the data and time complexities.

In Table 1 we compare the complexity of our attacks with previous results for certain numbers of rounds which have "clean" exponents.

While all the techniques described so far are completely generic, they allow us to significantly improve the best known attacks on several concrete block ciphers. In particular, we reduce the memory complexity of the best known attack on 8-round CAST-128 [1] from 2^{111} to 2^{64}, and the memory complexity of the best known attack on 8-round DEAL [15] with 256-bit keys from 2^{200} to 2^{144}, both without affecting the time and data complexities. It is interesting to note that an extension of our techniques can even be applied to certain non-Feistel cryptosystems, such as FOX [13], in which the best MITM attack uses the *partial matching* technique.

This paper is organized as follows: In Section 2 we describe the improved memory complexities we obtain when we consider the most time-efficient attacks, and in Section 3 we describe the improved time complexities which can be obtained when we consider the most memory-efficient attacks. In Section 4 we sketch the application of our results to concrete block ciphers. We conclude the paper in Section 5.

2 Improving the Memory Complexity of the most Time-Efficient Attacks on Feistel Structures

Consider a standard Feistel structure with an odd number $\ell = 2r + 1$ of rounds. The generic dissection attack on this construction (that does not exploit the Feis-

[2] In the case of 5-round Feistel, the dissection attack is not better than the meet in the middle attack.

Rounds	Complexity			Attack
	Time	Memory	Data	
5	2^{n}	2^{n}	3 KP	Meet in the Middle[2]
	$2^{1.5n}$	$2^{0.5n}$	3 KP	Meet in the Middle
	2^{n}	$2^{0.5n}$	3 KP	**New** (Section 3.1)
7	$2^{1.5n}$	$2^{1.5n}$	4 KP	Meet in the Middle
	2^{2n}	2^{n}	4 KP	Dissection
	$2^{1.5n}$	2^{n}	4 KP	**New** (Section 2.1)
8	2^{2n}	$2^{1.5n}$	4 KP	Meet in the Middle
	2^{2n}	2^{n}	4 KP	Dissection
	$2^{1.75n}$	$2^{1.75n}$	$2^{0.25n}$ CP	Splice-and-cut [11]
	$2^{1.75n}$	$2^{1.25n}$	$2^{0.25n}$ CP	**New** (Section 2.3)
15	$2^{3.5n}$	$2^{3.5n}$	8 KP	Meet in the Middle
	$2^{6.5n}$	$2^{0.5n}$	8 KP	Meet in the Middle
	2^{4n}	2^{2n}	8 KP	Dissection
	2^{5n}	$2^{0.5n}$	8 KP	**New** (Section 3.2)
	$2^{3.5n}$	2^{2n}	$2^{0.25n}$ KP	**New** (Section 2.2)
31	$2^{7.5n}$	$2^{7.5n}$	16 KP	Meet in the Middle
	$2^{14.5n}$	$2^{0.5n}$	16 KP	Meet in the Middle
	2^{8n}	2^{4n}	16 KP	Dissection
	2^{12n}	$2^{0.5n}$	16 KP	**New** (Section 3.2)
	$2^{7.5n}$	2^{5n}	16 KP	**New** (Section 2.1)
	$2^{7.5n}$	2^{4n}	$2^{0.375n}$ KP	**New** (Section 2.2)
32	2^{8n}	$2^{7.5n}$	16 KP	Meet in the Middle
	2^{15n}	$2^{0.5n}$	16 KP	Meet in the Middle
	2^{8n}	2^{4n}	16 KP	Dissection
	$2^{7.75n}$	$2^{7.75n}$	$2^{0.25n}$ CP	Splice-and-cut [11]
	$2^{12.5n}$	$2^{0.5n}$	16 KP	**New** (Section 3.2)
	$2^{7.75n}$	$2^{7.25n}$	$2^{0.25n}$ CP	**New** (Section 2.2)

KP — Known plaintext, CP — Chosen plaintext

Table 1. Comparing and Summarizing Some of our Results

tel structure) requires $2^{0.5(r+1)n}$ time and about $2^{0.25rn}$ memory.[3] The standard MITM attack can exploit the Feistel structure to reduce the time complexity to $2^{0.5rn}$, at the expense of enlarging the memory complexity to $2^{0.5rn}$. No attacks faster than $2^{0.5rn}$ are known (unless additional assumptions are made on the round functions or on the key schedule) and thus we concentrate in this section on attacks which have this time complexity. Our goal is to combine the benefits of both MITM and dissection attacks in order to reduce the memory complexity

[3] Note that such an attack has to treat every two consecutive Feistel rounds as a single round with an n-bit block and an n-bit key, and it is the last "half-round" which makes it suboptimal.

to $2^{0.25rn}$. We show that this is indeed possible, but at the expense of somewhat enlarging the data complexity of the attack.

First, we present a basic 7-round attack[4] that requires $2^{1.5n}$ time and 2^n memory (compared to $(2^{1.5n}, 2^{1.5n})$ and $(2^{2n}, 2^n)$ in generic MITM and dissection, respectively), and extend it to an attack on $2r+1$ rounds that requires $2^{0.5rn}$ time and about $2^{0.33rn}$ memory. Then, we present a more sophisticated attack that requires $2^{0.5rn}$ time and about $2^{0.25rn}$ memory as desired, but at the expense of enlarging the data complexity to $2^{\lceil (r-4)/r \rceil \cdot 0.5n}$ known plaintexts. Finally, we show that our attacks can be combined with a technique of Isobe and Shibutani [11] that allows extending MITM attacks on Feistel structures by one round using a splice-and-cut technique [2, 19]. We obtain an attack on $\ell = 2r$-round Feistel that requires $2^{(0.5r-0.25)n}$ time, about $2^{0.25(r+1)n}$ memory, and $2^{\max(\lceil \frac{r-4}{r} \rceil, 0.5)\cdot 0.5n}$ chosen plaintexts.

2.1 Attacks with a Low Data Complexity

In this section we present attacks that are time-efficient (i.e., have a time complexity of $2^{0.5rn}$ for $2r+1$ rounds) and also data-efficient (i.e., require only a few known plaintexts, like the standard MITM attack).

A standard MITM attack In order to put our attacks in context, we begin by describing a standard MITM attack on a 7-round Feistel structure.

A 7-Round MITM Attack

1. Obtain 4 plaintext-ciphertext pairs (P^i, C^i) $(i = 1, 2, 3, 4)$.
2. For each value of K_1, K_2, K_3:
   (a) Partially encrypt P^i for $i \in \{1, 2, 3, 4\}$ through the first three rounds and obtain suggestions for R_3^i. Store the suggestions in a list *List* sorted by the R_3^i values.
3. For each value of K_5, K_6, K_7:
   (a) Partially decrypt C^i for $i \in \{1, 2, 3, 4\}$ through the last three rounds, obtain suggestions for R_3^i and search the suggestions in *List*. For each match, retrieve K_1, K_2, K_3, guess[5] K_4, and test the full key using trial encryptions.

The time complexity of Step 2 is about $2^{1.5n}$, which is also the size of *List*. In order to calculate the time complexity of Step 3, we note that we have a total of $2^{1.5n}$ key suggestions from each side of the encryption, each associated with 4 values of R_3^i (filtering conditions). Thus, the expected total number of key suggestions that remain after the $2n$-bit match in Step 3 is $2^{1.5n+1.5n-2n} = 2^n$. For each such suggestion, we guess K_4, and thus we expect to perform about

[4] We consider attacks on less than 7 rounds in Section 3.

[5] We note that K_4 can also be found by a precomputed table instead of guessing, but this will not make a big difference as will be explained in the sequel.

$2^{1.5n}$ trial encryptions. Consequently, the time complexity of Step 3 is also about $2^{1.5n}$, which is the time complexity of the full attack.

A 7-round attack The basic idea of our reduced memory attack is to guess the $n/2$-bit value R_3^1 and to iterate over all the possible guesses as an outer loop. Each guess imposes an $n/2$-bit constraint on the key suggestions for K_1, K_2, K_3 and K_5, K_6, K_7, and thus, allows reducing the expected size of $List$ to 2^n. In order to compute the reduced lists efficiently, we prepare auxiliary tables T_{upper}, T_{lower} that allow retrieving the subkey K_3 (resp., K_5) instantly given the input (L_2, R_2) of round 3 (resp., the output (L_5, R_5) of round 5).

The table T_{upper} is computed as follows. We guess the intermediate value R_2^1 and the subkey K_3. Since $(R_2^1 = L_3^1, R_3^1)$ form the full state after the 3'rd round in the encryption process of P^1, the guesses enable us to partially decrypt through round 3 and obtain $(R_1^1 = L_2^1, R_2^1)$. We store the triple (L_2^1, R_2^1, K_3) in T_{upper}, sorted by (L_2^1, R_2^1). The table T_{lower} is constructed similarly.

```
7-Round Attack with Reduced Memory Complexity

1. Obtain 4 plaintext-ciphertext pairs $(P^i, C^i)$.
2. For each value of $R_3^1$:
   (a) For each value of $K_3$ and $I_3^1 = R_2^1$, compute $L_2^1 = F_3(K_3, I_3^1) \oplus R_3^1$,
       and store the triplet $(L_2^1, R_2^1, K_3)$ in a table $T_{upper}$, sorted according
       to $(L_2^1, R_2^1)$.
   (b) For each value of $K_5$ and $I_5^1 = R_4^1$, compute $R_5^1 = F_5(K_5, I_5^1) \oplus R_3^1$,
       and store the triplet $(L_5^1, R_5^1, K_5)$ in a table $T_{lower}$, sorted according
       to $(L_5^1, R_5^1)$.
   (c) For each value of $K_1, K_2$:
       i. Partially encrypt $P^1$ through the first two rounds to obtain
          suggestions for $R_2^1$ and $L_2^1$.
       ii. Search for the pair $(L_2^1, R_2^1)$ in $T_{upper}$ and obtain suggestions
           for $K_3$. For each suggestion, given $K_1, K_2, K_3$, partially encrypt
           $P^i$ for $i \in \{2, 3, 4\}$ through the first three rounds and obtain
           suggestions for $R_3^i$. Store the suggestions in a list $List1$, sorted
           by the values $R_3^2, R_3^3, R_3^4$.
   (d) For each value of $K_6, K_7$:
       i. Partially decrypt $C^1$ through the last two rounds to obtain sug-
          gestions for $R_5^1$ and $L_5^1 = R_4^1$.
       ii. Search for the pair $(L_5^1, R_5^1)$ in $T_{lower}$ and obtain suggestions for
           $K_5$. For each suggestion, given $K_5, K_6, K_7$, partially decrypt $C^i$
           for $i \in \{2, 3, 4\}$ through the last three rounds, obtain suggestions
           for $R_3^i$ and search the suggestions in $List1$. For each match,
           retrieve $K_1, K_2, K_3$, guess $K_4$, and test the full key using trial
           encryptions.
```

In Steps 2a and 2b, a single round function (either F_3 or F_5) is called once for each guess of (R_3^1, I_3, K_3) (or (R_3^1, I_5, K_5), respectively), and thus, their time complexity is $2^{1.5n}$. The memory complexity of the tables T_{upper} and T_{lower} is 2^n. In Steps 2(c)ii and 2(d)ii there is an average of one match in T_{upper} and T_{lower}, respectively. Thus, on average, we perform a constant number of partial encryption and decryption operations per guess of K_1, K_2 and K_6, K_7 in Steps 2c and 2d, respectively. The expected number of matches in Step 2(d)ii is $2^{n+n-1.5n} = 2^{0.5n}$, and the expected number of trial encryptions (after guessing K_4) is $2^{0.5n+0.5n} = 2^n$. Therefore, the time complexity of each of Steps 2c and 2d is about 2^n each, and the total time complexity of the attack is about $2^{1.5n}$, as in the standard MITM attack. On the other hand, the memory complexity of the attack is reduced from $2^{1.5n}$ to about 2^n, which is the expected number of elements in $List1$ (based on standard randomness assumptions on the round functions).

We note that the time complexity of the attack can be slightly reduced by precomputing a table for F_4, which allows to avoid guessing K_4 in Step 2(d)ii. However, this requires an additional table of size 2^n, i.e., maintaining the 2^n total memory complexity.

Extension to $6r + 1$ rounds. The 7-round attack presented above can be extended to an attack on a $6r + 1$-round Feistel structure, with time complexity of $2^{1.5rn}$ (as in standard MITM) and memory complexity of 2^{rn} (instead of $2^{1.5rn}$ in standard MITM). As the attack is similar to the 7-round attack, we describe it briefly. The reader can follow this attack by verifying that the case $r = 1$ reduces exactly to the attack described above.

First, we obtain $3r + 1$ plaintext/ciphertext pairs (P^i, C^i). Then, the outer loop is performed for all guesses of the r intermediate values $R^1_{3r}, R^2_{3r}, \ldots, R^r_{3r}$. In the inner loop, we guess the $2r$ values $R^1_{3r-1}, R^2_{3r-1}, \ldots, R^r_{3r-1}, K_{3r}, \ldots, K_{2r+1}$. Since for each i, $(R^i_{3r-1} = L^i_{3r}, R^i_{3r})$ forms the full state after the $3r$'th round in the encryption process of P^i, we can partially decrypt this state through rounds $2r+1, \ldots, 3r$ to obtain the corresponding values $(R^i_{2r-1} = L^i_{2r}, R^i_{2r})$. This allows us to prepare a table T_{upper} of size 2^{rn} of the values $((L^i_{2r}, R^i_{2r})^r_{i=1}, K_{2r+1}, \ldots, K_{3r})$, sorted by $((L^i_{2r}, R^i_{2r})^r_{i=1})$. The table T_{lower} is prepared similarly. Note that the $2r$ values guessed from each side are used only for preparing the tables and not in the rest of the attack.

After preparing the tables, we guess the subkeys $K_1, K_2, \ldots, K_{2r}$, obtain the intermediate values $(L^i_{2r}, R^i_{2r})^r_{i=1}$ and access the table T_{upper} to obtain a suggestion for the subkeys $K_{2r+1}, \ldots, K_{3r}$. For each suggestion, given $K_1, K_2, \ldots, K_{3r}$, we partially encrypt P^i for $i \in \{r+1, \ldots, 3r+1\}$ through the first $3r$ rounds and obtain suggestions for R^i_{3r}. We store the suggestions in a list $List1$, sorted by the values $R^{r+1}_{3r}, \ldots, R^{3r+1}_{3r}$. Then, we guess the subkeys $K_{6r+1}, \ldots, \ldots, K_{4r+2}$, access the table T_{lower} to obtain a suggestion for $K_{4r+1}, \ldots, K_{3r+2}$, partially decrypt the ciphertexts to get suggestions for R^i_{3r} $(i = r + 1, \ldots, 3r + 1)$, and search them in $List1$. For each match, we retrieve $K_1, \ldots, K_{3r}$, guess K_{3r+1}, and test the full key using trial encryptions.

The analysis of the attack is similar to that of the 7-round attack described above, and yields time complexity of $2^{1.5rn}$ and memory complexity of 2^{rn}.

The same attack applies for a general odd number $2r' + 1$ of rounds. The time complexity is $2^{r'n}$ (like in MITM), but the memory complexity has to be rounded up to $2^{\lceil 2r'/3 \rceil \cdot 0.5n}$, due to lack of balance between the part of table creation and the rest of the attack.

2.2 Using Multi-Collisions to Further Reduce the Memory Complexity

We now present a more sophisticated variant of the attacks described above, that allows to reduce the memory complexity to the "desired" $2^{0.25(r+1)n}$, with no increase in the time complexity, but at the expense of some increase in the data complexity.

Consider the $6r+1$-round attack described above. In the course of preparing the table T_{upper}, we make an auxiliary guess of the values $R^1_{3r-1}, \ldots, R^r_{3r-1}, K_{3r}, \ldots, K_{2r+1}$, and in the course of preparing the table T_{lower}, we guess $R^1_{3r+1}, \ldots, R^r_{3r+1}, K_{3r+2}, \ldots, K_{4r+1}$. If there was some relation between the guessed values, we could have used this relation to enumerate over some "common relative" in the outer loop of the attack, and thus reduce the memory complexity. We cannot hope for such a relation between the subkeys, as they are assumed to be independent. However, some relation between R^i_{3r-1} and R^i_{3r+1} may exist.

We observe that such a relation can be "created", using multi-collisions. Assume that the partial encryption of the plaintexts $P^1, P^2, \ldots, P^r$ considered in the attack results in an r-multi-collision at the state R_{3r}, i.e., that $R^1_{3r} = R^2_{3r} = \cdots = R^r_{3r}$. In such a case, the r values $R^i_{3r-1} \oplus R^i_{3r+1} = O^i_{3r+1}$ $(i = 1, \ldots, r)$ are all equal! This allows us to enumerate over the $r-1$ values $R^1_{3r-1} \oplus R^i_{3r-1}$ $(i = 2, \ldots, r)$ in the outer loop, such that in the inner loop, a single guess of R^1_{3r-1} provides all the values $\{R^i_{3r-1}\}_{i=2,\ldots,r}$, while a single guess of R^1_{3r+1} provides all the values $\{R^i_{3r+1}\}_{i=2,\ldots,r}$. In order to obtain the multi-collision, we consider $2^{((r-1)/r)\cdot 0.5n}$ known plaintexts (which guarantees that an r-multi-collision exists in the data with a constant probability), and repeat the attack for all r-tuples of plaintext/ciphertext pairs in the data set.

In the description of the algorithm below, we switch from $6r+1$ rounds to $8r-1$ rounds, in order to balance the complexities of all the steps of the attack. Hence, the external guesses are performed at state R_{4r-1}, instead of R_{3r}. Note that for $r = 1$, the algorithm reduces to the 7-round attack presented above.

An $8r-1$-Round Attack Using Multi-Collisions

1. Obtain $2^{((r-1)/r)\cdot 0.5n}$ plaintext-ciphertext pairs (P^i, C^i).
2. For each r-tuple $(P_{i_1}, C_{i_1}), (P_{i_2}, C_{i_2}), \ldots, (P_{i_r}, C_{i_r})$ of plaintext-ciphertext pairs in the data set (hereinafter denoted for simplicity by $(P^1, C^1), \ldots, (P^r, C^r)$), for each possible value of R^1_{4r-1}, and for all possible differences $R^1_{4r-2} \oplus R^i_{4r-2}$ $(i = 2, 3, \ldots, r)$:
   (a) For each $I^1_{4r-1} = R^1_{4r-2}$ and the subkeys $K_{4r-1}, K_{4r-2}, \ldots, K_{2r+1}$, compute[6] (L^i_{2r}, R^i_{2r}) for all $i = 1, \ldots, r$, and store the vector

$((L^i_{2r}, R^i_{2r})_{i=1,\ldots,r}, K_{2r+1}, \ldots, K_{4r-1})$ in a table T_{upper}, sorted according to $(L^i_{2r}, R^i_{2r})_{i=1,\ldots,r}$.

(b) For each $I^1_{4r+1} = R^1_{4r}$ and the subkeys $K_{4r+1}, \ldots, K_{6r-1}$, compute (L^i_{6r-1}, R^i_{6r-1}) for all $i = 1, \ldots, r$, and store the vector $((L^i_{6r-1}, R^i_{6r-1})_{i=1,\ldots,r}, K_{4r+1}, \ldots, K_{6r-1})$ in a table T_{lower}, sorted according to $(L^i_{6r-1}, R^i_{6r-1})_{i=1,\ldots,r}$.

(c) For each value of $K_1, K_2, \ldots, K_{2r}$:
   - i. Partially encrypt $P^1, \ldots, P^r$ through the first $2r$ rounds to obtain suggestions for $(L^i_{2r}, R^i_{2r})_{i=1,\ldots,r}$.
   - ii. Search for $(L^i_{2r}, R^i_{2r})_{i=1,\ldots,r}$ in T_{upper} and obtain suggestions for $K_{2r+1}, \ldots, K_{4r-1}$. For each suggestion, given $K_1, \ldots, K_{4r-1}$, partially encrypt $2r+1$ additional plaintexts P^j for $j \in \{1, \ldots, 2r+1\}$ through the first $4r-1$ rounds and obtain suggestions for R^j_{4r-1}. Store the suggestions in a list $List1$, sorted by the values $\{R^j_{4r-1}\}_{j=1,\ldots,2r+1}$.

(d) For each value of $K_{8r-1}, \ldots, K_{6r}$:
   - i. Partially decrypt $C^1, \ldots, C^r$ through the last $2r$ rounds to obtain suggestions for $(L^i_{6r-1}, R^i_{6r-1})_{i=1,\ldots,r}$.
   - ii. Search for $(L^i_{6r-1}, R^i_{6r-1})_{i=1,\ldots,r}$ in T_{lower} and obtain suggestions for $K_{4r+1}, \ldots, K_{6r-1}$. For each suggestion, given $K_{4r+1}, \ldots, K_{8r-1}$, partially decrypt the additional ciphertexts C'^j for $j \in \{1, \ldots, 2r+1\}$ through the last $4r-1$ rounds, obtain suggestions for $\{R^j_{4r-1}\}_{j=1,\ldots,2r+1}$, and search the suggestions in $List1$. For each match, retrieve $K_1, \ldots, K_{4r-1}$, guess K_{4r}, and test the full key using trial encryptions.

The inner loop of the algorithm is repeated for each of the $2^{(r-0.5)n}$ values of the external guess. In each of Steps 2(a) and 2(b), we perform 2^{rn} partial encryptions/decryptions and construct a table of size 2^{rn}. In Steps 2.(c).ii. and 2.(d).ii. there is an average of one match in T_{upper} and T_{lower}, respectively. Thus, on average, we perform a constant number of partial encryption and decryption operations per guess of $K_1, \ldots, K_{2r}$ and $K_{6n}, \ldots, K_{8n-1}$ in Steps 2.(c) and 2.(d), respectively. The expected number of matches in Step 2.(d).ii is $2^{rn+rn-(r+0.5)n} = 2^{(r-0.5)n}$, and the expected number of trial encryptions (after guessing K_4) is $2^{(r-0.5)n+0.5n} = 2^{rn}$. Therefore, the time complexity of Steps 2.(c) and 2.(d) is about 2^{rn} and the total time complexity of the attack is about $2^{(2r-0.5)n}$, as in the standard MITM attack. On the other hand, the memory complexity of the attack is reduced from $2^{(2r-0.5)n}$ to about 2^{rn}, which is the expected number of elements in $List1$.

The same attack applies for a general odd number $2r'+1$ of rounds. The time complexity is $2^{r'n}$ (like in MITM), the memory complexity has to be rounded up

[6] The computation of (L^i_{2r}, R^i_{2r}) for all $i = 1, \ldots, r$ is feasible, since by the assumption that $(P^1, C^1), \ldots, (P^r, C^r)$ is an r-multi-collision, the value R^1_{4r-2} along with the externally guessed values are sufficient for obtaining the values $R^2_{4r-2}, \ldots, R^r_{4r-2}$.

to $2^{\lceil 0.5(r'+1) \rceil \cdot 0.5n}$, due to lack of balance between the part of table creation and the rest of the attack, and the data complexity is $2^{\lceil \frac{r'-3}{r'+1} \rceil \cdot 0.5n}$ known plaintexts.

2.3 Attacks on Feistel Structures with an Even Number of Rounds

In this section we show that all the attacks presented above can be combined with the recent technique of Isobe and Shibutani [11] that allows to extend MITM attacks on Feistel structures by one round, at the expense of a relatively small increase in the time complexity and of using $2^{0.25n}$ chosen plaintexts. The generic attack of [11] on a $2r$-round Feistel structures requires $2^{(0.5r-0.25)n}$ time, $2^{(0.5r-0.25)n}$ memory, and $2^{0.25n}$ chosen plaintexts. Our attacks allow to either reduce the memory complexity to about $2^{0.33(r-1)n+0.25n}$ with no effect on the time and data complexities or to reduce the memory complexity all the way to about $2^{0.25(r+1)n}$, while increasing the data complexity to $2^{\max(\lceil \frac{r-4}{r} \rceil, 0.5) \cdot 0.5n}$ chosen plaintexts, with no effect on the time complexity. In the specific case of the 8-round Feistel structures considered in [11], our attack reduces the memory complexity significantly from $2^{1.75n}$ to $2^{1.25n}$, without affecting the other complexities.

Consider a MITM attack on a $2r$-round Feistel structure. In a standard application (skipping the guess of the middle subkey), the attack is not balanced, as r subkeys are guessed on one side of the MITM, while $r-1$ subkeys are guessed on the other side. The attack of [11] aims to rebalance the attack, by "splitting" the guess of one subkey between the two sides. The basic idea behind the attack is as follows. If in all plaintexts used in the attack, the right half is equal to a fixed value R_0, then in all encryptions, we have $R_1 = Const \oplus L_0$, where $Const$ is an unknown constant that depends on K_1. This allows to replace the $2r$-round Feistel with an equivalent construction that consists of a $2r-1$-round Feistel, prepended by an addition of $Const$ to the right half of the plaintext (that can be treated as a subkey addition). This, in turn, allows to use the splice-and-cut technique [2, 19] to "split" the guess of $Const$ between the two sides of the MITM, at the price of using $2^{n/4}$ chosen plaintexts. As a result, the attack becomes balanced and the time complexity is reduced from $2^{0.5rn}$ to $2^{(0.5r-0.25)n}$. For a full description of the attack, see [11].

In order to incorporate the splice-and-cut procedure of [11] into our attacks, we consider the equivalent $2r-1$-round variant, perform one of our attacks against it, and insert the splice-and-cut procedure into the "key guessing" part of the attack (i.e., Steps 2(c) and 2(d)), without changing the "table construction" part (Steps 2(a) and 2(b)). As a result, the time complexity of our $2r-1$-round attack is increased by a factor of $2^{0.25n}$ to $2^{(0.5r-0.25n)}$ (just like the complexity of the attack of [11]), and the memory complexity is increased by a factor of $2^{0.25n}$ to either $2^{0.33(r-1)n+0.25n}$ (in the low data complexity attack) or to about $2^{0.25(r+1)n}$ (in the attack using multi-collisions). As for the data complexity, in our low data complexity attack the data complexity increases to $2^{0.25n}$ chosen plaintexts (required for the splice-and-cut procedure), and in the multi-collision based attack the data complexity increases to $2^{\max(\lceil \frac{r-4}{r} \rceil, 0.5) \cdot 0.5n}$, as the plaintexts required for the multi-collision can be chosen in such a way that they will contain the structures required for the splice-and-cut attack.

3 Memory-Restricted Attacks on Feistel Structures

After analyzing the most time-efficient attacks, a natural question to explore is what are the most memory-efficient attacks one can devise against an r-round Feistel structure. Specifically, we shall concentrate on the problem of devising attacks with $2^{0.5n}$ memory complexity, since it is the smallest amount of memory that enables us to list all the values of a single subkey or of half a block.

With such a restriction, a standard meet in the middle attack takes $2^{(\ell-2)\cdot 0.5n}$ on an ℓ-round Feistel, and one can trade time for memory. One can also try to consider the original dissection attack of [9]. However, as noted before, dissection takes at least 2^n memory to store all the possible values of a full block, which implies that it cannot be used in this context, even though we adopt several concepts from it.

Section 3.1 presents our new attack on 5-round Feistel structures that uses 2^n time and $2^{0.5n}$ memory. This is to be compared with meet in the middle attacks that use time of $2^{1.5n}$ with $2^{0.5n}$ memory or time of 2^n with 2^n memory. We then generalize the attack to more rounds, and show in Section 3.2 how to increase the gain over meet in the middle attacks as the number of rounds increases. The attacks in this section assume that the round function is efficiently invertible given the round's subkey. Due to space constraints, we postpone the discussion of this assumption to Appendix B, but note that it holds for almost any Feistel block cipher we are aware of.

3.1 A Memory-Restricted Attack against 5-Round Feistel Constructions

The algorithm of our basic 5-round attack is as follows.

A 5-Round Attack with $2^{0.5n}$ Memory ($DF_2(5,1)$)

1. Obtain 4 plaintext-ciphertext pairs (P^i, C^i) $(i = 1, 2, 3, 4)$.
2. For each value of $R_2^1 = I_3^1$:
   - (a) Compute $O_2^1 = I_3^1 \oplus R_0^1$ and $O_4^1 = I_3^1 \oplus R_4^1$.
   - (b) For each value of K_1, compute $R_1^1 = F_1(K_1, I_1^1) \oplus L_0^1$ and store the pair (R_1^1, K_1) in a table T_{upper} sorted according to R_1^1.
   - (c) For each value of K_2, compute $R_1^1 = F_2^{-1}(K_2, O_2^1) \oplus R_2^1$, search for the value R_1^1 in T_{upper} and obtain suggestions for K_1. For each suggestion, given K_1, K_2, compute R_2^2, R_2^3 for P^2, P^3. Store the suggestions (R_2^2, R_2^3, K_1, K_2) in a list $List1$ sorted by the value of R_2^2.
   - (d) For each value of K_5, compute $R_3^1 = F_5(K_5, I_5^1) \oplus L_5^1$ and store the pair (R_3^1, K_5) in a table T_{lower} sorted according to R_3^1.

(e) For each value of K_4, compute $R_3^1 = F_4^{-1}(K_4, O_4^1) \oplus R_2^1$, search for the value R_3^1 in T_{lower} and obtain suggestions for K_5. For each suggestion, given K_4, K_5, compute R_2^2, R_2^3 from C^2, C^3, and search the suggestion in $List1$. For each match, retrieve K_1, K_2, guess K_3, and test the full key using trial encryptions.

For reasons which will become apparent later, we call the above attack $DF_2(5,1)$.

As before, T_{upper}, T_{lower}, are each of size $2^{0.5n}$. The memory complexity of $List1$ depends on the number of (K_1, K_2) pairs that satisfy the meet in the middle condition on the value of I_2 in Step 2c. For sufficiently random round functions, we expect about $2^{0.5n}$ such (K_1, K_2) pairs.

The time complexity of the algorithm is 2^n, as it iterates over $2^{n/2}$ values for R_2^1, and each step of the loop takes $2^{0.5n}$ operations (besides the XOR of Step 1a, which takes less).

We note that the time complexity of the attack can be slightly reduced by precomputing a table for F_3 (given its input value I_3^1), which allows to avoid guessing K_3. However, this requires an additional table of size $2^{0.5n}$, which increases the memory complexity by a small constant factor.

Finally, it is important to note that given only two plaintext-ciphertext pairs, the above attack finds all possible $(K_1, K_2, K_3, K_4, K_5)$ in time 2^n and memory of $2^{0.5n}$. The expected number of candidates is about $2^{0.5n}$. This observation will be used in the subsequent attacks.

3.2 Extension to More Rounds

As the time complexity of a MITM attack with $2^{0.5n}$ memory on an r-round Feistel structure is $2^{(\ell-2)0.5n}$, we define the *gain* over MITM of an attack on ℓ-round Feistel that requires T time and $2^{0.5n}$ memory by $(\ell-2) - \log_2(T)/0.5n$. Thus, the 5-round attack presented above has gain of 1. We denote by $Gain(\ell)$ the maximal gain achieved by an ℓ-round attack with $2^{0.5n}$ memory. In this section, we extend the 5-round attack to a sequence of attacks which show that asymptotically, $Gain(\ell) = \Omega(\sqrt{\ell})$, and compute the sequence of round numbers for which the gain is strictly increased.

Obviously, it is possible to attack 6-round Feistel by guessing K_6, and applying the 5-round attack for each guess. The result is an attack of $2^{1.5n}$ time and $2^{0.5n}$ memory on 6-round Feistel structure. This approach can obviously be extended, but maintains a gain of 1.

Attacking 10-Round Feistel Constructions To increase the gain to 2, we consider the case of 10-round Feistel, and develop the following attack:

10-Round Dissection Attack with $2^{n/2}$ Memory ($DF_5(10,4)$)

1. Obtain 5 plaintext-ciphertext pairs $(P^i, C^i)_{i=1,\dots,5}$.
2. For each value of (L_5^1, R_5^1) and (L_5^2, R_5^2):

(a) Run $DF_2(5,1)$ on the first 5 rounds, and obtain a list of $2^{0.5n}$ candidates for $(K_1, K_2, K_3, K_4, K_5)$.
(b) For each candidate for the subkeys $(K_1, K_2, K_3, K_4, K_5)$, partially encrypt a third plaintext P^3 through the first 5 rounds, and store the suggestions (with the keys) $(L_5^3, R_5^3, K_1, \ldots, K_5)$ in a list $List1$ sorted by the values of (L_5^3, R_5^3).
(c) Run $DF_2(5,1)$ on the last 5 rounds, and obtain a list of $2^{0.5n}$ candidates for $(K_6, K_7, K_8, K_9, K_{10})$.
(d) For each candidate for the subkeys $(K_6, K_7, K_8, K_9, K_{10})$, partially decrypt C^3 through the last 5 rounds, and obtain suggestions for (L_5^3, R_5^3) and search the suggestion in $List1$. For each match, retrieve $K_1, \ldots, K_5$, and test the full key using trial encryptions.

It is easy to see that the 10-round attack calls 2^{2n} times two independent 5-round attacks, each running in time 2^n. Hence, the time complexity of the 10-round attack is 2^{3n}, and the memory complexity is $2^{0.5n}$. Hence, the gain of the 10-round attack is 2.

We use the following notations, $DF_2(5,1)$ denotes the 5-round attack presented earlier, as it is a generalized Dissection attack on Feistel xistructures with 5 rounds, which guesses one $n/2$-bit value after two rounds of encryption. Similarly, $DF_5(10,4)$ attacks 10 rounds by guessing 4 $n/2$-bit values after 5 rounds of encryption (i.e., two full intermediate encryption values). As in [9], we now explore how to extend the attack to more rounds.

Attacking 15-Round Feistel Constructions We can increase the gain to 3, when attacking 15-round Feistel: Guess two complete intermediate encryption values after 5 and after 10 rounds (a total of four internal values), and run the 5-round attack three times subsequently (on rounds 1–5, 6–10, and 11–15), resulting in $2^{0.5n}$ candidates for each set of corresponding subkeys. Then, the correct value can be found by an additional standard MITM. If the memory is kept at $2^{0.5n}$, this means that the last layer of the MITM takes 2^n time. Hence, the total time complexity of the 15-round attack is 2^{5n}, and thus, its gain is 3.

In other words, $DF_5(15,4)$ is based on guessing four $n/2$-bit internal state words after 5 rounds, and running recursively running $DF_2(5,1)$ on the first rounds, and running $DF_5(10,4)$ on the last rounds. This contrasts with the works of [9], where each new layer in the dissection was of a different size.

The different "expanding" rule is due to two inherent differences between the dissection attacks presented in [9] and our new attacks. First, in our attacks, guessing a full intermediate state adds two "units" of time complexity (as each full state contains two $n/2$-bit values) compared with one in the case of regular dissection attacks. The second difference is more subtle, but has a larger effect on the way the attack scales up: In our attacks we can enjoy the "Feistel" advantage (meeting in the middle only on $n/2$ bits) only once in the internal recursion step (e.g., in the 5-round attack), as the external steps must rely on guessing a full internal state. The second difference is already apparent in the transition from

5-round to 10-round (comparing $DF_2(5,1)$ and $DF_5(10,4)$: whereas the 5-round attack guesses a single $n/2$-bit value, the 10-round attack starts by guessing 4 such values.

Attacking 22-Round Feistel Structures We now turn our attention to 22-round Feistel structures. Due to the differences between regular dissection attacks and attacking Feistels, the extension of the 15-round attack into the 22-round attack follows a slightly different path than the extension from the 10-round to the 15-round:

22-Round Dissection Attack with $2^{n/2}$ Memory ($DF_7(22,6)$)

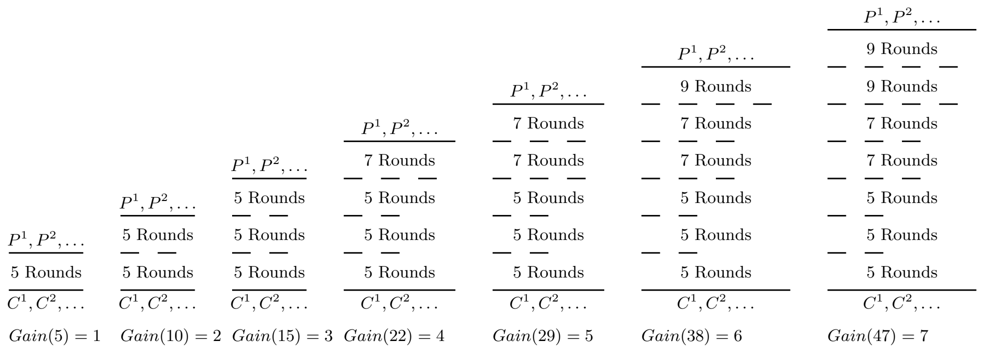
1. Obtain 11 plaintext-ciphertext pairs $(P^i, C^i)_{i=1,\ldots,11}$.
2. For each possible value of (L_7^1, R_7^1), (L_7^2, R_7^2), and (L_7^3, R_7^3):
   (a) Run $DF_2(7,3)$ on the first 7 rounds, and obtain a list of $2^{0.5n}$ candidates for $(K_1, K_2, \ldots, K_7)$.
   (b) For each candidate for the subkeys $(K_1, K_2, \ldots, K_7)$, partially encrypt a fourth plaintext P^4 through the first 7 rounds, and store the suggestions (with the keys) $(L_7^4, R_7^4, K_1, \ldots, K_7)$ in a list $List1$ sorted by the values of (L_7^4, R_7^4).
   (c) Run $DF_5(15,4)$ on the last 15 rounds, and obtain a list of $2^{1.5n}$ candidates for $(K_8, K_9, \ldots, K_{22})$.
   (d) For each candidate for the subkeys $(K_8, \ldots, K_{22})$, partially decrypt C^4 through the last 15 rounds, and obtain suggestions for (L_7^3, R_7^3) and search the suggestion in $List1$. For each match, retrieve $K_1, \ldots, K_7$, and test the full key using trial encryptions.

It is easy to see that the memory complexity of the attack is $2^{0.5n}$. The 7-round attack $DF_2(7,3)$ is actually $DF_2(5,1)$, run when K_6, K_7 are guessed, i.e., takes 2^{2n} time for $2^{0.5n}$. Both the 7-round attack and the 15-round attack are called 2^{3n} times, suggesting a total running time of 2^{8n}, i.e., the attack offers a gain of 4.

Generalization to More Rounds In the second generalization, we prepend 5 rounds to the 10-round attack (obtaining 15 rounds in total), and again, guess two full internal states in order to run two independent attacks — one on 5 rounds, and the other on 10 rounds. The 22-round attack is based on guessing three additional full internal states, and prepending 7 rounds before the 15-round attack. Similarly, a 29-round attack with a gain of 5 can be obtained by prepending 7 rounds before the 22-round attack and guessing three additional full internal states. It is now apparent that the sequence of round numbers for which the gain increases is $\{5, 10, 15, 22, 29, 38, 47, 58, 69, \ldots\}$,[7] which shows that for all k, $Gain(2k^2+6k+2) \geq 2k$ and $Gain(2k^2+8k+5) \geq 2k+1$. It follows that asymptotically, $Gain(\ell)$ grows as $\sqrt{2\ell}$.

[7] A gain of j is first achieved when attacking $5j + 2\sum_{i=1}^{j-1} \lfloor i/2 \rfloor$ rounds.

The general form of the recursion is described in Figure 2. We note that the recursion yields only a lower bound on $Gain(\ell)$. The discussion about the optimality of this lower bound will be presented in the full version of the paper.



Dashed lines represent internal state values which are guessed in the attack.

Fig. 2. Generalizing the 5-Round Attack and Increasing the Gain

4 Applications to Concrete Cryptosystems

While the new techniques presented in Sections 2 and 3 are generic, they can also be used to improve the memory complexity of the best known attacks on several block ciphers, including CAST-128, DEAL, and FOX. It turns out that even a straightforward application of the generic techniques already yields improvements over previously known attacks, and if we also exploit the specific properties of the analyzed cipher, the improvements become even more significant. Due to space constraints, we present in this section only a very brief description of our improved attacks on specific schemes. The full description of the attack on CAST-128 is given in Appendix A, and the full descriptions of the other applications will be given in the full version of the paper.

4.1 Lower Memory Attacks on DEAL

DEAL [15] is a 128-bit Feistel structure, designed in 1997 by Knudsen and submitted as a candidate to the AES selection process. The round function of DEAL is extremely complex — it consists of a full keyed DES [17] encryption. (Recall that DES has 64-bit blocks and 56-bit keys.) In return, the number of rounds is rather small — 8 rounds for the 256-bit key variant and 6 rounds for the 128-bit and 192-bit key variants. The only published attack on the full 8-round DEAL is a standard MITM attack mentioned by the designers [15], with time and memory complexities of $2^{4 \cdot 56} = 2^{224}$. The generic attack of [11] on 8-round Feistel structures (described in Section 2.3) can be used to reduce the time complexity to $2^{3 \cdot 56+32} = 2^{200}$, with memory complexity of 2^{200} and data complexity of 2^{32} chosen plaintexts. We show that by using our techniques, the memory complexity can be significantly reduced to 2^{144}, while maintaining the same data and time complexities.

Our generic attack on 8-round Feistel structures (presented in Section 2.3) requires $2^{1.75n}$ time, $2^{1.25n}$ memory, and $2^{0.25n}$ chosen plaintexts. A direct application of this attack to DEAL, taking into account the fact that each round key has only 56 bits rather than 64, yields time complexity of $2^{64+56+56+32} = 2^{208}$. However, the time complexity can be reduced to 2^{200} by performing the external enumeration over 56 out of the 64 bits of the intermediate value R_4^1, rather than over the full value. In the phase of table preparation, we guess the remaining 8 bits of R_4^1, along with the auxiliary guess of R_3^1, K_4, and thus, the complexity of this step is increased to $2^{8+64+56} = 2^{128}$. However, this complexity is still dominated by the $2^{32+56+56} = 2^{144}$ complexity of the key guessing step. As a result, the overall time complexity remains $2^{56+56+56+32} = 2^{200}$, the memory complexity is reduced to $2^{56+56+32} = 2^{144}$, and the data complexity remains 2^{32} chosen plaintexts.

In a similar way we can reduce the memory complexity of the improved MITM attack on the full 6-round DEAL with 192-bit keys from $2^{56+56+32} = 2^{144}$ to $2^{56+32} = 2^{88}$ (while keeping the 2^{144} time complexity and 2^{32} data complexity unchanged), using a modification of the generic attack on 6-round Feistel structures presented in Section 3. The resulting attack in this case is the best known attack which uses a practical data complexity, but is outperformed (in terms of time complexity) by the impossible differential attack presented by Knudsen [15] that requires 2^{121} time but uses an unrealistic amount of 2^{70} chosen plaintexts. Table 2 compares the complexities of attacks against the variants of DEAL.

4.2 A Lower Memory Attack on CAST-128

CAST-128 [1] is a 16-round Feistel structure that uses 64-bit inputs and 128-bit keys, which was designed in 1996 by Adams. It is used in several real-life products, such as GPG, PGP, and SSH2. The currently best known attack on the cipher (excluding weak-key attacks such as [21]) is the MITM attack of Isobe and Shibutani [11], breaking 8 out of the 16 rounds in time complexity of about 2^{118}, using 8 chosen ciphertexts and a memory complexity of 2^{111} words. Using our techniques, the memory complexity can be reduced significantly to 2^{64}, while maintaining the same data and time complexities.

The general structure of CAST-128 is shown in Figure 3 (which describes only 8 out of its 16 rounds). In the round function F_i, the 32 LSBs of the 37-bit round key K_i (denoted as K_{m_i}) are first either XORed, added (modulo 2^{32}), or subtracted (modulo 2^{32}) from R_{i-1}. Then, the result is rotated to the left by 0–31 bits, according to the value of the 5 MSBs of K_i (denoted as K_{r_i}). Finally, a key-less function f_i is applied to the result.

Since each round key of CAST-128 is of 37 bits, the time complexity of a basic MITM attack on a 8-round variant is $2^{4 \cdot 37} = 2^{148}$. Using the generic attack of [11] on 8-round Feistel structures (described in Section 2.3), the time complexity can be reduced to $2^{3 \cdot 37 + 16} = 2^{127}$, which is only slightly faster than exhaustive key search. Isobe and Shibutani [11] showed that the specific structure of the round function of CAST-128 can be used to further reduce the time complexity to 2^{118}. The main idea of [11] is that by fixing most of the ciphertext bits (in all ciphertexts) to a constant value and exploiting the specific round function structure, the amount of key material required for partial decryption can be *reduced* (and not only divided between the upper and lower halves of the MITM, like in the generic attack). To achieve this, [11] consider an equivalent 7-round Feistel structure, with different round functions F'_5, F'_6, F'_7 that imitate the four round functions F_5, F_6, F_7, F_8 for the specifically chosen ciphertexts. See Appendix A for details of the attack.

As in case of the generic attack of Isobe and Shibutani discussed in Section 2.3, we can incorporate the advanced attack procedure in the "key guessing" part of our generic memory-efficient attack on 7-round Feistel structures. As a result, the memory complexity of the attack is reduced from 2^{111} to 2^{79}, without increasing the time and data complexities. The memory complexity can be further reduced using a refined attack that exploits the relatively simple round function of CAST-128. As we show in Appendix A, it is possible to guess two intermediate values R_3^1, R_3^2 (instead of a single value in the generic 7-round attack) and to structures separate tables T_{upper1}, T_{upper2} for rounds 2,3 (and similarly, separate tables T_{lower1}, T_{lower2} for rounds 5,6). These tables make use of complex differential properties of F_2 and F_6 that simultaneously combine different operations over $GF(2)$ and over $GF(2^{32})$. As a result, the memory complexity is reduced to 2^{64} with no effect on the time complexity. The details of this (rather involved) attack are given in Appendix A.

4.3 Lower Memory Attacks on Other Cryptosystems

We conclude this section by mentioning briefly applications of our generic techniques to several other specific and generic structures.

1. **FOX** Fox is a non-Feistel block cipher. The memory complexity of all the attacks of [12] on round-reduced variants of the block cipher FOX (namely, on 6 and 7-round FOX-64 and FOX-128), which are currently the best known attacks on FOX, can be reduced by a factor of 2^{16}. We note that in these attacks, the 16 bits of filtering on which the attack iterates in the outer loop of the attack are not actual state bits, but rather linear combinations of state bits. However, our techniques are still applicable in this case, as in the inner loop, for each side of the attack, we simply complement these linear combinations to obtain an intermediate encryption state of FOX, and invert its round function as done in our generic attacks.
2. **Camellia** The memory complexity of the attacks of [11] on reduced variants of Camellia can be reduced by a factor of at least 2^{16} (depending on the

attack). We note however that MITM attacks are not the best known attacks on Camellia (in terms of the number of rounds).[8]

3. **Feistel-2 Schemes** The memory complexity of the attacks of [11] on the 8 and 9-round Feistel-2 scheme (which is a more specific Feistel implementation compared to the generic Feistel-1) with a $2n$-bit key can be reduced from about $2^{1.5n}$ to 2^n. We note that these MITM attacks are the best known attacks on this specific Feistel-2 only when the data complexity is limited. With large data complexity in the chosen plaintext model, the attacks of [10] have superior time complexities.

5 Conclusions

In this paper we introduced some new cryptanalytic techniques, and combined the known techniques of MITM and dissection in new ways which enabled us to merge their advantages and avoid their disadvantages. Taken together, these techniques allowed us to develop improved attacks on Feistel structures with more than four rounds, and to improve the best known concrete attacks on several well known block ciphers.

References

1. Adams, C.: The CAST-128 Encryption Algorithm. RFC 2144 (1997), https://tools.ietf.org/html/rfc2144
2. Aoki, K., Sasaki, Y.: Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5381, pp. 103–119. Springer (2008)
3. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT. Lecture Notes in Computer Science, vol. 7073, pp. 344–371. Springer (2011)
4. Bogdanov, A., Rechberger, C.: A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6544, pp. 229–240. Springer (2010)
5. Boura, C., Naya-Plasencia, M., Suder, V.: Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In: Sarkar and Iwata [18], pp. 179–199
6. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-Middle: Improved MITM Attacks. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 222–240. Springer (2013)

[8] Although they can handle fewer rounds, the advantage of MITM attacks over other attacks on Camellia (namely, impossible differential attacks [5]) is their low data complexity.

7. Chaum, D., Evertse, J.H.: Cryptanalysis of DES with a Reduced Number Of Rounds: Sequences of Linear Factors in Block Ciphers. In: Advances in Cryptology, CRYPTO 85. pp. 192–211. Springer (1986)
8. Diffie, W., Hellman, M.E.: Cryptanalysis of the NBS Data Encryption Standard. Computer 10(6), 74–84 (1977)
9. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 719–740. Springer (2012)
10. Guo, J., Jean, J., Nikolic, I., Sasaki, Y.: Meet-in-the-Middle Attacks on Generic Feistel Constructions. In: Sarkar and Iwata [18], pp. 458–477
11. Isobe, T., Shibutani, K.: Generic Key Recovery Attack on Feistel Scheme. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8269, pp. 464–485. Springer (2013)
12. Isobe, T., Shibutani, K.: Improved All-Subkeys Recovery Attacks on FOX, KATAN and SHACAL-2 Block Ciphers. Presented at FSE 2014. To appear in Lecture Notes in Computer Science. (2014)
13. Junod, P., Vaudenay, S.: FOX : A New Family of Block Ciphers. In: Handschuh, H., Hasan, M.A. (eds.) Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3357, pp. 114–129. Springer (2004)
14. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: Canteaut, A. (ed.) Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. Lecture Notes in Computer Science, vol. 7549, pp. 244–263. Springer (2012)
15. Knudsen, L.: DEAL - A 128-bit Block Cipher (1998), nIST AES Proposal
16. Luby, M., Rackoff, C.: How to Construct Pseudorandom Permutations from Pseudorandom Functions. SIAM J. Comput. 17(2), 373–386 (1988)
17. National Bureau of Standards: Data encryption standard. Federal Information Processing Standards Publications (FIPS) 46 (1977)
18. Sarkar, P., Iwata, T. (eds.): 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, Lecture Notes in Computer Science, vol. 8873. Springer (2014)
19. Sasaki, Y., Aoki, K.: Preimage Attacks on Step-Reduced MD5. In: Mu, Y., Susilo, W., Seberry, J. (eds.) Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia, July 7-9, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5107, pp. 282–296. Springer (2008)
20. Wang, L., Sasaki, Y.: Finding Preimages of Tiger Up to 23 Steps. In: Hong, S., Iwata, T. (eds.) Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6147, pp. 116–133. Springer (2010)
21. Wang, M., Wang, X., Chow, K., Hui, L.C.K.: New Differential Cryptanalytic Results for Reduced-Round CAST-128. IEICE Transactions 93-A(12), 2744–2754 (2010)

A Full Description of the Lower Memory Attack on 8-Round CAST-128

CAST-128 is 64-bit a Feistel structure which was designed in 1996 by Adams [1], and is currently deployed in several products and protocols such as PGP, GPG, and SSH2. The currently best known attack on the cipher (not including weak-key attacks such as [21]) is a MITM attack (described in [11]), breaking 8 out of the full 16 rounds in time complexity of about 2^{118}, using 8 chosen ciphertexts and memory complexity of 2^{111} words. In this section, we apply the dissection technique to CAST, reducing the memory complexity to 2^{64}, while maintaining the same data and time complexities.

A.1 Description of CAST-128

CAST-128 accepts keys of length between 40 and 128 bits. We concentrate in this paper on the most widely used and most secure version of 128-bit key, for which the previous attack of [11] (and our improved attack) is faster than exhaustive search.

The general structure of CAST-128 is shown in Figure 3 (which describes only 8 out of 16 rounds). We denote by $x \boxplus y$ addition modulo $2^{n/2}$, by $x \boxminus y$ subtraction modulo $2^{n/2}$, and by $x \ggg b$ (or $x \lll b$) rotation to the left (or to the right) by b bits. Depending on the round number, i, the 32 LSBs of the 37-bit subkey K_i (denoted as K_{m_i}) are first either XORed, added (modulo 2^{32}), or subtracted (modulo 2^{32}) from R_{i-1}. Then, the result is rotated to the left by 0–31 bits, according to the value of the 5 MSBs of K_i (denoted as K_{r_i}). Finally, a key-less function f_i is applied to the result. We note that CAST-128 alternates between 3 different functions, f_i, but our attack (and the previous one [11]) does not exploit this property, and we assume that the f_i functions are independent.

In total, each round function of CAST-128 depends on the 37-bit subkey $K_i = (K_{m_i}, K_{r_i})$, which is derived from the master key using a key schedule algorithm. However, our attack (and the previous one [11]) does not exploit dependencies between the subkeys, and thus it applies to any key schedule.

A.2 The Previous Attack on 8-Round CAST-128 [11]

It is possible to apply the attacks of Section 4.2 to 7-round CAST-128. In a straightforward application of these attacks, we requires 4 known plaintext-ciphertext pairs, and break the scheme in time and memory complexities of about $2^{3\cdot(32+5)} = 2^{111}$ (as we guess 3 subkeys from each side of the computation). Using our generic technique, the memory complexity can be optimized to about $2^{111-32} = 2^{79}$.

We can try to attack an additional round of CAST-128 using the splice-and-cut technique, as in Section 2.3. However, a straightforward application of this technique has time complexity of more than $2^{111+16} = 2^{127}$, and is not much faster than exhaustive search. The attack of [11] allows[9] to break 8 rounds of CAST-128 by exploiting the internal structure of the CAST-128 round functions, described in Figure 3. The attack (sketched in Figure 4) uses 8 (chosen) ciphertexts, and computes 8 filtering conditions (of 32 bits) on $R_3 = L_4$ from both sides of the encryption process, where the computation from the encryption side is done in a straightforward way by partially encrypting the 8 plaintexts for all possible values of K_1, K_2, K_3. On the other hand, a straightforward computation of $R_3 = L_4$ from the decryption side requires iterating over all possible values of K_5, K_6, K_7, K_8, which has an inefficient time complexity of at least $2^{4 \cdot 37} = 2^{148}$.

The main idea of [11] (which is called "function reduction") is to reduce the amount of key material required for partial decryption by fixing some ciphertext bits to constants. In particular, the attack requests the decryption of 8 chosen ciphertexts (L_8^i, R_8^i) for which all but the 3 MSBs of L_8 are fixed to zero (or an arbitrary constant), and these 3 MSBs range over their 8 possible values. This essentially allows to compute the suggestions for R_3^i without having to guess K_8, by starting the computation from round 7, and evaluating the decryption process for 8 values $(L_7'^i, R_7'^i)$, where $R_7'^i = R_8^i$ is known to be zero (from the choice of ciphertexts), the 29 LSBs of $L_7'^i$ are set to zero and their 3 MSBs range over their 8 possible values. In particular, [11] defines subkeys $K_5', K_6' = K_6, K_7'$, where $K_5' = (K_{m_5}', K_{r_5})$ is a new 37-bit subkey (that is derived from the original K_5 and K_8), and K_7' is a new 40-bit subkey (that is derived from the original K_7 and K_8). For these subkeys, [11] shows that $R_3'^i = R_3^i$ (for all $1 \leq i \leq 8$) and thus the amount of key material required to compute the filtering conditions is reduced to $37 + 37 + 40 = 114$.

The high-level description of the resultant scheme is shown in Figure 4. We note that the new round function F_7' has a slightly more complicated structure than F_7, and its exact details are irrelevant to our subsequent attack. More generally, the full details of the function reduction procedure are not required to understand the rest of this paper, and are given in [11]. The total time complexity of the attack of [11] is about 2^{118}, while its memory complexity is about 2^{111}.

A.3 Lower Memory Attack on 8-Round CAST-128

In this section, we adapt our techniques to reduce the memory complexity of the attack of [11]. First, we note that it is possible to reduce the memory complexity by a factor of 2^{32} (from 2^{111} to $2^{111-32} = 2^{79}$) by iterating over the 32-bit value of R_3^1. In order to efficiently enumerate over the solutions from both sides of the computation, we preprocess F_3 and F_5 similarly to the attack of Section 4.2. However, in the case of CAST-128, we can exploit the (relatively) simple round functions in order to further reduce the memory complexity to about 2^{64}.

The main idea of the optimized attack is to iterate over two 32-bit values of R_3^1 and R_3^2 (64 bits in total). However, in this case, the algorithm for efficiently enumerating over the solutions (subkeys) from both sides of the computation is more involved than the previous attack described in this paper.

[9] We note that [11] uses slightly different notation than this paper.

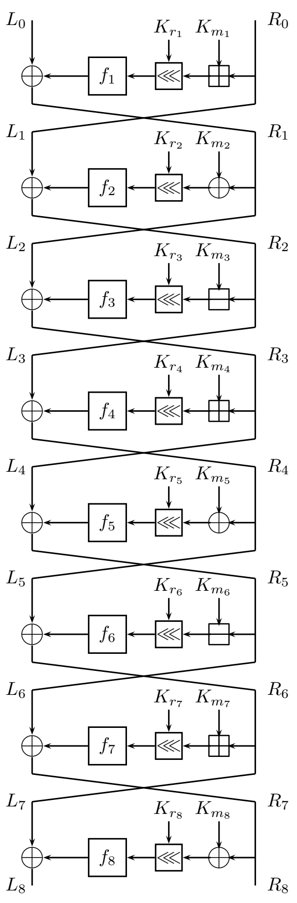


Fig. 3. 8-Round CAST-128

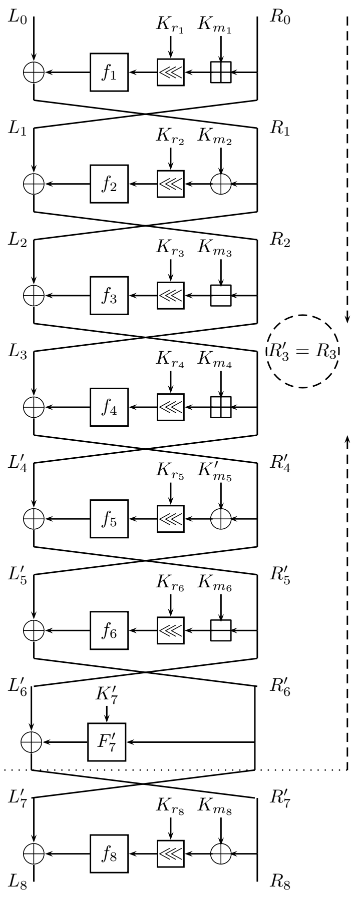


Fig. 4. Attack on 8-Round CAST-128

Decryption Side We start by describing the enumeration algorithm for the decryption side. The algorithm requires preprocessing the two key-less functions f_5 and f_6. For f_5, we prepare a table, T_5, which allows to invert it efficiently, namely, given a 32-bit word y, $T_5[y]$ contains all the words x such that $f_5(x) = y$. Such a table can be easily prepared in 2^{32} time, and it requires 2^{32} memory. For f_6, we prepare a table, T_6, whose entries are sorted according to two 32-bit words Δ_1, Δ_2. The entry $T_6[\Delta_1, \Delta_2]$ contains all the words x such that $f_6(x) \oplus f_6(x \boxminus \Delta_1) = \Delta_2$ (in other words T_6 is a difference distribution table of f_6, where the input differences are subtraction-based, and the output difference operation is XOR). Note that, on average, there exists one 32-bit word x that satisfies the 32-bit condition imposed by Δ_1, Δ_2. Thus, T_6 requires 2^{64} words of memory, and can be computed in time complexity 2^{64} according to the preprocessing algorithm below.

```
T_6 Computation

1. For each value of $\Delta_1$:
   (a) For each value of $x$:
       i. Compute $\Delta_2 = f_6(x) \oplus f_6(x \boxminus \Delta_1)$.
       ii. Add $x$ to the entry $T_6[\Delta_1, \Delta_2]$.
```

Given T_5 and T_6, for fixed values of R_3^1 and R_3^2, the corresponding solutions (subkeys $K'_5, K'_6 = K_6, K'_7$) from the decryption side are computed according to the following algorithm, which is an adaptation of the previous attack (described in Section A.2).

```
Decryption Side Algorithm

1. For each 40-bit value of $K'_7$, the 5-bit value of $K_{r_5}$ and the 5-bit value of $K_{r_6}$:
   (a) Partially decrypt $(L'^1_7, R'^1_7)$ and $(L'^2_7, R'^2_7)$ (as defined in the previous attack, described in Section A.2) through round 7, and obtain suggestions for $R'^1_5$, $R'^2_5$.
   (b) Invert the output values of $f_5$ by computing suggestions for $x_5^1 \triangleq T_5[R'^1_5 \oplus R_3^1]$ and $x_5^2 \triangleq T_5[R'^2_5 \oplus R_3^2]$.
   (c) Compute $R'^1_4 \oplus R'^2_4 = (x_5^1 \ggg K_{r_5}) \oplus (x_5^2 \ggg K_{r_5})$, and compute $\Delta_2 \triangleq (R'^1_4 \oplus R'^2_4) \oplus (R'^1_6 \oplus R'^2_6)$.
   (d) Compute $\Delta_1 \triangleq (R'^1_5 \boxminus R'^2_5) \lll K_{r_6}$.
   (e) Find all $x_6$ which satisfy the differential transition $[\Delta_1, \Delta_2]$ from $T_6$, and compute a suggestion for $K_{m_6} = R'^1_5 \boxminus (x_6 \ggg K_{r_6})$.
   (f) Partially decrypt $(L'^1_6, R'^1_6)$ through round 6, compute $R'^1_4$, and obtain a suggestion for $K'_{m_5} = R'^1_4 \oplus (x_5^1 \ggg K_{r_5})$.
```

The algorithm iterates over the $40 + 5 + 5 = 50$ key bits, and for each iterations performs a constant number of operations (on average). Thus, the time complexity of the algorithm is about 2^{50}.

Encryption Side The enumeration algorithm for the encryption side requires preprocessing the two key-less functions f_2 and f_3. For f_3 (similarly to f_5 from the decryption side), we prepare a table, T_3, which allows to invert it efficiently, namely, given a 32-bit word y, $T_3[y]$ contains all the words x such that $f_5(x) = y$.

For f_2, we also prepare a table, T_2, but its computation depends on the plaintexts (L_0^i, R_0^i) (more precisely, it depends on R_0^1 and R_0^2), and cannot be performed in preprocessing as the computation of T_6 (from the decryption side). The reason for this complication is that the third round function mixes K_{m_3} into the state via modular subtraction and we cannot easily compute the XOR output difference of f_2 , whereas K'_{m_5} is mixed via XOR from the decryption side (which allows us to compute the XOR difference at the output of f_6). However, we note that the computation time for preparing T_2 is still negligible compared to the time complexity of the full attack.

Similarly to T_5, the entries of T_2 are sorted according to two 32-bit words $\varDelta_1, \varDelta_2$. The entry $T_2[\varDelta_1, \varDelta_2]$ contains all the words x such that $(f_2(x) \oplus R_0^1) \boxminus (f_2(x \oplus \varDelta_1) \oplus R_0^2) = \varDelta_2$. Given R_0^1, R_0^2, on average, there exists one 32-bit word x that satisfies the 32-bit condition imposed by $\varDelta_1, \varDelta_2$. Thus, T_2 requires 2^{64} words of memory, and can be computed in time complexity 2^{64} according to the preprocessing algorithm below.

$\mathbf{T_2}$ Computation

```
1. For each value of $\varDelta_1$:
   (a) For each value of $x$:
       i. Compute $\varDelta_2 = (f_2(x) \oplus R_0^1) \boxminus (f_2(x \oplus \varDelta_1) \oplus R_0^2)$.
       ii. Add $x$ to the entry $T_2[\varDelta_1, \varDelta_2]$.
```

Given (L_0^1, R_0^1) and (L_0^2, R_0^2), T_2 and T_3, for fixed values of R_3^1 and R_3^2, the corresponding solutions (subkeys K_1, K_2, K_3) from the encryption side are computed according to the following algorithm (which has a similar structure to the algorithm for the decryption side).

Encryption Side Algorithm

```
1. For each 37-bit value of $K_1$, the 5-bit value of $K_{r_2}$ and the 5-bit value
   of $K_{r_3}$:
   (a) Partially encrypt $(L_0^1, R_0^1)$ and $(L_0^2, R_0^2)$ through round 1, and obtain
       suggestions for $R_1^1$, $R_1^2$.
   (b) Invert the output values of $f_3$ by computing suggestions for $x_3^1 \triangleq$
       $T_3[R_1^1 \oplus R_3^1]$ and $x_3^2 \triangleq T_3[R_1^2 \oplus R_3^2]$.
   (c) Compute $\varDelta_2 \triangleq R_2^1 \boxminus R_2^2 = (x_1 \ggg K_{r_3}) \boxminus (x_2 \ggg K_{r_3})$.
   (d) Compute $\varDelta_1 \triangleq (R_1^1 \oplus R_1^2) \lll K_{r_2}$.
   (e) Find all $x_2$ which satisfy the differential transition $[\varDelta_1, \varDelta_2]$ from
       $T_2$, and compute a suggestion for $K_{m_2} = R_1^1 \oplus (x_2 \ggg K_{r_2})$.
   (f) Partially encrypt $(L_1^1, R_1^1)$ through round 2, compute $R_2^1$, and obtain
       a suggestion for $K_{m_3} = R_2^1 \boxminus (x_3^1 \ggg K_{r_3})$.
```

The algorithm iterates over the $37 + 5 + 5 = 47$ key bits, and for each iterations performs a constant number of operations (on average). Thus, the time complexity of the algorithm is about 2^{47}.

The Full Algorithm We now describe the full dissection algorithm that uses the enumeration algorithms from the encryption and decryption sides.

Full Optimized Algorithm for 8-Round CAST-128

1. Compute the tables T_3, T_5, T_6, as described above.
2. Request the decryption of the 8 chosen ciphertexts (L_8^i, R_8^i), as in the previous attack [11] (described in Section A.2).
3. Given (L_0^1, R_0^1) and (L_0^2, R_0^2), compute the table T_2, as described above.
4. For each possible value of R_3^1 and R_3^2:
   (a) Apply the encryption side enumeration algorithm, retrieve suggestions for K_1, K_2, K_3, and use them to obtain suggestions for R_3^i by partially encrypting (L_0^i, R_0^i) (for $3 \leq i \leq 8$). Store the suggestions in a sorted List, *List*, next to the values of K_1, K_2, K_3.
   (b) Apply the decryption side enumeration algorithm, retrieve suggestions for K_5', K_6, K_7', and use them to obtain suggestions for R_3^i by partially decrypting (L_8^i, R_8^i) (for $3 \leq i \leq 8$). Search the suggestions in *List*, and for each match, retrieve K_1, K_2, K_3. Given $K_1, K_2, K_3, K_5', K_6, K_7'$, use simple key relations to retrieve a suggestion for the full key (as described in [11]) and test it using trial encryptions.

In Step 4, we iterate over 2^{64} values, and for each one, we perform about $2^{47} \cdot 8 = 2^{50}$ partial encryption operations in Step 4.(a) (about 2^{47} operations for each plaintext) and $2^{50} \cdot 8 = 2^{53}$ partial decryption operations in Step 4.(b). We have a total of 6 filtering conditions of 32-bits of the key suggestions in Step 4.(b), and thus we are expected to remain with $2^{47+50-6 \cdot 32} < 1$ key suggestions for each value of R_3^1 and R_3^2, and the complexity required for the trial encryptions is negligible. Therefore, the total time complexity of Step 4 is about $2^{64+53} = 2^{117}$, and since the time complexity of Steps 1 and 3 is about 2^{64}, the total time complexity of the attack is also about 2^{117} (which is close to the time complexity estimation of [11], as expected). Since *List* is expected to contain about 2^{47} values, the memory complexity of the attack is about 2^{64}, dominated by the tables T_2, T_6. Therefore, we reduce the memory complexity of the best known attack on CAST-128 [11] from 2^{111} to 2^{64}, without increasing its time nor its data complexities.

We note that it is possible to slightly reduce the time complexity of the attack by using only 4 plaintext-ciphertext pairs to compute filtering conditions in Steps 4.(a) and 4.(b) (and performing more trial encryptions, or alternatively, verifying that the suggested keys satisfy the constraints imposed by the key schedule of CAST-128).

B On the Invertibility of the Round Function

We first note that our 5-round only needs two round functions to be efficiently invertible — namely, $F_2(\cdot)$ and $F_4(\cdot)$. As noted before, we are not aware of any Feistel cipher which does not posses this property. Even DEAL [15], whose round functions are full DES encryptions, has invertible round functions given the subkey, as our attack described in Section 4.1 shows.

Moreover, we can relax a bit the requirement over the invertibility of the round functions $F_2(\cdot)$ and $F_4(\cdot)$. We remind the reader that we are allowed $2^{0.5n}$ memory, which can help in inverting the round functions. For example, if the cipher is a Feistel-2 structure (i.e., the round function is $F_i(K_i, I_i) = G_i(K_i \oplus I_i)$, for some completely one-way function $G_i(\cdot)$), a simple enumeration of all input/output pairs of $G_i(\cdot)$ is sufficient to invert the round function.

Finally, we note that when we discuss the general Feistel-2 structure, the memory complexity can be slightly reduced, as no memory is needed for the meet in the middle step in itself. For example, instead of structure T_1, given O_2^1, we invert G_2, to obtain $I_2^1 \oplus K_2$. Hence, for any K_1 value, it is possible to immediately obtain the corresponding K_2.

C Comparison of Results on DEAL

Key	Rounds	Complexity			Attack
Size		Time	Memory	Data	
192	6	2^{121}	2^{64}	2^{70} CP	Impossible differential [15]
	6	2^{144}	2^{144}	2^{32} CP	Splice-and-cut[10] [11]
	6	2^{144}	2^{88}	2^{32} CP	**New** (Section 3.1)
256	8	2^{224}	2^{168}	4 KP	Meet in the Middle
	8	2^{200}	2^{200}	2^{32} CP	Splice-and-cut[10] [11]
	8	2^{200}	2^{144}	2^{32} CP	**New** (Section 3.1)

KP — Known plaintext, CP — Chosen plaintext

Table 2. Comparison of Results against DEAL

[10] This attack was not really suggested in [11], but can be derived from the paper.