The State-Test Technique on Differential Attacks: a 26-Round Attack on Craft and Other Applications

Dounia M'Foukh, María Naya-Plasencia, and Patrick Neumann

Inria, France
{dounia.mfoukh,maria.naya_plasencia,patrick.neumann}@inria.fr

Abstract. The state-test technique, originally introduced in the context of impossible-differential cryptanalysis and recently used as an improvement for truncated-differential Meet-in-the-Middle attacks, has proven to be useful for reducing the complexity of attacks. In essence, the idea is to guess parts of the state instead of the key during the key-guessing stage of an attack, ultimately reducing the number of guesses needed.
We generalize the idea of the state-test technique, allowing it to be applied not only to impossible-differential and (truncated-)differential Meet-in-the-Middle, but also to differential and differential-linear cryptanalysis, proposing also a new performant technique exploiting the state-test technique and probabilistic key-recovery. Additionally, we provide insights on the interaction between cipher and difference needed for the state-test technique to be applicable, finding it to be a promising option for many ciphers.
To illustrate our findings, we provide 3 new applications of the state-test technique: we show how it can be used to improve the best known attack on the block cipher Pride, how it can be used to improve a step in the best known attack on Serpent, and use it to present the first known attacks on 24, 25 and 26 rounds of Craft (out of 32), improving by up to three rounds over the previous best ones.

Keywords: key-recovery attack · state-test technique · differential attacks · differential MitM · differential-linear · impossible differential · Craft · Pride.

1 Introduction

The trust we have on symmetric primitives is mainly given by cryptanalysis: The more we analyze a primitive without finding weaknesses, the more trust we will have in it. In order to be able to predict how far a primitive is from being broken, it is very important to keep its security margin up to date: we try to find the most efficient attacks on the highest possible number of rounds. This allows us to estimate how far we are from improving this attack and from reaching more rounds. Having an accurate estimation of the security margin is essential. Once an attack is proposed, any further improvements that appear help to refine it and to have a better notion of the current security of the primitive.

Unhappily, the state of the art of symmetric cryptanalysis is kind of chaotic, as a great number of new families, variants, improvements and ad-hoc attacks appeared in the last two decades due to cryptographic competitions. Today, it is not possible to easily test the resistance of a given primitive against all known attacks and all the related optimizations that have appeared: most of the optimizations are proposed in specific scenarios, which makes it very hard to grasp the nature and potential of the improvement.

In this paper, we aim at generalizing and clarifying new potential uses of the state-test technique. This idea was originally introduced by Boura *et al* in [10] in the particular context of impossible-differential attacks but a similar idea was used in [15,17] in the Meet-in-the-Middle context, and it was recently used by Ahmadian *et al* in [1] in the context of (truncated-)differential Meet-in-the-Middle attacks. In essence, this technique allows to reduce the overall number of guesses by guessing internal state values instead of the involved key bits. This allows, depending on the cases, to identify a unique partition of the key bits (impossible-differential attacks) or to recover non-linear equations involving some key bits and, therefore, a system of equations to solve in the end. This technique also seems related to guess-and-determine attacks, that were introduced in the context of stream ciphers [25], and also applied to hash functions and block ciphers, like in [23,18]. As in the state-test technique, cells of the state

Cipher	Rounds	Time	Memory	Data	Attack	Ref.
Craft	21	$2^{106.53}$	2^{100}	$2^{60.99}$	ID	[19]
	21	2^{116}	2^{68}	2^{56}	TD-MitM	[1]
	22	2^{125}	2^{72}	2^{58}	TD-MitM	[1]
	23	2^{125}	2^{101}	2^{60}	TD-MitM	[1]
	23	$2^{111.46}$	2^{120}	$2^{60.99}$	D	[26]
	23	2^{109}	2^{36}	2^{58}	TD-MitM	Sec. 4
	24	2^{110}	2^{34}	2^{60}	TD-MitM	Sec. 4
	25	$2^{117.58}$	2^{48}	2^{60}	TD-MitM	Sec. 4
	26	2^{118}	2^{34}	2^{64}	TD-MitM	Sec. 4
Pride	18	$2^{63.3}$	2^{35}	2^{61}	D	[21]
	18	$2^{57.83}$	2^{35}	2^{61}	D	Sec. 5
	18	$2^{65.86}$	2^{34}	$2^{60.17}$	D	Sec. 5
Serpent	12	$2^{233.55}$	$2^{127.92}$	$2^{127.92}$	CP DL	[11]
	12	$2^{242.93}$	$2^{118.40}$	$2^{118.40}$	CP DL	[11]
	12	$2^{214.36}$	$2^{125.16}$	$2^{125.16}$	KP L	[16]
	12	$2^{210.36}$	$2^{126.30}$	$2^{126.30}$	KP L	[16]
	12	$2^{233.55}$	$2^{127.92}$	$2^{127.92}$	CP DL	Sec. 6
	12	2^{240}	$2^{118.40}$	$2^{118.40}$	CP DL	Sec. 6

D	Differential	TD-MitM	Truncated Differential MitM
DL	Differential-Linear	ID	Impossible Differential
L	Linear	KP	Known Plaintext
CP	Chosen Plaintext		

Table 1: Summary of best known attacks on Craft, Serpent and Pride

are guessed, and then verified to be coherent with the key. We push forward the application of the state-test technique by first giving insights for which ciphers it could be most useful (Section 3.1), enabling us to identify ciphers in which the state-test technique leads to improved attacks. Additionally, we propose three new application scenarios: an improvement of differential MitM attacks by better exploiting the state-test technique and probabilistic key-recovery (enabling a *0-round distinguisher*), two separated scenarios in classical differential attacks, and differential-linear attacks (Section 3.2).

Finally, we propose several new applications, that are summarized in Table 1. In particular, thanks to the 0-round technique we are able to build for the first time a 2-round structure on Craft, which is additionally the first time a 2-round structure for a differential MitM attack is build for a construction having a key-addition in the whole state. This allows to improves the best known attacks on Craft [5] by three rounds (Section 4).

We also improve the best known attack by Lallemand and Rasoolzadeh [21] on Pride [2] (Section 5), and we show how a step of the best attack on Serpent from Broll *et al* [12] could be done more efficiently, which allows to slightly improve the time complexity of the attack needing the lower data complexity on the highest number of rounds (Section 6).

2 Preliminaries

Notation Besides standard notation, we use $|A|$ not only to denote the cardinality of the set A, but also $|a|$ to denote the length of the vector a. The concrete meaning will be clear from the context.

2.1 Description of Ciphers

Let us briefly describe the two main ciphers used throughout this work in our applications: Craft and Pride. The block cipher Serpent is also used, but as we present its detailed application in Supplementary Material B, we also introduce the cipher in that same section.

Craft Craft [5] is a 64-bit tweakable AES-like block cipher using a 128-bit key. Its state is represented as a 4×4 matrix of 4-bit values, and its round function consists of a column wise linear transformation, referred to by Mix-Columns (MC), the addition of a round constant and a round (tweak-)key, followed by a permutation of the cells (PN) and a cell-wise application of the s-box (SB). More specifically, MC corresponds to the matrix

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

and PN to the permutation $[15, 12, 13, 14,\ 10, 9, 8, 11,\ 6, 5, 4, 7,\ 1, 2, 3, 0]$. Both these transformations are involutions. This function is iterated 32 times with different round constants and round keys, but the last round ends after the addition of the round constants. As we consider the tweak to be zero, the round keys are derived by splitting the 128-bit key into two 64-bit values K_0 and K_1 and alternating between them. More details can be found in [5].

Pride Pride is a 20-round lightweight block cipher introduced at Crypto 2014 in [2]. The cipher operates on block of size 64-bit with a 128-bit key $k = k_0||k_1$. One half of the key, k_0 is used as a pre-whitening and post-whitening key and we derive 19 subkey from the other half k_1. For the 19 first rounds, the round function consists of the following operations : A subkey addition; an s-box layer applying an s-box on each nibble of the state; and a Linear layer, consisting in the application of a bit permutations P, then the application of matrices and the application of P^{-1}.

The subkey K_i in round i, $1 \leq i \leq 20$ is given by $K_i = P^{-1}(f_i(k_1))$, with $f_i(k_1)$ is:

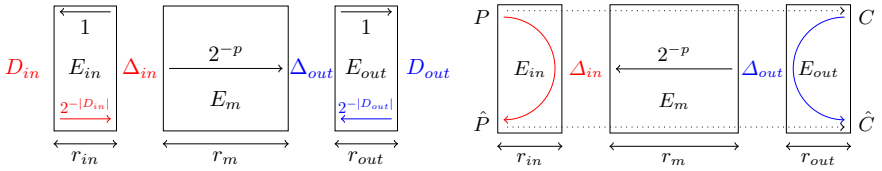
$f(k_1) = k_{1_0}||g_i^{(0)}(k_{1_1})||k_{1_2}||g_i^{(1)}(k_{1_3})||k_{1_4}||g_i^{(2)}(k_{1_5})||k_{1_6}||g_i^{(3)}(k_{1_7})$,
where k_{1_i} is byte number i of k_1 and the g_i functions are given by:

$$g_i^{(0)}(x) = (x + 193i) \bmod 256, \qquad g_i^{(1)}(x) = (x + 165i) \bmod 256$$
$$g_i^{(2)}(x) = (x + 81i) \bmod 256, \qquad g_i^{(3)}(x) = (x + 197i) \bmod 256$$

2.2 Brief Description of Differential Families of Attacks

In the following section, we consider an n-bit cipher E decomposed into three sub-ciphers $E_{out} \circ E_m \circ E_{in}$ of r_{in}, r_m and r_{out} rounds respectively. We denote by K the key of size k-bit. Further, we consider a (truncated) differential distinguisher $\Delta_{in} \rightarrow \Delta_{out}$ covering the r_m middle rounds of probability 2^{-p} for differential and differential Meet-in-the-Middle cryptanalysis, and of probability 0 for impossible-differential cryptanalysis. Then, we extend the differential distinguisher for r_{in} rounds backward and for r_{out} rounds forward with probability 1. This propagation implies two sets of differences D_{in} and D_{out} for the plaintexts and ciphertexts differences respectively. We use the same simplified cost analysis than the attacks we are comparing to in the applications section.

Differential Attacks Differential cryptanalysis, introduced in 1990 by Biham and Shamir [7], is one of the most well known family of symmetric attacks and has been used to study the resistance of many ciphers since then. The goal of differential cryptanalysis is to exploit the high probability of the propagation of specific differences through some rounds of the cipher. The scheme of the attack is summarized on Fig. 1a. In this paper, we will refer to a bit with a non-zero difference as an active bit. Then a differential key recovery consists of mainly three steps :



(a) Differential (b) (Truncated-)differential MitM

Fig. 1: Scheme of attacks

- **Pair generation:** we build structures consisting of sets of plaintexts having their differences in D_{in}. To build them, we fix the inactive bits to a certain value and take all the possible values for the bits in D_{in}. The number of pairs we compute is such that the right key is suggested about a times.
- **Pair sieving:** we only keep the pairs such that their corresponding ciphertexts have a difference in D_{out}.
- **Key guessing phase:** we want to generate candidate tuples $(P,\hat{P},C,\hat{C},k_{in} \cup k_{out})$ that imply the differences Δ_{in} and Δ_{out}. To find those candidates, we make some key guesses k_{in} and k_{out} and verify if the transition through the active s-boxes is possible. Then we can either directly perform an exhaustive search for each candidates or first use a counter to keep only the candidates that verify a threshold, as we detail later.

(Truncate-)Differential MitM Attacks The differential Meet-in-the-Middle (MitM) technique, introduced in [8], is a new type of cryptanalysis combining two significant families of symmetric attacks: the Meet-in-the-Middle attack and the differential attack. The core idea of the attack involves employing the MitM approach to compute the pairs $(P, C, \hat{P}, \hat{C})$ verifying the differential distinguisher. In other words, given a pair of plaintext/ciphertext $(P,\ C)$, we independently compute the candidate plaintext and ciphertext $\hat{P}$ and $\hat{C}$ such that they imply the input/output differences of the distinguisher as summarized on Fig. 1b. $\hat{P}$ is computed from the plaintext P and the difference Δ_{in} for each possible value of the associated key k_{in}, and $\hat{C}$ from the ciphertext C and the difference Δ_{out} for each possible value of the associated key k_{out}.

We have to repeat the procedure 2^p times using 2^p different pairs of plaintext/ciphertext so that we can expect to find a pair satisfying the distinguisher.

Complexity In the case that we need to guess the remaining bits of the master key, specifically if $|k_{in}|+|k_{out}|-n-|k_{in} \cap k_{out}|+p > 0$, the total time complexity would be:

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

where $c_{in} = 2^{|k_{in}|-|k_{in} \cap k_{out}|}$ and $c_{out} = 2^{|k_{out}|-|k_{in} \cap k_{out}|}$.

The data complexity is $\mathcal{D} = \min(2^n, 2^{p+\min(|k_{in}|,|k_{out}|)})$, and the memory complexity is $\mathcal{M} = \min(c_{in}, c_{out})$.

Truncated-Differential MitM Attacks Differential MitM attacks have later been extended to truncated-differential MitM attacks [1]. The general idea is the same, but the computation of $\hat{P}$ and $\hat{C}$ is done for all differences contained in the corresponding set of differences. With this, the time complexity becomes

$$\mathcal{T} = 2^{p-|\delta_{in}|+|k_{in}\cap k_{out}|}\left(c_{in} + c_{out} + c_{in}\cdot c_{out}\cdot 2^{-n}\right) + 2^{k-n+p},$$

$$\text{where} \qquad c_{in} = 2^{|\delta_{in}|+|k_{in}|-|k_{in}\cap k_{out}|} \quad \text{and} \quad c_{out} = 2^{|\delta_{out}|+|k_{out}|-|k_{in}\cap k_{out}|}.$$

The data complexity is $\mathcal{D} = \min(2^n, 2^{p+\min(|k_{in}|+|\delta_{in}|,|k_{out}|+|\delta_{out}|)})$, and the memory complexity is $\mathcal{M} = \min(c_{in}, c_{out})$.

Impossible-Differential Attacks Impossible-differential was independently introduced in [20] and [7] in 1998 and 1999 respectively. As in differential attacks, the idea is to exploit a statistical property of the propagation of specific differences thorough some rounds of the cipher. But contrary to classical differential cryptanalysis, the differential is of probability zero.

We denote by c_{in}, k_{in} and c_{out}, k_{out}, the number of bit-conditions to be verified and the key bits involved to obtain Δ_{in} and Δ_{out} respectively. Thus the probability that for a given key, a pair of plaintext/ciphertext in D_{in} and D_{out} satisfy the $c_{in} + c_{out}$ conditions is $2^{-(c_{in}+c_{out})}$. Since the probability for a trial key to be kept using N different pairs is $P = (1 - 2^{-(c_{in}+c_{out})})^N$, N should be chosen such that $N \leq 2^{c_{in}+c_{out}}$. Furthermore, the naive approach is to test for each candidate key if one of the N pairs lead to the impossible differential and if it is the case we discard the candidate key. Then we test the expected $2^k P$ candidates key left with a new plaintext/ciphertext pair. Thus the time complexity is $\mathcal{T} = (C_N + (N2^{|k_{in}\cup k_{out}|})C'_E + 2^k P)C_E$, where C_E is the cost of an encryption, C_N is the cost in data to compute the N pairs and C'_E is the ratio of the cost of partial encryption to the full encryption. The second term of the complexity correspond to the cost of filtering the candidate keys.

2.3 State-Test Technique History

In [10], the state-test technique was introduced in the context of impossible-differential attacks. It was applied in particular to two Feistel ciphers: Clefia [24] and Camellia [3]. The technique allowed to reduce the number of total involved key bits to guess in the key-recovery phase. In order to do that, the authors proposed guessing the value of a part of the state, of size s bits, smaller than the size of the involved key bits in that state (denoted by k_{st}). If $s < k_{st}$ the time complexity can be potentially reduced. For the impossible-differential to work, the authors needed to fix the values of some plaintext bits, in order to associate each guess of the state to a unique partition of the candidate keys: in an impossible-differential attack, this was the only way information on the key could be discarded so that the attack worked [1]. Later, in a paper improving differential

[1] Otherwise the discarded key belongs to a different set each time, where the sets are non-disjoint, and this does not allow to efficiently combine the information for only keeping the potentially correct keys.

MitM attacks [1], the authors proposed to use the technique to improve some attacks. The application here seemed more natural and easier, given that in these applications, the guessed states did not need to determine disjoint partitions, but just implied additional information on the secret key bits in the form of a non-linear equation, and the challenge became to efficiently use this information by solving a system of equations to recover the whole key in the end.

Impossible Differential The state-test technique was first introduced in [10] in the context of impossible-differential key-recovery attacks. Thanks to this technique the authors managed to find some of the best known attacks at the time on several Feistel ciphers such as Clefia-128 [24] or the different instances of the cipher Camellia [3].

The idea in this context is to fix the bits of the plaintext involved to compute the part of the internal state we are interested in, so that it will create a partition of the involved key bits depending only in the value of the guessed internal state. This way, by guessing s bits of the internal state and fixing the involved plaintext part, the size of the key information guessed is reduced by s bits. Indeed, if X is an s-bit guess of the internal state using the state-test technique then X can be written as an equation over the (still) unknown key bits and depending on plaintext bits and known key bits. We can rewrite this equation by putting on the left side, the terms that we can compute and on the right side the terms we can not compute, depending non linearly on the unknown key bits and the plaintext bits then if for a fixed value of the already guessed key candidate, the left part of the equation can take all the possible 2^s values that means that any choice of the unknown key bits implies the impossible differential and thus can be discarded. Hence the technique can be useful to decrease the time complexity of the filtering phase since the size of the internal state needed is often smaller than the number of the key bits it depends on. The downside here is that since we fix part of the plaintexts, the data available to perform the attack is smaller.

Differential MitM The state-test technique has also been used in differential Meet-in-the-Middle attacks: in [1], the authors applied the state-test technique on two SPN ciphers Skinny [4] and Craft [5].

The technique was used to reduce the number of candidates by lowering the number of key information needed to know if a pair follows the differential path. The idea here seems more natural than in the impossible-differential case since we don't need to fix part of the plaintexts. Instead, for each set of partial candidates formed by tuples of data and key information and for the guess of s bits of internal state, we try all the 2^s possibilities for the internal state which gives 2^s times more candidates. Thus if the internal state depends on more than s bits of the key then the number of candidates in a set is reduced. The difficulty with this method can arise in the last step of the attack, when we want to recover the full key with the equations given by the guess of internal states, merging both partial sets of solutions including to efficiently solving a system of equations given by the values of the guess of the internal state. This can be challenging if the

number of final candidates is close to the full exhaustive search. In the case of CRAFT, the authors used a precomputation table to store the solutions of some of the equations. Another possible limitation seen in the case of SKINNY for example, is that the state-test technique might allow less sieving regarding the linear relations between the subkey bits given by the upper and lower parts of the attack and by the key schedule to reduce the number of candidates, which could be a problem during the matching of candidates of the upper and lower parts in differential Meet-in-the-Middle cryptanalysis.

3 New Insights and Applications for the State-Test Technique

In this section we will identify and define, for the first time, on which type of ciphers and situations the state-test can potentially improve the time complexity of the attacks. We will also introduce three new applications of the state-test technique: an interesting extension of differential MitM attacks thanks to state-test and probabilistic extension, the case of classical differential attacks and differential-linear attacks. Thought it seems very difficult to provide a generic algorithmic framework, we will present the particularities of each scenario, and we will see later in the applications some examples of concrete scenarios and attacks. In the classical differential attacks case, we will also discuss the different settings that can arise and how to deal with the state-test equations in each.

3.1 Ciphers on which the State-Test Technique Can Improve the Attacks

On a high level, the state-test technique can be applied whenever a part of some internal state required to be known in the key-guessing phase of an attack depends on a bigger part of the key that would not be guessed otherwise. This is quite broad and in order to say more about the applicability of this technique we need to specify some details of the attack.

Hence, let us assume that given a pair of data, we want to detect the keys that would imply that a certain differential occurs at some point during the encryption process. Without loss of generality, we assume that we start from the plaintext side.

For simplicity, let us assume that, given a pair of inputs (X_0, X_1), the key-guessing part requires us to verify if a difference δ appears after two rounds of an SPN, as depicted in Fig. 2. First, we propagate δ backwards, recording the differences that can appear at each step in the sets Δ_Y, Δ_Z, see Fig. 2. Obviously, $\Delta_Y = L^{-1}(\Delta_Z)$. Next, we want to know the difference $Y_0 + Y_1 = \overline{S}_0(X_0 + K_0) + \overline{S}_0(X_1 + K_0)$. But this equation is equivalent to

$$X_0 + X_1 = \overline{S}_0^{-1}(\overline{S}_0(X_0 + K_0) + Y_0 + Y_1) + X_0 + K_0,$$

where the left-hand side is known. In many cases, the right-hand side does not depend on the full key K_0 if we fix the difference $Y_0 + Y_1$ to be contained in

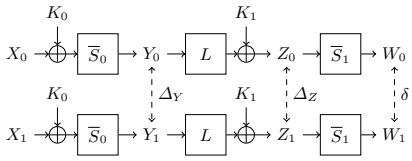


Fig. 2: Verifying a difference δ after two rounds of an SPN. The $\overline{S}_i$ are s-box layers, while L is a linear layer and $\Delta_X, \Delta_Y, \Delta_Z$ are a probability-one truncated differential backwards extension of δ

Δ_Y (so that δ holds).[2] Since the part of the key K_0 it depends on might include linear combinations and not only include direct bits of K_0, let us denote the information needed by the set of masks Γ_X, such that the inner product with these masks corresponds to those linear combinations. Hence, after guessing the bits of K_0 defined by Γ_X we are either able to compute the difference $Y_0 + Y_1$, or can conclude that $Y_0 + Y_1 \notin \Delta_Y$, discarding the pair. Let us assume that $Y_0 + Y_1 \in \Delta_Y$. Knowing this difference trivially implies knowing $Z_0 + Z_1$, and we can similarly compute the masks Γ_Z that correspond to the bits of Z_0 (and therefore K_1) that need to be known to verify that $W_0 + W_1 = \delta$. To see which information of Y_0 those masks translate to, we simply apply the transpose of L, i.e., $\Gamma_Y = L^T(\Gamma_Z)$ defines the bits of Y_0 we need to know. We now need to traverse these masks through the s-box layer $\overline{S}_0$,[3] and denote the corresponding masks by Γ'_X. If $\Gamma'_X \not\subseteq \Gamma_X$, meaning that we need additional information from K_0 in order to verify $W_0 + W_1 = \delta$ (compared to computing the difference $Y_0 + Y_1$), we can (potentially) apply the state-test technique in order to reduce the number of overall guesses needed.

Note that it is enough to consider two consecutive rounds at a time, as $\Gamma'_X \subseteq \Gamma_X$ is invariant under traversing both sets through additional rounds. While this shows that the state-test technique has a great potential to be applied to many ciphers, the discussion also shows that if a cipher uses an orthogonal linear layer L, i.e., $L^{-1} = L^T$, and if the whole word of each active s-box needs to be known in order to verify the associated difference,[4] the state-test technique cannot be applied in this setting. However, the probabilistic key-recovery technique might be a way around this, as we discuss below.

[2] There is a straight-forward relation of this to the boomerang-connectivity table [13]: if the entries at (α, δ) are 2^n for all $\delta \in \Delta_Y$ the equation above is invariant under the addition of α to the key K_0, meaning that we do not need to know the associated bit. The easiest instance of this is when some s-boxes are differentially inactive, causing all such BCT entries to be 2^n if α is active in those s-boxes only.

[3] This could be done using linear structures, where the difference again corresponds to the bit that has no (or only a liner) influence on Y_0.

[4] If this is the case for all differences, it is equivalent to the boomerang uniformity [13] being lower than 2^n.

Feistel ciphers While the discussion above is focused on SPNs, the state-test technique can similarly apply to Feistel ciphers as well: the arguments above easily translate if the non-linear part of the Feistel cipher applies multiple s-boxes in parallel. Indeed, if we need to know the input bits of an s-box that depend on the output bits of some s-boxes round(s) before that were not needed otherwise then we are once more in the a scenario where the state-test technique might help to improve the time complexity of the attack.

On probabilistic key-recoveries In the case where the differential is not propagated with probability one, but in a probabilistic manner, as in [9,1], we artificially restrict the differences in the set Δ_Z, reducing the size of the set Γ_X in the process. Aside from that, the same argument can be applied. Note that this can be a way to apply the state-test technique to a cipher where it is not possible otherwise.

3.2 New Applications of the State-Test Technique

In this subsection, we present three new scenarios of application for the state-test techniques.

Enabling 0-Round Distinguishers As we will see in Section 4, combining the state-test technique with the probabilistic key recovery technique [9,1,26], where a fraction of 2^{-p} initial candidates will survive the key-guessing part only, can allow us to reduce the number of rounds r_m covered by the distinguisher to zero without increasing the number of candidates. In essence, we show that for the attack on Craft in [1] the amount of state-test guesses is in equilibrium to the survival rate (i.e., $s = p$), allowing us to convert any number of rounds of the distinguisher (r_m) into rounds covered by key/state-test guesses (r_{in} or r_{out}). While this means that we can get more state-test equations for free compared to [1], the equations tend to get more complex the deeper we go into the encryption/decryption process. It is therefore unclear if those additional equations can help to recover the full key efficiently in the last step of the attack. The most interesting part comes if we are able to reduce r_m to 0: in this case we are able to partially compute the same parts of the middle states from both sides, which we can use for additional filtering during the matching phase. We use these idea in Section 4, and combining this observation with additional extensions and optimizations allows us to attack up to three more rounds than previous attacks.

Classical Differential The state-test technique has not been used in the context of classical differential attacks until our present results. There are mainly two possible scenarios, whether the attack needs to use counters or not. It is worth pointing out that the application on the scenario not needing a counter has similarities with the application on differential MitM attacks, and the one with counter has similarities with the impossible-differential scenario:

Without needing a counter. The first scenario considers attacks that do not need a counter for key candidates, as the final number of tuple-candidates determining a part of the key is small enough to fully determine the whole key efficiently. In this case, the procedure to recover the information is more straightforward and similar to the scenario of differential MitM attacks. Here, each word guessed by the state-test technique can be seen as one word of information on the key. Thus, in the end of the filtering phase, the set of candidates is smaller than the exhaustive search of all the information bits we have recovered.

Note here that we do not need to fix any plaintext bits in order to form partitions of the key space (as it is the case on impossible-differential attacks), since here the information on the key can be directly exploited. The same happened with differential MitM attacks: though there were groups of triplets with a common plaintext, this is not needed nor exploited in the attack, and the equations can just be treated as independent information on potential candidates.

The aim of the final step will be to find a way of recovering the whole key using all this information, which implies efficiently solving the system of equations given by the state-tests while having a minimum impact on the time complexity (and for this a priori we do not need to fix any values in the plaintexts). In Section 3.3, we discuss a bit more how to solve these equations.

With a counter. In the second scenario, we need a counter to perform the attack. This is the case, for example, when the set of candidates we recover is not smaller than the number of key bits we have determined or when the limit for the security of the cipher is the product of the data and time complexities and thus we can not test all the remaining candidates.

This situation is a bit similar to the impossible-differential attacks as we might fix part of the plaintexts to generate a partition of the key, that can be included in the counter. Indeed by fixing the same bits to a given value for all of the plaintexts used the attack, the guess of the s-bit internal state X partition the set of the involved key bits into disjoint sets. Thus a wrong guess of X should appears with probability 2^{-s} and the counter of the associated key candidate will be incremented less often than for a good guess of X. However, we need to be careful as we need to take into account the following problems:

1. Firstly the equations without fixing bits can not be used in the counter as they do not generate a partition. But those equations can still be useful since they form an over defined system of equations which has a solution for the good guess of the key and no solution otherwise. In subsection 5, we use this method to propose an improved attack on 18 rounds of Pride.
2. Secondly, in some cases, for the good candidate key, the guess of the internal state might have given as candidate a pair that seems to follow the differential path but induce a wrong guess of the unknown key bits involved in the equation. This is to be carefully verified when it would introduce a wrong equation in the correct system (correct guess for the information on the counter). We need to carefully process those noise equations. This is also considered in subsection 5.

Differential-Linear In this scenario, the techniques apply the same way as for differential attacks. See Section 6 for an example.

3.3 Solving the State-test Equations

Whenever we make a state-test guess, this guess defines an equation in plain- and ciphertext, as well as in the key. Hence, for a tuple of partial key guesses, state-test guesses and associated plain- and ciphertexts, we can use such equations to either find inconsistencies, allowing to filter out such tuples, or to solve the system for the still unknown part of the key. If there are a few unknowns only, solving them can be done by a simple guess-and-determine strategy. However, the deeper into the cipher the state-test guess is performed, the more complex the equations tend to get.

One approach to solve such more complex equations has been recently proposed in the context of differential MitM attacks [1]: they recovered 2 sets of candidates for some bits of the key and associated state-test equations, that needed to be merged efficiently in order to recover a unique set of key candidates that could be efficiently tested. Several lists merging techniques, like the ones from [22] combined with clever partial parallel guessing of some of the variables allowed to reduce the complexity. Also, if multiple candidates lead to the same equations, it is possible to precompute the solutions of those equations once, but the solution will work for multiple candidates at the same time, ultimately reducing the solving cost. This is also done in the 23-round attack on Craft [1]: the candidates are grouped by the equations they define, meaning that the equations need to be solved only once for the whole group. The equations are then solved individually, yielding lists of solutions, which are then merged to get solutions that satisfy all equations at once.

While this is certainly not the only possible approach – for instance, a substitution-based approach seems to produce promising results as well – we seem to lack generic tools to make full use of the state-test equation's potential. This will become especially apparent in our attacks on Craft: despite having around 20 such equations we make (efficient) use of at most 6 of them. While we don't expect equations for a guess deep inside a cipher to be easily solvable, it would still be interesting to see by how much better solving techniques could improve existing attacks, or enable new ones in the first place.

4 From 23 to up to 26 Rounds of Craft

Considering a differential MitM attack and using the *0-round distinguisher* technique presented in the previous section, we obtain a filter that allows us to add, for the first time, a two-round structure on Craft. Thanks to this, we here present the first attacks against 25 and 26 rounds of Craft (whose specifications are given in Section 2.1). It is worth noting that, due to the concrete construction of Craft, the 0-round distinguisher technique works particularly well, as in each internal round only one new word needs to be guessed to verify

the differential path, and the probability of verifying the path compensates this guess, which is an ideal scenario for freely adding the additional filtering.

In essence, both attacks are based on the 23-round *truncated-differential Meet-in-the-Middle* attack presented at EUROCRYPT'24 [1], which is among the best known ones at the time of writing, and use at the core the same iterative truncated differential distinguisher represented in Fig. 3. The previous attack on 23 rounds combined an 11-round differential with 6 upper and 5 lower probabilistic extension rounds, plus a final round covered by an initial structure. In our attacks, we will have a 0-round distinguisher, 2 final rounds covered by an initial structure that we will detail here, and the remaining ones associated to probabilistic key-recovery.

The overall procedure is similar for both attacks. We start by explaining the 25-round attack in detail and then specify the differences of the 26-round one.

4.1 An Attack on 25 Rounds of CRAFT

As the 23-round attack from [1] uses a truncated differential characteristic of alternating truncated differences, we can trivially extend this characteristic by one additional round, see Fig. 3. As in the previous attack, we apply the *state-test* technique to reduce the amount of key material k_{in} and k_{out} that needs to be guessed when computing $\hat{P}$ and $\hat{C}$, respectively. We will denote by s_{in} the amount of state-test bits guessed in the upper part, and by s_{out} in the lower part. Additionally, we apply a *probabilistic key-recovery* technique similar to [1], meaning that our computations of $\hat{P}$ and $\hat{C}$ require some internal differences to be zero, which happens with probability $2^{-p_{in}}$ and $2^{-p_{out}}$, respectively, but allows us to have less key material involved.

In fact, the *probabilistic key-recovery* technique forces the differences that are traced during the key-guessing steps to correspond to the truncated differences in the characteristic of the distinguisher. Additionally, the probabilistic transition in the key-guessing step happens with probability 2^{-4} for each round where it is applied, while each round where the state-test technique is used leads to guessing a 4-bits value. Hence, we can convert an arbitrary number of rounds of the distinguisher into key-guessing rounds, which decreases the probability of the key-guessing part by a factor of 2^4, while increasing the probability of the distinguisher by the same factor, but also requires an additional 4-bit of state-test guesses. As a result, the number of candidates stays the same as can be seen in the time complexity of the matching phase

$$\mathcal{T} = 2^{p+p_{in}+p_{out}-|\delta_{in}|+|k_{in}\cap k_{out}|}\left(c_{in} + c_{out} + c_{in}\cdot c_{out}\cdot 2^{-f}\right),$$

with $c_{in} = 2^{|\delta_{in}|+|k_{in}|-|k_{in}\cap k_{out}|+s_{in}-p_{in}}$, $c_{out} = 2^{|\delta_{out}|+|k_{out}|-|k_{in}\cap k_{out}|+s_{out}-p_{out}}$, and f corresponds to the amount of filtering that will be done through the structure and, additionally in our attack, through the middle conditions of the 0-round distinguisher technique. This procedure corresponds to the 23-round core of our attack, which is depicted in Fig. 3. Note that the cut between the upper and lower parts can be arbitrarily chosen, and will mainly change the

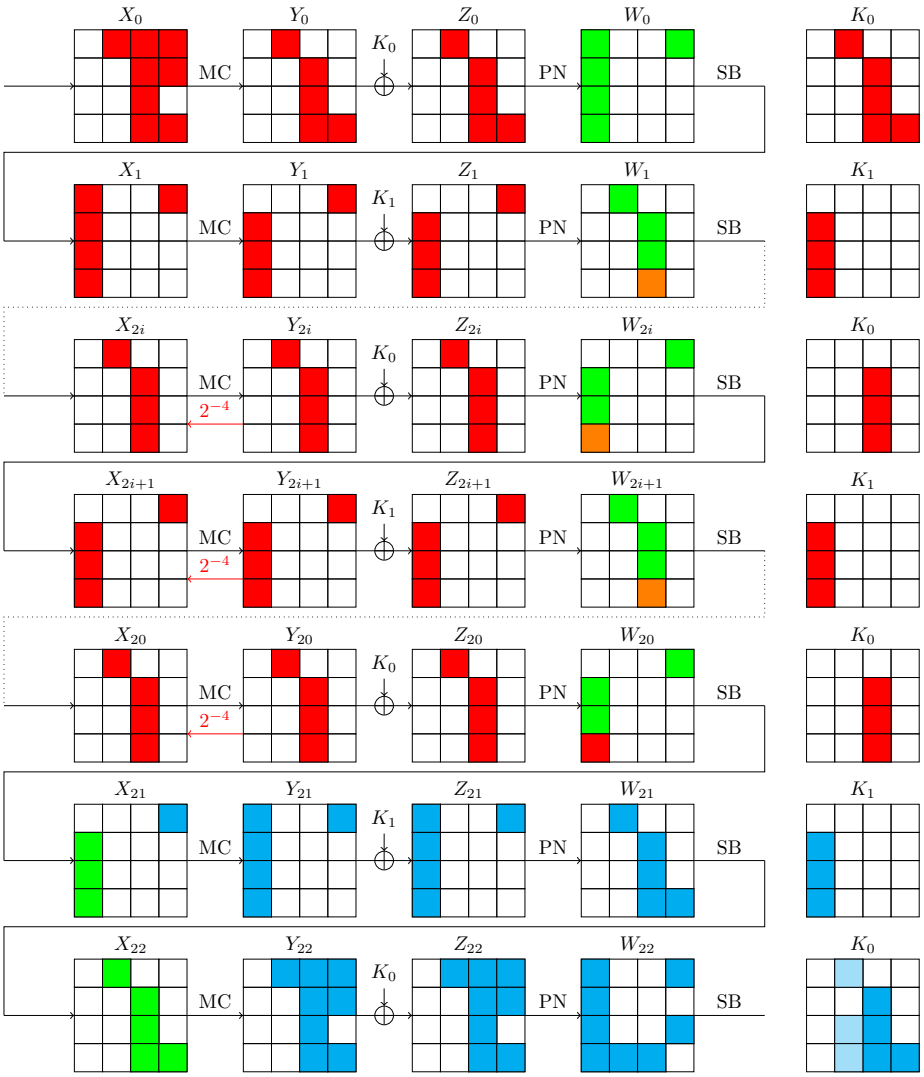


Fig. 3: Core of the 25-round attack on CRAFT. Green indicates that the values are known, while orange indicates the state-test guesses. Red and blue indicate the differential propagation in the upper and lower parts, respectively, as well as the guesses parts from the key. The light blue nibbles in K_{22} indicate that their XOR is guessed. Two rounds will be added with an structure in the end.

shape of the state-test equations. For the sake of simplicity, we chose it such that the probabilistic key-recovery and state-test guesses happen only in the upper part.

As alluded to earlier, reducing r_m to zero with this configuration means that the values at positions $W_{20}[3,4,8]$ and $X_{21}[4,8,12]$ from Fig. 3, as well as the differences in $W_{20}[12]$ and $X_{21}[3]$ are known both from the upper computation and from the lower one. More precisely, if we denote the pairs by $(W_{20}, \hat{W}_{20})$ and $(X_{21}, \hat{X}_{21})$, respectively, and the s-box of CRAFT by S, we get that the following conditions need to be verified when matching the upper and lower parts:

$$\begin{aligned} S(W_{20}[3]) + S(\hat{W}_{20}[3]) &= X_{21}[3] + \hat{X}_{21}[3] \\ W_{20}[12] + \hat{W}_{20}[12] &= S^{-1}(X_{21}[12]) + S^{-1}(\hat{X}_{21}[12]) \\ S(W_{20}[4]) &= X_{21}[4] \qquad & S(\hat{W}_{20}[4]) &= \hat{X}_{21}[4] \\ S(W_{20}[8]) &= X_{21}[8] \qquad & S(\hat{W}_{20}[8]) &= \hat{X}_{21}[8]. \end{aligned}$$

This implies a 24-bit filter. In addition, when building the candidates for the upper and lower parts we can organize them by the values of $\delta_{in} = \delta_{out}$, given by the differences in the 4 nibbles $X_{21}[3,4,8]$ and $W_{20}[12]$, to reduce the memory needs. The factors $2^{\delta_{in}} = 2^{16}$ and $2^{\delta_{out}} = 2^{16}$ are then common to both the first and second terms in the parenthesis and can be taken out from the time complexity. Additionally, the third term is reduced by a factor $2^{\delta_{in}+\delta_{out}}$ with respect to the previous complexities. As we are already ensuring the match of the middle difference, the 24-bit middle filter reduces to a 8-bit filter based on the values that need to match in $X_{21}[4,8]$ from both sides. Let us record this by $f_{dist} = 8$. With this, the time complexity of the matching phase becomes

$$\mathcal{T} = 2^{p+p_{in}+p_{out}+|k_{in}\cap k_{out}|}\left(c_{in} + c_{out} + c_{in}\cdot c_{out}\cdot 2^{-f-f_{dist}}\right),$$

$$\text{where} \qquad c_{in} = 2^{|k_{in}|-|k_{in}\cap k_{out}|+s_{in}-p_{in}}, \quad c_{out} = 2^{|k_{out}|-|k_{in}\cap k_{out}|+s_{out}-p_{out}},$$

where we have split the original f into f_{dist} and f to better keep track of the amount of filtering.

It should be noted that a naive computation of $\hat{P}$ from P could require us to make all the state-test guesses before we can start filtering by checking if the pair follows the (truncated) differential characteristic. However, this can easily be avoided by using rebound-like techniques, as it was already suggested in [8], and as we explain in more detail in Supplementary Material A.1. In the end, this has no (significant) impact on the complexity of the attack.

Adding Two Rounds using Structures Like previous attacks, we extend the core part of the attack by adding some rounds covered with initial structures. While previous attacks on CRAFT only used structures covering one round, we manage here to build an efficient two-round one. Previously, two round structures for differential MitM attacks had only been built in the case of partial key-additions, and only covered one round for full state key additions [1].

A structure in differential MitM attacks will allow to build a set of states of size $2^{\mathcal{S}} < 2^n$ in its input (or end of the lower part) and a set of states in its output (or ciphertext) of the same size, such that, depending on the key, there is a perfect correspondence between one element in the first set with one element in

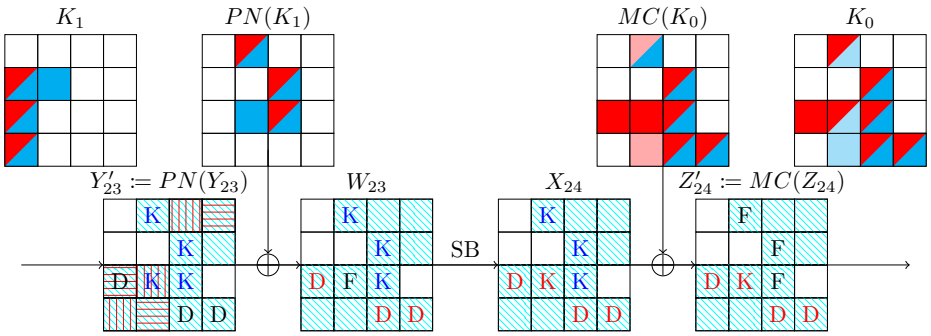


Fig. 4: Two-round structure of the 25-round attack on Craft. Red and blue key nibbles indicate known values from the upper and lower parts, respectively, while lighter colors indicate that a linear relation between them is known. For the state, the diagonally-striped (blue) nibbles indicate the continued propagation of the difference from the lower part, while the horizontally- and vertically-striped nibbles (red) indicate that their respective differences are identical. K indicates that the nibble can be computed by the part of the corresponding color (and correspond to the structure), while F indicates that the nibbles are fixed within the structure. A black D indicates that the difference in this nibble can be computed from both sides, while a red D indicates that the nibble is known from the upper part (and used to compute said difference).

the second. Using structures can sometimes not increase the complexity despite adding more rounds, as this $2^{\mathcal{S}}$ cost can be absorbed by the number of data we need to use, related to the probability of the core part. The important question next is how much we can filter then in the matching phase, in order to efficiently determine the candidate keys.

In contrast to [1], we extend the core of our attack by adding a two-round structure (compared to one round) at the ciphertext side. For convenience, let us move the key addition of these two rounds right before and right after the s-box layer, as depicted in Fig. 4. We are omitting some linear layers in this representation (while we take them into account in the equivalent key addition) that will be taken into account for computing the ciphertexts, but that will not affect the following reasoning, so we leave them out for the sake of simplicity. Besides the needed keys from each side seen in Fig. 3, we additionally guess the key nibbles $K_0[8,9]$ during the upper part and $K_1[5]$ during the lower part of the attack, which will provide an additional filtering for matching both parts, as the key $K_0[13]$ will become determined, as we have $K_0[1,9]$ and the difference $K_0[1] + K_0[9] + K_0[13]$ at that point, allowing for additional filtering using the difference $W_{23}[13] + \hat{W}_{23}[13]$ after the matching phase.

Like in [1], we will choose some words to fix, that allow to link determined values on each side so that the size of the structure is smaller than 2^{64} and can be used efficiently. For this, we fix $W_{23}[9]$ and $Z'_{24}[1,6,10]$ (marked with

an F in Fig. 4) to a chosen value. This corresponds to knowing the values of $Y'_{23}[1, 6, 9, 10]$ within the lower part and those of $Z'_{24}[1, 6, 9, 10]$ within the upper part, as the needed nibbles of the key have been guessed before in the respective parts.

As explained in [1], and with 4 fixed nibbles, we denote the size of the structure by $\mathcal{S} = n - 4 \cdot 4 = 48$, this gives us a filter of $f_{\mathcal{S}} = n - \mathcal{S}$ bits during the matching phase, which corresponds to the fixed parts that can be verified for $\hat{P}$ and $\hat{C}$.

Furthermore, we can compute the difference in Y'_{23} in the 3 nibbles marked by D solely based on information available in the upper and lower parts, respectively, meaning that we get an additional filter of $f_D = 12$ bits. Additionally, we can see that the difference of the lower part leads to no difference in the white nibbles of Z'_{24}, which gives us an additional filter of $f_{ZD} = 16$ bits. Notably, this filter can be applied directly in the upper part, potentially reducing memory complexity.

Since the overall probability of the core of our attack is $2^{-19 \cdot 4} = 2^{-76}$ and since the number of pairs we are able to construct is $2^{(2 \cdot 4 + 12) \cdot 4} = 2^{80}$, we can additionally fix one of the white nibbles of Z'_{24} (e.g., to zero) and still expect one right pair on average. Since fixing $z = 4$ bits in the upper part implies decreasing the probability of the matching by 2^z, we need to repeat the attack an additional 2^z times. However, as this also reduces the number of candidates c_{in} by the same factor, it effectively increases c_{out} only. Additionally, this implies that 4 bits of the ciphertext are fixed during the whole attack, reducing the data complexity to 2^{60}.

Additional Matching using Differentials To further filter the candidates during the matching phase, we realize that the differences at positions $Y'_{23}[2, 9, 12]$ (and $Y'_{23}[3, 8, 13]$) all come from a single active word, so they have to be the same (as indicated by the horizontal and vertical stripes in Fig. 4). From the upper part we can compute the difference at $Y'_{23}[8, 9]$, which means that, on average, at most half of the differences in each of the 4 nibbles $Z'_{24}[2, 3, 12, 13]$ are possible. This gives us an additional filter of $f_{ID} = 4$ bits. Similarly to the filter based on the inactive nibbles of Z'_{24}, this filter is independent of the lower part and can therefore be applied in the upper part directly, potentially reducing memory complexity.

Complexity We record the parameters of our attack in Table 2. With this, the time complexity of all the previous steps is

$$\mathcal{T} = I \cdot \left(c_{in} + c_{out} + c_{in} \cdot c_{out} \cdot 2^{-f}\right) \approx 2^{117.58}, \quad \text{where}$$

$$I = 2^{p+p_{in}+p_{out}+|k_{in} \cap k_{out}|-|\mathcal{S}|+z} = 2^{60}, c_{out} = 2^{|\mathcal{S}|+|k_{out}|-|k_{in} \cap k_{out}|+s_{out}-p_{out}} = 2^{56},$$

$$c_{in} = 2^{|\mathcal{S}|+|k_{in}|-|k_{in} \cap k_{out}|+s_{in}-p_{in}-z} = 2^{56}, f = f_{\mathcal{S}} + f_{ZD} + f_D + f_{ID} + f_{dist} = 56,$$

while $I \cdot c_{in} \cdot c_{out} \cdot 2^{-f} = 2^{60+56+56-56} = 2^{116}$ candidates survive the matching.

Further, the data complexity is 2^{60} by fixing one of the nibbles $Z'_{24}[0, 4, 5, 11]$ (e.g., $Z'_{24}[0] = 0$) during the whole attack (as explained above). However, while

the memory complexity would usually corresponds to $\min(c_{in} \cdot 2^{-16-4}, c_{out}) = 2^{36}$, the data reduction trick requires us to store plain- and ciphertexts in order to prevent an encryption query that results in a ciphertext outside of those of interest (i.e., $Z'_{24}[0] \neq 0$). To solve this, from each given ciphertext, we request and store the decryption of the 2^{48} associated ciphertexts that have 0 difference at the four nibbles without difference (and additionally have one of them fixed to a 4-bit value for all the ciphertexts). This allows us to encrypt during the upper part of the precedure using a simple lookup, while limiting the data to 2^{60} with a memory of 2^{48}. If the lookup fails, we know that the current candidate is not of interest and can be discarded, which is identical to filtering for the 4 differences in Z'_{24} to be zero.

While we still need to recover the whole key, this can be done without increasing the complexity, as we will see next. With that, a pseudo-code for the whole attack is given in Supplementary Material A.2 for the interested reader.

Recovering the Remaining Part of the Key After matching the candidates from the upper and lower parts, we still need to recover the part of the key that we have not guessed. For this, we can make use of the equations we get from the state-test guesses and the two round structure. Starting with the structure, we note that we get equations of the form

$$X_{24}[i] + k'_i = S(W_{23}[i] + k_i) \quad \text{and} \quad \hat{X}_{24}[i] + k'_i = S(\hat{W}_{23}[i] + k_i)$$

for all the blue nibbles in Fig. 4, while we get a single equation $X_{24}[i] + k'_i = S(W_{23}[i]+k_i)$ for each of the remaining four. Hence, we can solve equations of the form $S(W_{23}[i]+k_i)+S(\hat{W}_{23}[i]+k_i) = X_{24}[i]+\hat{X}_{24}[i]$ nibble-wise by considering which input pairs $(W_{23}[i]+k_i, \hat{W}_{23}[i]+k_i)$ lead to difference $X_{24}[i]+\hat{X}_{24}[i]$. As we can assume the difference $X_{24}[i] + \hat{X}_{24}[i]$ to be uniformly distributed, and since the average number of keys satisfying such an equation is one (where the average is taken over this difference), we expect one solution for k_i.

However, we already used 4 of these differences for the matching phase, meaning that we now expect 2 solutions for them on average. But as mentioned above, we can first use $W_{23}[13] + W'_{23}[13]$ to have an additional filter of 2^{-4}. In other words, the amortized time complexity (over the number of candidates) of this step matches the number of candidates. As knowing k_i trivially gives k'_i by using one of the equations above, the only unknown values of the key are $K_0[0, 4, 5, 11]$ and $K_1[7, 9, 10, 15]$ at this point of the attack, since they correspond to the nibbles of the structure without a difference. Still, we have four single equations of the form $X_{24}[i]+k'_i = S(W_{23}[i]+k_i)$ relating each unknown key nibble of K_0 to one of K_1.

r_{in}	r_m	r_{out}	p	p_{in}	p_{out}	s_{in}	s_{out}	$\lvert k_{in}\rvert$	$\lvert k_{out}\rvert$	$\lvert k_{in} \cap k_{out}\rvert$	$\lvert\mathcal{S}\rvert$	$f_{\mathcal{S}}$	f_{ZD}	f_D	f_{ID}	f_{dist}	z
21	0	2	0	76	0	76	0	40	36	28	48	16	16	12	4	8	4

Table 2: Parameters of our 25-round attack on CRAFT

Additionally, we can make use of the equations that result from our state-test guesses and the fact that we know the values of $X_{21}[3]$ and $W_{20}[12]$. The most important ones are the following four:

$$W_1[14] = K_1[3] + S(K_0[0] + X_0[0] + X_0[12] + X_0[8]) + S(K_0[14] + X_0[14]) + S(K_0[7] + X_0[15] + X_0[7])$$

$$W_2[12] = K_0[1] + S(K_1[12] + S(K_0[1] + X_0[1] + X_0[13] + X_0[9])) + S(K_1[2] + S(K_0[13] + X_0[13]) + S(K_0[3] + X_0[11] + X_0[15] + X_0[3]) + S(K_0[4] + C_0[0] + X_0[12] + X_0[4])) + S(K_1[5] + C_1[1] + S(K_0[2] + X_0[10] + X_0[14] + X_0[2]) + S(K_0[9] + X_0[9]))$$

$$W_3[14] = K_1[3] + S(K_0[0] + S(K_1[1] + S(K_0[12] + X_0[12]) + S(K_0[2] + X_0[10] + X_0[14] + X_0[2]) + S(K_0[5] + C_0[1] + X_0[13] + X_0[5])) + S(K_1[15] + S(K_0[0] + X_0[0] + X_0[12] + X_0[8])) + S(K_1[6] + S(K_0[3] + X_0[11] + X_0[15] + X_0[3]) + S(K_0[8] + X_0[8]))) + S(K_0[14] + S(G[0])) + S(K_0[7] + S(K_1[0] + S(K_0[1] + X_0[1] + X_0[13] + X_0[9]) + S(K_0[15] + X_0[15]) + S(K_0[6] + X_0[14] + X_0[6])) + S(K_1[11] + S(K_0[7] + X_0[15] + X_0[7])))$$

$$W_{20}[12] = K_0[1] + K_0[13] + K_0[9] + S(K_1[10] + K_1[14] + K_1[2] + S(K_0[11] + K_0[15] + K_0[3] + W_{22}[0] + W_{22}[14] + W_{22}[7]) + S(K_0[12] + K_0[4] + C_{22}[0] + W_{22}[1] + W_{22}[10]) + S(K_0[13] + W_{22}[2])) + S(K_1[12] + S(K_0[1] + K_0[13] + K_0[9] + W_{22}[12] + W_{22}[2] + W_{22}[5])) + S(K_1[13] + K_1[5] + C_{21}[1] + S(K_0[10] + K_0[14] + K_0[2] + W_{22}[13] + W_{22}[3] + W_{22}[4]) + S(K_0[9] + W_{22}[5]))$$

As can be seen, the first two equations contain one unknown key nibble (purple) only. Hence, we immediately get $K_0[0, 4]$, and therefore $K_1[10, 15]$, from the state-test guesses. With this, the other two equations give $K_0[5, 11]$, and thus also $K_1[7, 9]$. In other words, we have recovered the whole key.

While the remaining state-test equations can be used to further filter the candidates at various points of this step, it is easy to see that the average time and memory complexity is at most one for each candidate. In other words, further improvements in the recovery of the remaining part of the key have no (significant) effect on the complexity of the attack.

4.2 An Attack on 26 Rounds of Craft

Our 26-round attack on Craft works in a similar way. We give the core of the attack, based on the same iterative distinguisher as the previous ones, but with one round added at the ciphertext side, in Supplementary Material A.3 and the two-round added structure in Fig. 5. For the upper part, we guess in addition to the key words needed in the core scheme, the word $K_1[0]$, and for the lower part the word $K_0[2]$. Since $p_{in} + p + p_{out} - \delta_{in} = 80 + 0 + 0 - 16 = 64$, this attack requires the whole code-book. We therefore cannot fix part of the ciphertexts during the attack to apply the data reduction technique. But this also means that we do not need to store the plain- and ciphertexts, as we can simply assume oracle access to the en- and decryption functions.

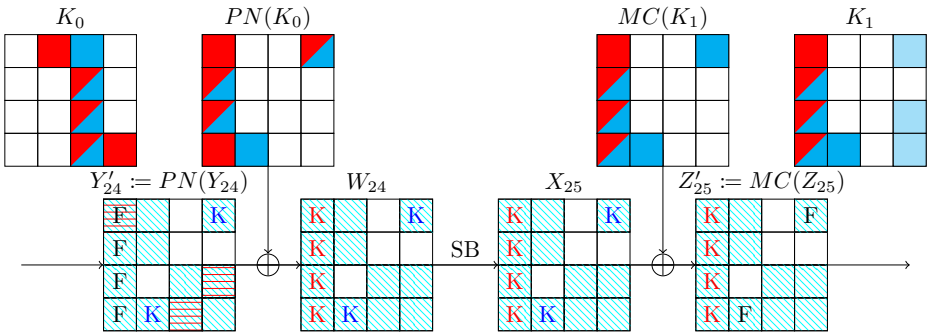


Fig. 5: Two-round structure of the 26-round attack on Craft. Red and blue key nibbles indicate known values from the upper and lower parts, respectively, while lighter colors indicate that a linear relation between them is known. For the state, the diagonally-striped (blue) nibbles indicate the continued propagation of the difference from the lower part, the horizontal-striped (red) ones indicate that the difference in these nibbles is identical (due to MC; only depicting those used in the attack). K indicates that the nibble can be computed by the part of the corresponding color, F indicates that the nibbles are fixed within the structure.

r_{in}	r_m	r_{out}	p	p_{in}	p_{out}	s_{in}	s_{out}	$\lvert k_{in}\rvert$	$\lvert k_{out}\rvert$	$\lvert k_{in} \cap k_{out}\rvert$	$\lvert\mathcal{S}\rvert$	$f_{\mathcal{S}}$	f_{ZD}	f_D	f_{ID}	f_{dist}	z
22	0	2	0	80	0	80	0	36	36	24	40	24	16	0	2	8	0

Table 3: Parameters of our 26-round attack on Craft

We can see in the figure that we have an structure of size $2^{\mathcal{S}} = 2^{64-24}$, as we fix $Y'_{24}[0, 4, 8, 12]$, $Z'_{25}[13]$ and $Z'_{25}[3] + Z'_{25}[11] + Z'_{25}[15]$ (marked by F in Fig. 5). We can similarly apply the 16-bit filter due to the 4 words without difference to the upper part. Here, as in the previous attack, we can also filter the candidates in the upper part based on the (known) difference in $Y'_{24}[0]$, which needs to be identical to the ones in $Y'_{24}[11]$ and $Y'_{24}[14]$ (striped in red). As this implies that (at most) half of the differences in $Z'_{25}[11]$ and $Z'_{25}[14]$ are possible, this gives us an additional filter of 2^{-2}.

We record the parameters of this attack in Table 3. Similar to the previous attack, the complexity so far is

$$\mathcal{T} = I\left(c_{in} + c_{out} + c_{in} \cdot c_{out} \cdot 2^{-f}\right) \approx 2^{118}, \quad \text{where}$$
$$I = 2^{p+p_{in}+p_{out}+|k_{in}\cap k_{out}|-|\mathcal{S}|} = 2^{64}, c_{out} = 2^{|\mathcal{S}|+|k_{out}|-|k_{in}\cap k_{out}|+s_{out}-p_{out}} = 2^{52},$$
$$c_{in} = 2^{|\mathcal{S}|+|k_{in}|-|k_{in}\cap k_{out}|+s_{in}-p_{in}} = 2^{52}, f = f_{\mathcal{S}} + f_{ZD} + f_D + f_{ID} + f_{dist} = 50,$$

which corresponds to the number of candidates. The data complexity is the whole code book, and the memory complexity will be $\min(c_{in} \cdot 2^{-16-2}, c_{out}) = 2^{34}$.

The remaining of the key can again be recovered with an average complexity of at most 1 per candidate, by making use of the state-test equations and the

structure, similar to before. We provide the full details on this in Supplementary Material A.4.

4.3 An Attack on 23 and 24 Rounds of Craft

As summarized in Table 1, we can mount an attack on 23 and 24 rounds of Craft by removing **two rounds** in the core attack of the 25 Fig. 4 and 26-round Fig. 6 attacks respectively, and using the same procedure. This way, the probability of the differential path increase of a factor 2^8 for each attack and thus the time complexity decreases of the same factor. Moreover, we will use the data reduction technique originally proposed in [8] as it does not increase the time complexity.

5 Improved Cryptanalysis of Pride

In this section, we provide a new application of the state-test technique on the block cipher Pride (described in section 2.1). The best known attack so far is a classical differential attack [21], reaching 18 rounds. Other attacks have been published [28,27] though they were shown to be incorrect in [21]. No improved results on Pride have appeared in the last 8 years.

We have applied the state-test technique to improve the previous best attack, allowing to improve the time complexity (or slightly the data complexity in another trade-off). This application, besides improving the best known attack on Pride up to date, gives the first application using the state-test technique in the classical differential setting using counters.

5.1 Previous best known attack

In [21], the authors proposed a differential attack on 18 rounds of Pride. They use the following 1-round iterative differential characteristic given in hexadecimal value : 0000 0008 0000 0008. The distinguisher is an iteration of this characteristic for 14 rounds and has a probability of 2^{-56}. The distinguisher covers from round 3 to round 17 and is extended for two rounds in the backward and forward directions, as shown in Table 4. The time, data and memory complexities of their attack are $\mathcal{T} = 2^{63.3}$ Pride encryptions, $\mathcal{D} = 2^{61}$ chosen plaintexts and $\mathcal{M} = 2^{35}$ 64-bit blocks. In the rest of this subsection, as the first steps of our attack are the same as in [21], we give the outline of their attack.

Output/input difference of 8 *in a Pride s-box* We describe here a property used in their attack that will also have an impact in our analysis. As shown in [21], the s-box of Pride has an interesting property when the input or output is 8. Indeed, there are 4 possible input differences to get an output difference of 8: $2, 3, 8, a$. We denote the bits of a nibble as $x_1x_2x_3x_4$, where x_1 is the most significant bit. For each input difference, we have the following possible pairs of values :

- Input difference of 2 : $\{(0000); (0010)\}$ and $\{(1000); (1010)\}$.

– Input differences of 3 : $\{(0100);(0111)\}$ and $\{(1100);(1111)\}$.
– Input differences of 8 : $\{(0110);(1110)\}$ and $\{(0101);(1101)\}$.
– Input differences of a : $\{(0001);(1011)\}$ and $\{(1001);(0011)\}$.

Thus, in every case, we get a condition on the value of the bit x_2 and in the case of an input difference of 2 and a the value of the least significant bit is fixed. Otherwise, if the value of the least significant bit is fixed, then the value of the bit x_3 is determined, thus we get a condition on the bit x_3.

Generating pairs We need to estimate the number of pairs needed to obtain about a pairs that will follow the differential path. We see in Table 4 that after the backward propagation of the difference δ_{in} with probability 1, there are 7 active nibbles in round 1. Therefore, a structure contains 2^{28} different plaintexts and can form $2^{28+28-1} = 2^{55}$ different pairs. There are 2^{36} possible structures, thus we can compute $2^{36+55} = 2^{91}$ pairs. Out of the 2^{91} pairs, a proportion of 2^{-28} will be partially encrypted to δ_{in} for the correct key. In total, to have a pairs that verify the first two rounds and also follow the differential path, we need to construct around $a2^{56+28} = 2^{84}$ pairs. For this, we build $a2^{84-55} = a \cdot 2^{29}$ structures. In [21], they choose $a = 2^4$.

Pre-sieving To limit the memory complexity, the authors pre-compute the possible differences in the plaintexts and ciphertexts that can lead to Δ_{in} and Δ_{out}. They found that there are $2^{17.38}$ possible ΔP and $2^{19.93}$ ΔC. Hence, out of the $a \cdot 2^{84}$ pairs, only $a \cdot 2^{84}(2^{17.38} \cdot 2^{-27})(2^{19.93} \cdot 2^{-64}) = a \cdot 2^{29.31}$ can lead to the distinguisher. Thus, instead of storing the counters for the keys, the authors proposed to store the $a \cdot 2^{29.31}$ pairs so we need to save $a \cdot 2^{31.31}$64-bit blocks.

Filtering candidates For each of the $a \cdot 2^{29.31}$ pairs, we do the following operations, while we keep a counter of the guessed key bits to check the ones that have more appearances [5]. In the following attack, the authors compare the time

[5] As for Pride the product of time and data complexity must not be bigger than 2^{128}, all the candidate pairs can not be tested in the end because the product would be too high, and a counter for just keeping the values with most occurrences is needed.

$\Delta P = \Delta X_1$	0000 0000 0000 ???? ???? ???? 0000 ???? ???? 0000 0000 ???? 0000 0000 0000 ????
ΔY_1	0000 0000 0000 ?00? 00?0 00?0 0000 ?00? 00?0 0000 0000 ?00? 0000 0000 0000 ?0??
ΔZ_1	000? 000? 000? 000? 0000 0000 0000 0000 0000 ??00 ?000 000? 000? 000? 000? 000?
ΔW_1	0000 000? 0000 000? 0000 0000 0000 0000 000? 0000 000? 0000 0000 000? 0000 000?
$\Delta I_2 = \Delta X_2$	0000 0000 0000 0000 0000 0000 0000 ?0?? 0000 0000 0000 0000 0000 0000 0000 ?0??
ΔY_2	0000 0000 0000 0000 0000 0000 0000 1000 0000 0000 0000 0000 0000 0000 0000 1000
...	...
$\Delta I_{17} = \Delta X_{17}$	0000 0000 0000 0000 0000 0000 0000 1000 0000 0000 0000 0000 0000 0000 0000 1000
ΔY_{17}	0000 0000 0000 0000 0000 0000 0000 ?0?? 0000 0000 0000 0000 0000 0000 0000 ?0??
ΔZ_{17}	0000 000? 0000 000? 0000 0000 0000 0000 000? 0000 000? 0000 0000 000? 0000 000?
ΔW_{17}	000? 000? 000? 000? 0000 0000 0000 0000 ?0?? 0000 ?00? 000? 000? 000? 000? 000?
$\Delta I_{18} = \Delta X_{18}$	00?0 0000 00?0 ?0?? 0000 0000 0000 ?00? 00?0 0000 0000 ?0?? 0000 0000 0000 ?0??
$\Delta Y_{18} = \Delta C$	???? 0000 ???? ???? 0000 0000 0000 ???? ???? 0000 0000 ???? 0000 0000 0000 ????

Table 4: Differential path of the 14-round characteristic.

complexity of each operation to a Pride encryption consisting of $18 \cdot 16 = 2^{8.17}$ s-box computations. Thus they consider a factor $2^{-8.17}$ when comparing the complexity to one Pride encryption.

1. First we guess $K_0[1, 3, 4, 8, 9, 12, 16]$ and filter the candidates through the active s-boxes at round 18. There are now $a \cdot 2^{29.31} \cdot 16 \cdot 2^{-19.93} \cdot 2^{28} = a \cdot 2^{41.38}$ candidates for 28 bits of the key.
2. Then we filter the candidates to verify the transitions in round 1. Guess $K_0 \oplus K_1[4, 5, 6, 8, 9, 12, 16]$ and obtain $a \cdot 2^{41.38} \cdot 16 \cdot 2^{-17.38} \cdot 2^{28} = a \cdot 2^{56}$ candidates for 56 bits of the key.
3. Now we need to filter through the s-boxes at round 2 and 17. Thanks to the key schedule we already know $K_2[8], K_2[16], K_{17}[8], K_{17}[16]$ and can compute the most and least significant bits of $X_2[8]$, $X_2[16]$, $Y_{17}[8]$ and $Y_{17}[16]$.
   In [21], they guess $(K_0 \oplus K_1)^{2,3,4}[1]$, $(K_0 \oplus K_1)^{2,3,4}[1]$ to compute $X_2^2[8]$, and $X_2^2[16]$. The time complexity of this operation is $@ \cdot a \cdot 2^{56+6} 2^{-8.17} = a \cdot 2^{63-8.17} = a \cdot 2^{54.83}$ Pride encryptions.
4. The authors remark that all the bits to compute $X_2^3[8] \oplus X_2^3[16]$ are known. Thus we have a 1-bit filter without guessing more bits. Then to compute $X_2^3[8]$, guess $(K_0 \oplus K_1)[15]$. It filters one more bit and gives $a2^{60+4-2} = a2^{62}$ candidates. The cost of this is $2^{67-8.17} = 2^{58.83}$ Pride encryptions.
5. At this point, they have 2^{66} candidates for 66 key bits. The wrong keys should appear on average only one time while the right pair should be suggested 2^4 times. Since the counter for a wrong key follows a binomial distribution $\mathcal{B}(2^{66}, 2^{-66})$, by only keeping the key candidates that appear at least 14 times, the probability that a wrong candidate is suggested enough times is $2^{-37.7}$ thus they keep $2^{66-37.7} = 2^{28.3}$ candidates.
6. Then they guess 3 more bits to compute $Y_{17}^2[8]$ with a complexity of $2^{28.3+3-8.17} = 2^{23.13}$ Pride encryption and they guess 7 bits to compute $Y_{17}^2[16]$ for a time complexity of $2^{30.3+7-8.17} = 2^{29.13}$ Pride encryption.
7. Finally, they filter again the candidates and obtain $2^{11.3}$ candidates for 76 key bits. Then, the exhaustive search cost $2^{11.3+52} = 2^{63.3}$ Pride encryptions. This is the bottleneck of the time complexity.

Moreover as all the active bits of the differential path are not computed, the authors choose a equal to 2^4 to filter more candidates. The filtering phase of the attack is not the bottleneck of the time complexity.

The final complexities are: $\mathcal{D} = 2^{61}$, $\mathcal{T} = 2^{63.3}$ for the exhaustive search and $\mathcal{M} = 2^{35}$ 64-bit blocks. Thus the product of the data and time complexities is $\mathcal{D} \times \mathcal{T} = 2^{61} \times 2^{63.3} = 2^{123.3}$

5.2 Overview: using state-test equations for reducing the guesses

We can notice that in rounds 2 and 17, for computing the necessary bits to verify the distinguisher, namely $X_2^2[8, 16]$, $X_2^3[8, 16]$, $Y_{17}^2[8, 16]$ and $Y_{17}^3[8, 16]$ given in the subsection 5.3, we need to guess 36 bits of the subkeys that are still unknown after guessing the bits needed to compute the first and last rounds, which considerably increases the complexity. Instead, we can use the state-test technique

to improve the attack. First by fixing some bits of the plaintexts which will improve the signal to noise ratio of the counter as it will allow to use some of the state-test equations to define partitions of the key as in impossible-differential attacks, that can be included in the determined part regarding the number of occurrences. Next, after filtering the candidates with regard to the counter, we can use the remaining state-test equations to recover more information on the key and further reduce the number of remaining candidates.

5.3 State-test equations from Pride

We give here the 8 state-test equations that we will consider in our attack. P and C are the plaintexts and ciphertexts respectively. And the states I,Y and X, as shown in 4, are the states before applying the round function, the states after the s-box layer and after the addition of the round key respectively. The still unknown bits of the key are in purple.

Equations on round 2:

$$X_2^2[8] = K_2^2[8] \oplus Y_1^2[8] \oplus P^4[1] \oplus (K_0 \oplus K_1)^4[1] \oplus P^4[11] \oplus (K_0 \oplus K_1)^4[11] \oplus (P^2[1] \oplus (K_0 \oplus K_1)^2[1])(P^3[1] \oplus (K_0 \oplus K_1)^3[1]) \oplus (P^2[11] \oplus (K_0 \oplus K_1)^2[11])(P^3[11] \oplus (K_0 \oplus K_1)^3[11]).$$

$$X_2^3[8] = K_2^3[8] \oplus Y_1^3[4] \oplus Y_1^3[5] \oplus SB(P^1[15] \oplus (K_0 \oplus K_1)^1[15], P^2[15] \oplus (K_0 \oplus K_1)^2[15], P^3[15] \oplus (K_0 \oplus K_1)^3[15], P^4[15] \oplus (K_0 \oplus K_1)^4[15])^3.$$

$$X_2^2[16] = K_2^2[16] \oplus Y_1^2[8] \oplus Y_1^2[12] \oplus P^4[11] \oplus (K_0 \oplus K_1)^4[11] \oplus (P^2[11] \oplus (K_0 \oplus K_1)^2[11])(P^3[11] \oplus (K_0 \oplus K_1)^3[11]).$$

$$X_2^3[16] = K_2^3[8] \oplus Y_1^3[4] \oplus Y_1^3[16] \oplus SB(P^1[15] \oplus (K_0 \oplus K_1)^1[15], P^2[15] \oplus (K_0 \oplus K_1)^2[15], P^3[15] \oplus (K_0 \oplus K_1)^3[15], P^4[15] \oplus (K_0 \oplus K_1)^4[15])^3.$$

Equations on round 17:

$$Y_{17}^2[8] = C^4[6] \oplus C^4[7] \oplus C^4[14] \oplus (C^2[6] \oplus K_0^2[6])(C^3[6] \oplus K_0^3[6]) \oplus (C^2[7] \oplus K_0^2[7])(C^3[7] \oplus K_0^3[7]) \oplus (C^2[14] \oplus K_0^2[14])(C^3[14] \oplus K_0^3[14]) \oplus K_0^4[6] \oplus K_0^4[7] \oplus K_0^4[14] \oplus K_{18}^2[6] \oplus K_{18}^2[7] \oplus K_{18}^2[14].$$

$$Y_{17}^3[8] = I_{18}^3[3] \oplus SB(C^1[2] \oplus K_0^1[2], C^2[2] \oplus K_0^2[2], C^3[2] \oplus K_0^3[2], C^4[2] \oplus K_0^4[2])^3 \oplus SB(C^1[10] \oplus K_0^1[10], C^2[10] \oplus K_0^2[10], C^3[10] \oplus K_0^3[10], C^4[10] \oplus K_0^4[10])^3 \oplus K_{18}^3[2] \oplus K_{18}^3[3] \oplus K_{18}^3[10].$$

$$Y_{17}^2[16] = I_{18}^2[3] \oplus I_{18}^2[12] \oplus C^4[11] \oplus (C^2[11] \oplus K_0^2[11])(C^3[11] \oplus K_0^3[11]) \oplus K_0^4[11] \oplus K_{18}^2[3] \oplus K_{18}^2[11] \oplus K_{18}^2[12].$$

$$Y_{17}^3[16] = I_{18}^3[16] \oplus K_{18}^3[16] \oplus SB(C^1[7] \oplus K_0^1[7], C^2[7] \oplus K_0^2[7], C^3[7] \oplus K_0^3[7], C^4[7] \oplus K_0^4[7])^3 \oplus SB(C^1[15] \oplus K_0^1[15], C^2[15] \oplus K_0^2[15], C^3[15] \oplus K_0^3[15], C^4[15] \oplus K_0^4[15])^3 \oplus K_{18}^3[7] \oplus K_{18}^3[15].$$

5.4 How to consider some state-test equations in the counter

In this subsection, we explain how to build partitions thanks to state-test equations so that this information can be included in the counter. Since we make a guess on the value of the bit of the internal state, we cannot use this bit of information in the key counter directly. To take the state-test equations into account

in the counter, we use a similar idea as in [10]: that is to fix some part of the plaintexts to obtain a partition of the key. Equations on s bits of the internal state can be separated in two parts, one part involving the value of the state and known bits of the key and the plaintext, and another part involving κ unknown bits of the key and y bits of the plaintext on which the equation depends on non-linearly. If we fix those y bits then for a guess of the internal state and a given candidate, only $2^{\kappa-s}$ solutions are left for the κ bits of the key. That means that the probability for a wrong candidate key with state-test equations on s bits of internal state to reach the differential distinguisher decreases by a factor 2^{-s}, and the guesses of the internal bits given by state-test equations define disjoint partitions of the key space that can be treated as key bits of information.

However, since we are fixing bits of the plaintexts, we are limited by the data needed to perform the attack. As such we can not use all the equations given by the state-test in the counter, as we can not fix enough bits in the plaintexts (we wouldn't have enough data). Fortunately, the rest of the equations defines a system of equations which, for the good key guess, gives the correct solution. As we expect to find several occurrences verifying the distinguisher for the counter of the correct guess, considering all the associated equations together form a much bigger and overdetermined system of equations, that will only have a solution for the correct value of the counter. In addition of allowing us to filter the bad key counters, this system of equations allows us to recover more key bits. For a wrong key guess, the equations of the overdetermined system are uniformly distributed and we can verify that the system is not consistent. The probability that a system of x equations over k key bits, for a wrong key candidate, has a solution is 2^{x-n}. In our case, this will be $t_o \times (8 - n_p)$, where n_p is the number of equations already used for defining partitions, and t_o is the expected number of occurrences in the counter. It is easy to intuitively see how this probability can be quite low, and allows to filter many additional bad key candidates.

5.5 How to deal with the bad candidates

For the good key candidate of the counter, it could happen that we recover a tuple of associated data and key guess with incorrect values for the state-test equations that incorrectly seems to lead to the differential path. This would be translated in one incorrect equation among the overall system of correct equations previously discussed. The probability of this is very low, as for the correct key guess, leading to the differential probability through the external rounds without verifying the differential happens on average with less than the inverse of the numbers of pairs considered. In our attack, we consider we have at most one incorrect equation in the overall system, and we will try to solve all the subsystems when erasing one different equation at the time. We will also see that this won't affect the bottleneck of the complexity, as when we arrive at this step, the remaining number of candidates is already considerately reduced.

5.6 How to deal with bad equations for good values of the counter

As seen with the property of the s-box of Pride, for a difference of 2 or a, the bit in position x_3 can take two possible values to reach the distinguisher. This means that after filtering the candidates, the same equation can appears two times but with two different values. Obviously only one of the 2 related equations can be correct and associated to the correct key. We will call this "double equations", in opposition to the rest that will be "single equations".

In our attack, for the good key, this happens for half of the occurrences (lets call the number of double equations n_d). In this case, for solving the correct final system, we need to consider all the 2^{n_d} possible systems that consider all the possible configurations formed by choosing one out of the two double equations for all of them. If we keep the key candidates that appear in the counter at least t times, then on average we will have $\frac{t}{2}$ "single" equations and $\frac{t}{2}$ "double" equations. Thus, if we also consider the previous paragraph, on average for each considered candidate in the counter we need to try to solve around $\frac{t}{2}2^{\frac{t}{2}}+(\frac{t}{4})2^{\frac{t-1}{2}}$ systems to find the one leading to the correct key. The first term is associated with erasing a single equation because of a bad candidate and trying all the possibilities for the double equations, and the second is associated with erasing a double equation and trying all the possibilities for the remaining ones. This complexity is usually small and does not increase the complexity of the attack, as it is applied after a first sieving of candidates through the counter.

5.7 Step-by-step scheme of the attack

Our attack on 18 rounds of Pride uses the same path as the previous one. We use the state-test technique to increase the signal to noise ratio and improve the attack. As in [21], the time complexity will be counted in function of the cost of a Pride encryption. In the following paragraph, we give the steps of our attack depending on the number of equations used directly in the counter:

1. First, we fix m bits of the plaintexts or ciphertexts that are part of the equations of the value of n_p bits of the 8 internal state bits from 5.3 and involve k_p subkey bits. This way we have a partition of $2^{k_p-n_p}$ disjoint sets of the k_p subkey bits depending of the guessed values of the n_p bits of internal states. Thus those n_p bits distinguish the right key from the wrong keys and we have n_p bits of key information taken into account in the counter. We can also remark that the number m of plaintext or ciphertext bits fixed depends on the data available, here $m \leq 64 - (57 + log_2(a))$.
2. We build $a \cdot 2^{29}$ structures of 2^{55} pairs of plaintexts or ciphertexts each.
3. Then the next 3 steps of our attack are done the same way as the first 3 steps of the previous attack. Thus, to limit the memory complexity, we store the $a \cdot 2^{29.31}$ pairs with possible differences instead of the counters for the keys so we need to save $a \cdot 2^{31.31}$ 64-bit. And we filter the candidates at round 18 and 1 by guessing $K_0[1, 3, 4, 8, 9, 12, 16]$ and $K_0 \oplus K_1[4, 5, 6, 8, 9, 12, 16]$. We obtain $a \cdot 2^{56}$ candidates for $56 + n_p$ bits of key information. Those steps cost at most $a \cdot 2^{53.63}$ Pride encryptions.

4. We filter through the active s-boxes of round 2 and 17. Thanks to the key schedule we already know $K_2[8], K_2[16], K_{17}[8]$ and $K_{17}[16]$, and we can compute the most and least significant bits of $X_2[8]$, $X_2[16]$, $Y_{17}[8]$ and $Y_{17}[16]$. First, since $X_2^3[8] \oplus X_2^3[16]$ depend only on known bits, we can compute its value and filter half of the candidates. We have now $a \cdot 2^{55}$ candidates for $56 + n_p$ bits of key information.
   The idea of the following steps is to guess all the key bits needed to compute some of the active bits of round 2 and 17 and filter out some of the candidates to improve the signal to noise ratio.
5. Thus we guess k_s subkey bits determining s bits of the internal state. We have now $a \cdot 2^{55+k_s-s}$ candidates for $56 + n_p + k_s$ bits of key information. The bottleneck of the time complexity of this step is $a \cdot 2^{55+k_s-s+1} \cdot 2^{-8.17}$. At this point, The counters of the wrong key guesses follow a binomial distribution $\mathcal{B}(a \cdot 2^{55+k_s-s}, 2^{-56-n_p-k_s})$ while the counter of the right key follows the binomial distribution $\mathcal{B}(a \cdot 2^{56}, 2^{-56})$ thus a wrong key is supposed to be suggested about $\lambda = a \cdot 2^{55+k_s-s-(56+n_p+k_s)} = a \cdot 2^{-1-s-n_p}$ times while the right key is suggested about a times.
6. By keeping the keys that are suggested at least t times, we reduce the number of candidates we handle by a factor P_t which is the probability for a wrong pair to be suggested at least t times. As we can approximate a binomial distribution of parameter n, p ($\mathcal{B}(n, p)$) as a Poisson distribution of parameter np then $P_t = 1 - e^{-\lambda} \sum_{k=0}^{t} \frac{\lambda^k}{k!}$ and the probability of success of the attack is $P_s = 1 - e^{-a} \sum_{k=0}^{t} \frac{a^k}{k!}$. Since $\lambda = a \cdot 2^{-1-s-n_p}$, we remark that the higher s and n_p, the lower P_t is. Moreover, P_t also decreases if we take a higher threshold t. However the probability of success also decreases as t increases above a. There is a trade-off between the threshold t of the counters and a, the number of times the good key is expected to be suggested.
   Now, we have $a \cdot 2^{55+k_s-s} \cdot P_t = 2^{\kappa}$ candidate keys to test.
7. We still have $n_o = 8 - n_p - 1 - s$ equations over k_o subkey bits to use. For each remaining candidates, the n_o equations of the internal states appear at least t times so we recover a system of $t \cdot n_o$ equations which is often an overdetermined system. As explained in subsection 5.6, we need to solve the system while taking into account the potential bad candidates. Moreover the probability for a wrong key to have a solution is $2^{k_o - t \cdot n_o}$. Thus we expect to keep $2^{\kappa + k_o - t \cdot n_o}$ candidates and recover k_o more bits of the key.
8. The last step consists of an exhaustive search on the rest of the key. It is done in $\mathcal{T}_{exhaustive} = 2^{\kappa + 72 - n_p - k_s - k_o}$ Pride encryptions.

5.8 Optimal parameters and final complexity

We have tried different configurations of the different parameters and the different discussed steps, and the best complexity is given by considering Table 5. We therefore build $2^4 2^{29}$ structures of size 2^{28}. We take into account the equation of $Y_{17}^2[16]$ in the counter, by fixing the bits $C^2[11]$ and $C^3[11]$, which defines a partition for the key bits $K_0^2[11], K_0^3[11]$ and $K_0^4[11] \oplus K_{18}^2[3] \oplus K_{18}^2[11] \oplus K_{18}^2[12]$.

Objective	a	m	n_p	k_p	s	k_s	n_o	k_o	t	P_t
Time	2^4	2	1	3	3	10	3	23	13	$2^{-46.2}$
Data	9	2	1	3	3	16	3	17	9	$2^{-35.30}$

Table 5: Best time/data complexity parameters of our 18-round attack on Pride

Thus at the end of step 6, we keep $2^{66} \cdot 2^{-46.2} = 2^{19.80}$ key candidates for 67 bits of key information. We still have the state-test equations of the bits $Y_{17}^2[8]$, $Y_{17}^3[8]$ and $Y_{17}^3[16]$ and as explained in subsection 5.6, for half of the equations of $Y_{17}^3[8]$ and $Y_{17}^3[16]$, we have two double equations. Thus we have to solve the systems of equations by trying all the possible values for those "double" equations.

We can do this with a complexity of $(10 \times 2^3)^2 \times 2^{19.80} \approx 2^{12.64} \cdot 2^{19.80} = 2^{32.44}$ operations and we can discard one of the double equations since both can not be right. Moreover, the probability for a wrong key to have a solution is $2^{23-2\times10-13} = 2^{-10}$ since we have a system of $2 \times 10 + 13$ equations in 23 variables. Thus we expect to keep $10 \times 2^{19.80-7} = 2^{16.12}$ candidates and we have recovered 23 more bits of the key. Thus the time complexity of the exhaustive search is $\mathcal{T}_{exhaustive} = 2^{16.12+38} = 2^{54.12}$. In the end, the data complexity of our attack is $\mathcal{D} = 2^{61}$ with a time complexity of $2^{57.83}$ which is the bottleneck of our attack and a memory complexity of 2^{35} to store the candidates.

Moreover some trade offs can be made by using the other equations before filtering the candidates to have a better signal to noise ratio and to have a data complexity slightly smaller, but a bigger time complexity. For example if we fix the plaintexts bits of the state $Y_{17}^2[16]$ and we guess the bits of the states $X_2^3[8], X_2^3[16], X_2^2[16]$ and $Y_{17}^3[8]$, the parameters are given in Table 5. And the exhaustive search becomes the bottleneck of the time complexity with a cost of $\mathcal{T} = 2^{65.86}$ Pride encryptions, the data complexity is $\mathcal{D} = 2^{60.17}$ and the memory complexity is 2^{34} to store the candidates.

6 Application on the 12-round attack on Serpent

In this section, we give a summary of the improvements of the key recovery of the differential part of the attack in [11] on the block cipher Serpent. The details of the procedure are given in Supplementary Material B.

6.1 Summary

In the differential path of the attack in [11], some active bits need the values of inactive bits to be computed. Thus the authors used the properties of the s-boxes and binary decision trees as proposed in [12] to optimize the amount of key bits guessed and improve the naive approach to the key recovery part of their original attack. Those techniques mainly helped to find the smartest way to guess the needed key bits and reduced the overall guess of the key information

in the differential key recovery part. Our idea was thus to use the state-test technique to only guess the bits needed, to compare it with the decision trees technique. For the attack with the best time complexity, we need to guess 11 bits less than in the original attack reducing the time complexity of the key recovery part (being therefore to perform better than the previous technique for this part) but not the overall time complexity of the attack as the bottleneck is the final exhaustive search. For the attack with the best data complexity, we need to guess 12 bits less than in the original attack and this time it improves the time complexity of a factor $2^{2.93}$.

7 Conclusion

As we have seen, the state-test technique has many applications. Not only can it be applied in the context of (truncated-)differential MitM and impossible-differential cryptanalysis, but also to other scenarios. While it can be more complicated to apply it to some families of attacks, as it can be the case for differential attacks with a counter, we were able to show examples where its application leads to significantly better complexities, and even increase the number of rounds covered.

Acknowledgements. This research is funded by the European Union (ERC-2023-COG, SoBaSyC, 101125450). 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

1. Ahmadian, Z., Khalesi, A., M'foukh, D., Moghimi, H., Naya-Plasencia, M.: Improved differential meet-in-the-middle cryptanalysis. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14651, pp. 280–309. Springer (2024)
2. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçin, T.: Block ciphers - focus on the linear layer (feat. PRIDE). In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 57–76. Springer (2014)
3. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: The 128-bit block cipher *Camellia*. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **85-A**(1), 11–24 (2002)
4. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA,

August 14-18, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016)
5. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. IACR Trans. Symmetric Cryptol. **2019**(1), 5–45 (2019)
6. Biham, E., Anderson, R.J., Knudsen, L.R.: Serpent: A new block cipher proposal. In: Vaudenay, S. (ed.) Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1372, pp. 222–238. Springer (1998)
7. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer (1990)
8. Boura, C., David, N., Derbez, P., Leander, G., Naya-Plasencia, M.: Differential meet-in-the-middle cryptanalysis. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14083, pp. 240–272. Springer (2023)
9. Boura, C., Lallemand, V., Naya-Plasencia, M., Suder, V.: Making the impossible possible. J. Cryptol. **31**(1), 101–133 (2018). https://doi.org/10.1007/S00145-016-9251-7, https://doi.org/10.1007/s00145-016-9251-7
10. Boura, C., Naya-Plasencia, M., Suder, V.: Scrutinizing and improving impossible differential attacks: Applications to clefia, camellia, lblock and simon. In: 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, pp. 179–199. Springer (2014)
11. Broll, M., Canale, F., David, N., Flórez-Gutiérrez, A., Leander, G., Naya-Plasencia, M., Todo, Y.: New attacks from old distinguishers improved attacks on serpent. In: Galbraith, S.D. (ed.) Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13161, pp. 484–510. Springer (2022)
12. Broll, M., Canale, F., Flórez-Gutiérrez, A., Leander, G., Naya-Plasencia, M.: Generic framework for key-guessing improvements. In: Tibouchi, M., Wang, H. (eds.) 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. Lecture Notes in Computer Science, vol. 13090, pp. 453–483. Springer (2021). https://doi.org/10.1007/978-3-030-92062-3_16, https://doi.org/10.1007/978-3-030-92062-3_16
13. Cid, C., Huang, T., Peyrin, T., Sasaki, Y., Song, L.: Boomerang connectivity table: A new cryptanalysis tool. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 683–714. Springer (2018). https://doi.org/10.1007/978-3-319-78375-8_22, https://doi.org/10.1007/978-3-319-78375-8_22
14. Dunkelman, O., Indesteege, S., Keller, N.: A differential-linear attack on 12-round serpent. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) Progress in Cryptol-

ogy - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5365, pp. 308–321. Springer (2008)
15. Dunkelman, O., Sekar, G., Preneel, B.: Improved meet-in-the-middle attacks on reduced-round DES. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4859, pp. 86–100. Springer (2007)
16. Flórez-Gutiérrez, A., Todo, Y.: Improving linear key recovery attacks using walsh spectrum puncturing. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14651, pp. 187–216. Springer (2024)
17. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for aes-like permutations. 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. 365–383. Springer (2010)
18. Hadipour, H., Eichlseder, M.: Autoguess: A tool for finding guess-and-determine attacks and key bridges. In: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13269, pp. 230–250. Springer (2022)
19. Hadipour, H., Sadeghi, S., Eichlseder, M.: Finding the impossible: Automated search for full impossible-differential, zero-correlation, and integral attacks. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14007, pp. 128–157. Springer (2023). https://doi.org/10.1007/978-3-031-30634-1_5, https://doi.org/10.1007/978-3-031-30634-1_5
20. Knudsen, L.: Deal-a 128-bit block cipher. complexity **258**(2), 216 (1998)
21. Lallemand, V., Rasoolzadeh, S.: Differential cryptanalysis of 18-round PRIDE. In: Patra, A., Smart, N.P. (eds.) Progress in Cryptology - INDOCRYPT 2017 - 18th International Conference on Cryptology in India, Chennai, India, December 10-13, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10698, pp. 126–146. Springer (2017)
22. Naya-Plasencia, M.: How to improve rebound attacks. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6841, pp. 188–205. Springer (2011)
23. Sasaki, Y., Wang, L., Wu, S., Wu, W.: Investigating fundamental security requirements on whirlpool: Improved preimage and collision attacks. In: Wang, X., Sako, K. (eds.) Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7658, pp. 562–579. Springer (2012)
24. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher CLEFIA (extended abstract). In: Biryukov, A. (ed.) Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg,

March 26-28, 2007, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4593, pp. 181–195. Springer (2007)
25. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Trans. Computers **34**(1), 81–85 (1985)
26. Song, L., Yang, Q., Chen, Y., Hu, L., Weng, J.: Probabilistic extensions: A one-step framework for finding rectangle attacks and beyond. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14651, pp. 339–367. Springer (2024)
27. Yang, Q., Hu, L., Sun, S., Qiao, K., Song, L., Shan, J., Ma, X.: Improved differential analysis of block cipher PRIDE. In: López, J., Wu, Y. (eds.) Information Security Practice and Experience - 11th International Conference, ISPEC 2015, Beijing, China, May 5-8, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9065, pp. 209–219. Springer (2015)
28. Zhao, J., Wang, X., Wang, M., Dong, X.: Differential analysis on block cipher PRIDE. IACR Cryptol. ePrint Arch. p. 525 (2014)

Supplementary Material

A More Details on the Attacks on Craft

A.1 How to Handle the State-Test Guesses

In a naive approach we would need to make 19 state-test guesses (for the 25-round attack) in order to compute $\hat{P} = E_{in}^{-1}(E_{in}(P) + \delta)$ from P and for a difference δ. While the probabilistic key-guessing approach ultimately leads to this not affecting the number of candidates, we would need to make all the guesses during the computation of $E_{in}(P)$, and would only be able to apply the filter we get from the probabilistic key-recovery approach during the application of E_{in}^{-1}. As this would lead to a time complexity worse than brute-force, we use a different approach.

The basic idea is that only part of the state is involved in the calculation of the difference $P + \hat{P}$: starting at W_1, we are only interested in the green and orange nibbles $W_1[1, 6, 10, 14]$. So, instead of first computing E_{in} and then E_{in}^{-1}, we can create a list of all pairs $(W_1[1, 6, 10, 14], \hat{W}_1[1, 6, 10, 14])$, of which there are 2^{32} only, and propagate them through E_{in}. In each round, we first filter those pairs that do not yield the right difference after MC and then add all possible choices for the state-test guess. Note that in the last round of E_{in} we have no state-test guess, meaning that the number of pairs reduces to 2^{28}. Doing this computation separately for each value of δ allows us to further reduce the number of pairs to 2^{12}.

As we use the keys $K_0[6, 10, 14]$ and $K_1[4, 8, 12]$ during this computation, we perform this computation after this part of the key has been determined. Hence, for each δ and (partial) key $K_0[6, 10, 14]$ and $K_1[4, 8, 12]$ we can compute a list of pairs $(W_1[1, 6, 10, 14], \hat{W}_1[1, 6, 10, 14])$, associated state-test guesses and values of $X_{21}[4, 8]$ that correspond to the upper part of the attack. Then, given a plaintext P and a guess of k_{in} we can compute $W_1[1, 6, 10]$, look-up all candidates for $W_1[14]$ and $\hat{W}_1[1, 6, 10, 14]$ and compute $\hat{P}$ by making use of the difference $(W_1 + \hat{W}_1)[1, 6, 10, 14]$.

The overall time complexity of computing all these lists is $2^{16+24+32} = 2^{72}$, while we need 2^{12} memory to store one of them, meaning that this step has no significant impact on the complexity of the attack.

A.2 Overview of the 25-round attack

We give the pseudo-code of our 25-round attack in Algorithm 1.

A.3 Core of the 26-Round Attack on Craft

The core of our 26-round attack on Craft is shown in Fig. 6.

Algorithm 1 Truncated differential Meet-in-the-Middle attack using the state-test and 0-round distinguisher technique, as well as a 2-round structure

Require: Spaces $\mathcal{K}_{in}, \mathcal{K}_{out}, \mathcal{K}_{\cap}$ representing the key information needed in E_{in}, E_{out} and the parts they have in common, respectively, in addition to oracle access to E and E^{-1}. Further, a set of differences Δ at the boundary of upper and lower part and two spaces $\mathcal{F}$ and $\mathcal{S}$ such that $f \in \mathcal{F}$ corresponds to the fixed values of the structure $f + \mathcal{S}$.

for $\delta \in \Delta$ **do** ▷ $|\Delta| = 2^{16}$
  for $k_{\cap} \in \mathcal{K}_{\cap}$ **do** ▷ $|\mathcal{K}_{\cap}| = 2^{28}$
    Compute a list L that, given $W_1[1, 6, 10]$, yields all choices of $W_1[14]$, $W_1'[1, 6, 10, 14]$, associated state-test guesses and values of $X_{21}[4, 8]$ in compliance with the upper part of the attack.
    for $f \in \mathcal{F}$ **do** ▷ $|\mathcal{F}| = 2^{16}$
      Compute $Z_{24}'[9]$
      Initialize hash table H
      for $C \in f + \mathcal{S}$ s.t. $Z_{24}'[0] = 0$ **do** ▷ $2^{-4} \cdot |\mathcal{S}| = 2^{44}$
        $P \leftarrow E^{-1}(C)$
        for $k_{in} \in \mathcal{K}_{in} \setminus \mathcal{K}_{\cap}$ **do** ▷ $|\mathcal{K}_{in} \setminus \mathcal{K}_{\cap}| = 2^{12}$
          Compute $W_1[1, 6, 10]$
          for $W_1[14], \hat{W}_1[1, 6, 10, 14], X_{21}[4, 8]$ and state-test guesses g that correspond to $W_1[1, 6, 10]$ in L **do** ▷ $2^{12-12} = 1$ *(avg.)*
            Compute $\hat{P}$ and $\hat{C} \leftarrow E(\hat{P})$
            if $\hat{C} \oplus C$ have the 16-bit difference to 0 **then** ▷ 2^{-16}
              $\alpha \leftarrow (W_{23} + \hat{W}_{23})[8]$
              $\beta \leftarrow (W_{23} + \hat{W}_{23})[9]$
              if $(W_{23} + \hat{W}_{23})[3, 13]$ can both be α **then** ▷ 2^{-2}
                if $(W_{23} + \hat{W}_{23})[2, 12]$ can both be β **then** ▷ 2^{-2}
                  $A \leftarrow (\hat{Z}_{24}'[1, 6, 10], Y_{23}'[9], (Y_{23}' + \hat{Y}_{23}')[8, 14, 15], X_{21}[4, 8])$
                  $H[A] \leftarrow (k_{in}, g, C, \hat{C})$
      for $k_{out} \in \mathcal{K}_{out} \setminus \mathcal{K}_{\cap}$ **do** ▷ $|\mathcal{K}_{out} \setminus \mathcal{K}_{\cap}| = 2^{8}$
        Compute $Y'23[1, 6, 10]$ based on f
        for All choices of the remaining 12 nibbles of $Y'23$ **do** ▷ 2^{48}
          Compute $B \coloneqq (\hat{Z}_{24}'[1, 6, 7, 10], (Y_{23}' + \hat{Y}_{23}')[8, 14, 15], X_{21}[4, 8])$
          for $(k_{in}, g, C, \hat{C}) \in H(B)$ **do** ▷ $2^{(44+12-16-4)-(16+12+8)} = 1$ *(avg.)*
            Filter further using the now known value of $K_{24}[13]$ ▷ 2^{-4}
            Recover remaining key using structure and state-test equations
            if No contradiction found **then** ▷ $2^{-4 \cdot (23-2-4)} = 2^{-68}$
              Test key using trial decryption

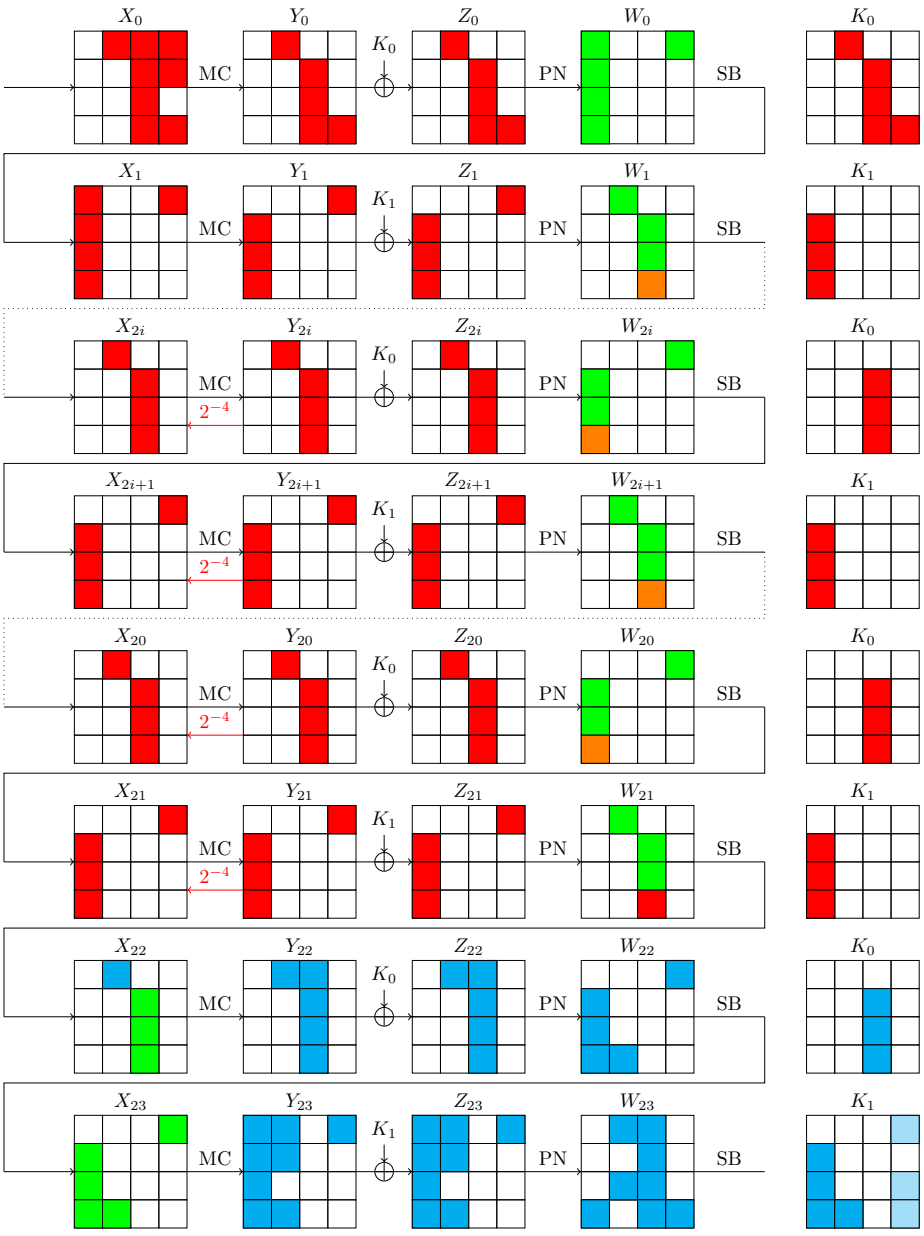


Fig. 6: Core of the 26-round attack on CRAFT. Green indicates that the values are known, while orange indicates the state-test guesses. Red and blue indicate the differential propagation in the upper and lower parts, respectively. The gray nibbles in K_{23} indicate that their difference is guessed.

A.4 Recovering the Remaining Part of the Key in the 26-Round Attack

After the matching phase and using the (differentially) active nibbles in the structure whose difference we did not use for filtering so far to recover all but the 12 nibbles $K_0[3,5,7,8,11,13]$ and $K_1[2,6,7,9,11,14]$ of the key, we continue using the state-test equations. First, we use the equation

$$W_1[14] = K_1[3] + S(K_0[0] + X_0[0] + X_0[12] + X_0[8]) + S(K_0[14] + X_0[14]) + S(K_0[7] + X_0[15] + X_0[7]),$$

to recover $K_0[7]$, which gives us $K_1[11]$ by using the two-round structure. We can now verify the difference that we formerly just checked to be among the possible ones to get a filter of 2^{-3}. At this point, the state-test equation

$$W_{21}[14] = S(K_0[14] + S(K_1[11] + K_1[15] + K_1[3] + W_{23}[0] + W_{23}[14] + W_{23}[7]))$$

consists of known values only, increasing our filter to $2^{-3-4} = 2^{-7}$. Next, we use one of the remaining differences in the structure to determine $K_0[3]$ and $K_1[14]$. As we already used this one for filtering out impossible differences, this decreases our filter to $2^{-7+1} = 2^{-6}$. Then, we can use the two state-test equations

$$W_2[12] = K_0[1] + S(K_1[12] + S(K_0[1] + X_0[1] + X_0[13] + X_0[9])) + S(K_1[2] + S(K_0[13] + X_0[13]) + S(K_0[3] + X_0[11] + X_0[15] + X_0[3]) + S(K_0[4] + C_0[0] + X_0[12] + X_0[4])) + S(K_1[5] + C_1[1] + S(K_0[2] + X_0[10] + X_0[14] + X_0[2]) + S(K_0[9] + X_0[9])),$$

$$X_{22}[1] = K_0[1] + K_0[13] + K_0[9] + S(K_1[10] + K_1[14] + K_1[2] + W_{23}[13] + W_{23}[3] + W_{23}[4]) + S(K_1[12] + W_{23}[1]) + S(K_1[13] + K_1[5] + C_{21}[1] + W_{23}[2] + W_{23}[9])$$

and the equation from the structure to guess either $K_0[13]$ or $K_1[2]$, determine the other and filter further, increasing our total filter to $2^{-6+4-8} = 2^{-10}$. Next, we guess either $K_0[8]$ or $K_1[6]$, determine the other using the structure and use the state-test equation

$$W3[14] = K_1[3] + S(K_0[0] + S(K_1[1] + S(K_0[12] + X_0[12]) + S(K_0[2] + X_0[10] + X_0[14] + X_0[2]) + S(K_0[5] + C_0[1] + X_0[13] + X_0[5])) + S(K_1[15] + S(K_0[0] + X_0[0] + X_0[12] + X_0[8])) + S(K_1[6] + S(K_0[3] + X_0[11] + X_0[15] + X_0[3]) + S(K_0[8] + X_0[8]))) + S(K_0[14] + S(G[0])) + S(K_0[7] + S(K_1[0] + S(K_0[1] + X_0[1] + X_0[13] + X_0[9]) + S(K_0[15] + X_0[15]) + S(K_0[6] + X_0[14] + X_0[6])) + S(K_1[11] + S(K_0[7] + X_0[15] + X_0[7])))$$

to recover $K_0[5]$, which in turn gives us $K_1[9]$ by using the structure and decreases our total filtering capability to $2^{-10+4} = 2^{-6}$. We can now filter once more using the state-test equation

$$W_{19}[14] = S(K_0[14] + S(W_{21}[14] + K_1[11] + K_1[15] + K_1[3] + S(K_0[0] + K_0[12] + K_0[8] + S(K_1[1] + K_1[13] + K_1[9] + W_{23}[12] + W_{23}[2] + W_{23}[5]) + S(K_1[14] + K_1[6] + W_{23}[3] + W_{23}[8]) + S(K_1[15] + W_{23}[0])) + S(K_0[15] + K_0[7] + S(K_1[0] + K_1[12] + K_1[8] + W_{23}[1] + W_{23}[15] + W_{23}[6]) + S(K_1[11] + W_{23}[7]))))$$

that contains known values only and guess one of the remaining nibbles $K_0[11]$ and $K_1[7]$ to recover the other using the structure. In total, this shows that we can recover the remaining key with a time complexity lower than the number of candidates, and we can filter even more using the remaining state-test equations.

B Application on the 12-round attack on Serpent

In [11], the authors presented the best known attacks on the cipher Serpent. Their attack reach 12 rounds of the cipher and they use a differential-linear distinguisher on 8 rounds that they extend to a partial 10-round distinguisher with correlation $2^{60.75}$ as shown on Fig. 7. To lower the information guessed on the key, they used some conditional linear properties of the s-box in the differential key recovery part. Those conditions could be used because not all the output or input of the s-boxes needed to be known. Our idea is to use the state-test technique in the key recovery part of their attack to reduce furthermore the time complexity of recovering the candidates. We would like to point out that further improvements of this attacks using techniques from [16] might be possible, but we find this to be out of the scope of the paper.

B.1 A brief description of the Serpent cipher

Serpent is a 32-round block cipher introduced in [6] by Anderson, Biham and Knudsen and was a finalist in the AES competition. The Serpent family operates on block of size 128 bits with a key of size k, where $k \in \{128, 192, 256\}$. The state is a 4×32 matrix of bits, in other words the state is organized in four 32-bit words $X_0, \cdots, X_3$ and we note $X_j[i]$ the i-th leftmost bit of word j. The round function is composed of the three operations :

- **Key mixing.** A 128-bit subkey is XORed to the internal state. The subkey at round i will be denoted by K_i.
- **s-box Layer.** 32 copies of an s-box is applied to the internal state. The Serpent cipher use eight different s-boxes at round i, we apply the s-box $S_{(i mod 8)}$.
- **Linear transformation.** The linear operation LT consist of the following operations :

$$
\begin{aligned}
&X_0 \leftarrow X_0 \lll 13; X_2 \leftarrow X_2 \lll 3\\
&X_1 \leftarrow X_1 \oplus X_0 \oplus X_2; X_3 \leftarrow X_3 \oplus X_2 \oplus (X_0 \ll 3)\\
&X_1 \leftarrow X_1 \lll 1; X_3 \leftarrow X_3 \lll 7\\
&X_0 \leftarrow X_0 \oplus X_1 \oplus X_3; X_2 \leftarrow X_2 \oplus X_3 \oplus (X_1 \ll 7)\\
&X_0 \leftarrow X_0 \lll 5; X_2 \leftarrow X_2 \lll 22
\end{aligned}
$$

Here $\ll j$ denotes a j-bit left shift and $\lll j$ denotes a j-bit left rotation. In round 31 this linear transformation is omitted and an additional subkey is XORed to the state instead.

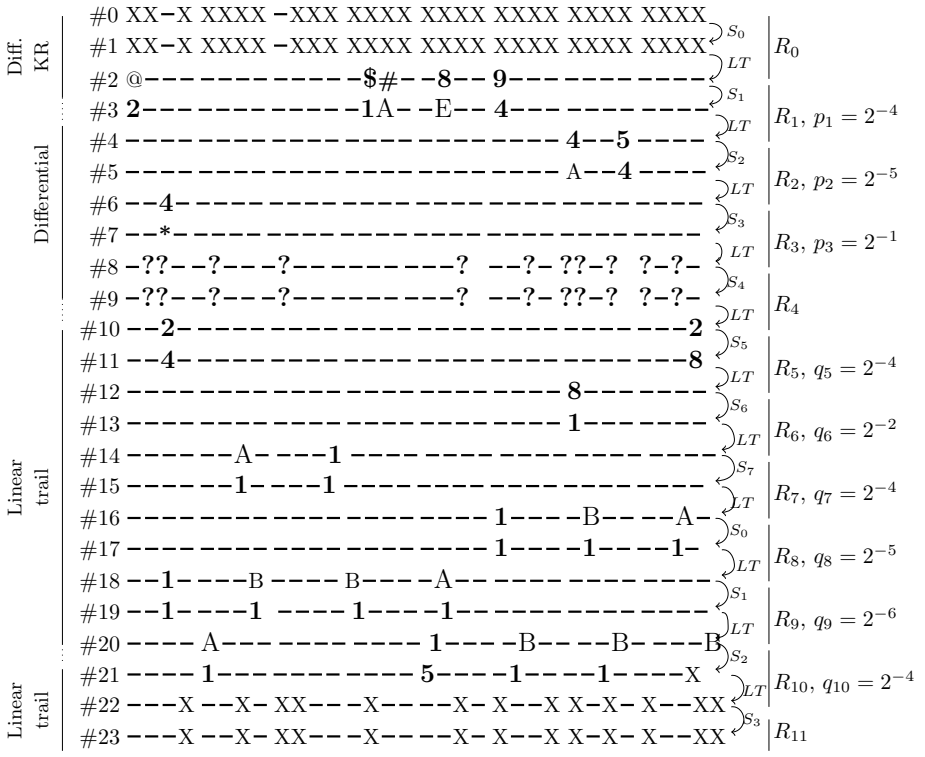


Fig. 7: The differential-linear attack on 12-round Serpent with staggered key-recovery from [11]. The nibble difference \$ is undetermined but is one in the leftmost bit x_3 because of the differential properties of S_1. The nibble difference ***** is undetermined but is zero in the rightmost bit x_0. The nibble difference @ is undetermined but is zero in the rightmost bit x_0. The nibble difference # is undetermined but is zero in the leftmost bit x_3.

B.2 The previous 12-round attack

The attack in [11] is based on the same 9-round differential-linear distinguisher that was used in [14]. As described in Fig. 7, their attack begin a "central" distinguisher which starts at round 2 with a 3-round differential of probability$p = 2^{-6}$ with input difference $\Omega_P = 00000000000000000000000040050000_x$ and then ends with a five-round linear approximation with correlation $q = 2^{-21}$ with output mask $\lambda_C = 00001000000000005000010000100001_x$. The expected correlation for the differential-linear distinguisher is thus $pq^2 = 2^{-48}$. However, experiments with the correlation of the transition rounds between the differential and the linear trail suggest that the actual correlation should be at least $2^{-48.75}$.

In their attacks, the authors partially extend the distinguisher by adding one round in the differential part and one round in the linear part. The distinguisher has now a correlation of $2^{60.75}$ and start at round 1 and finish at round 10. Fig. 7

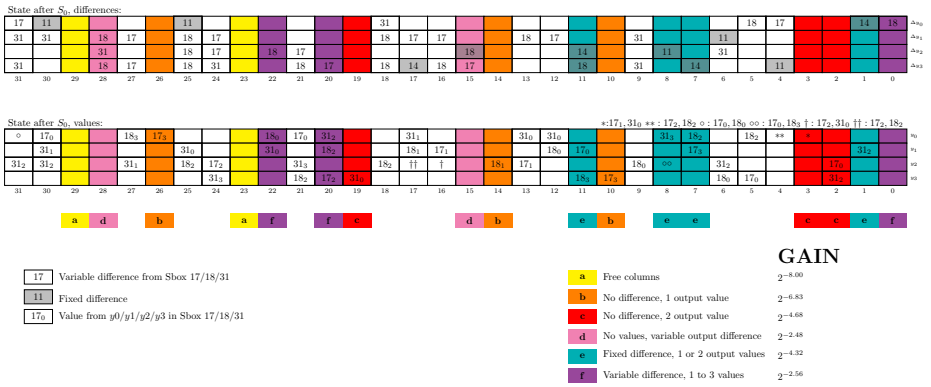


Fig. 8: Reducing the required key bit guesses in the first round in the attack of [11].

give an overview of their attack. We notice that we need to know almost all the bits of K_0 and the bits of K_1 in column 17,18 and 31 to verify that the pair follow the differential distinguisher thus we need to know 132 bits of subkeys. Moreover with a data complexity of $2^{127.92}$, an advantage of 15 bits is obtained with probability 0.1. Thus we have a data and memory complexities of $2^{127.92}$ for a time complexity of $2^{252.46}$.

To reduce the complexity of their attack, the authors used the properties of the s-boxes and used binary decision trees to optimize the amount of key bits guessed. Indeed, in Fig. 8, we notice that for some column we don't need to know the exact value of all the bits of either the input or the output of the s-boxes or we need to know the exact value of the output of the s-box but the inputs bit aren't active, i.e. there is no differences in the output bits of the s-box. Moreover there is linear conditions in the s-boxes used in Serpent, in particular in S_0, that the authors used to compute some values of the output without needing to know the whole input. To use those information, the authors, through the binary decision trees representing the conditions of the s-box, find the least amount of key information we need to know to be able to compute the values of the bits we are interested in. Thus the complexity of the attack decreases. The authors also use conditional proprieties of the s-boxes S_2 to improve the overall correlation by a factor $2^{1.66}$ and thus improve the linear key recovery since the advantage increase to 23 by keeping the same data complexity as before. However the number of active bits in the last round is now 56.

Thus the data and memory complexities of this attack is still $2^{127.92}$. In the naive approach of the attack, we need to know all the bits of K_0 and the bits of K_1 in column 17,18 and 31 to verify that the pair follow the differential distinguisher. Thus we need to guess $128+12 = 140$ bits of the two first subkeys. In Fig. 8, we can see the gain of using conditional relations of the s-box. Thus

now, the time complexity of the key recovery part is

$$2^{140-8-6.83-4.68-2.48-4.32-2.56} \cdot (2^{123.92-3.5} + 2^4 \cdot 112 \cdot 2^{112} \cdot 2^{-4.17}) \simeq 2^{231.91}$$

equivalent encryptions while the exhaustive search has a cost of 2^{256-23}. The total time complexity is thus around $2^{233.55}$ 12-round encryptions.

B.3 Improving with the State-test technique

Our idea is to use the state-test technique to further reduce the amount of key information we need to guess. As seen in Fig. 8, in column of type a, b and c, there is no output differences but in column of type b and c we need to know the exact values of some the output bits to compute the exact values of the input bits of the s-boxes in column $17, 18$ and 31. Thus here instead of using the conditional relations of the s-boxes, we directly guess the value of the internal state that we need to know with the state-test technique which decrease the amount of bits that need to be guessed and we obtain the information on the subkeys with non-linear equations.

In this improvement, the column of type a, d, e and f will be handled in the same way as the previous attack and we will have the same gain for those column. However, we use the state-test technique to guess the following bits of the columns of type b and c, in round $1 : 17_0^1$, 17_1^1, 17_2^1,17_3^1,18_0, 18_1^1, 31_0^1 and 31_2^1. Thus instead of guessing the $6 \times 4 = 24$ bits of K_0 to know the values of the bit in column 2,3,10,14,19 and 26 after S_0 and the 8 bits of K_1, we now guess only 7 bits of the internal states.

With the state-test technique, we obtain the following equations :

Equation 0:

$$\begin{aligned}0 = f_0 \oplus K_1^{68} \oplus SB(X_0^{20,21,22,23} \oplus K_0^{20,21,22,23})_3 \oplus SB(X_0^{8,9,10,11} \oplus K_0^{8,9,10,11})_2 \oplus \\ SB(X_0^{32,33,34,35} \oplus K_0^{32,33,34,35})_2 \oplus SB(X_0^{44,45,46,47} \oplus K_0^{44,45,46,47})_1 \oplus \\ SB(X_0^{84,85,86,87} \oplus K_0^{84,85,86,87})_0 \oplus SB(X_0^{120,121,122,123} \oplus \\ K_0^{120,121,122,123})_0 \oplus SB(X_0^{124,125,126,127} \oplus K_0^{124,125,126,127})_0\end{aligned}$$

Equation 1:

$$\begin{aligned}0 = f_1 \oplus K_1^{69} \oplus SB(X_0^{12,13,14,15} \oplus K_0^{12,13,14,15})^0 \oplus SB(X_0^{52,53,54,55} \oplus \\ K_0^{52,53,54,55})^2 \oplus SB(X_0^{64,65,66,67} \oplus K_0^{64,65,66,67})^1\end{aligned}$$

Equation 2:

$$\begin{aligned}0 = f_2 \oplus K_1^{70} \oplus SB(X_0^{16,17,18,19} \oplus K_0^{16,17,18,19})^0 \oplus SB(X_0^{24,25,26,27} \oplus \\ K_0^{24,25,26,27})^0 \oplus SB(X_0^{64,65,66,67} \oplus K_0^{64,65,66,67})^2 \oplus SB(X_0^{68,69,70,71} \oplus \\ K_0^{68,69,70,71})^2 \oplus SB(X_0^{76,77,78,79} \oplus K_0^{76,77,78,79})^1 \oplus SB(X_0^{80,81,82,83} \oplus \\ K_0^{80,81,82,83})^3 \oplus SB(X_0^{96,97,98,99} \oplus K_0^{96,97,98,99})^2\end{aligned}$$

Equation 3:

$$\begin{aligned}0 = f_3 \oplus K_1^{71} \oplus SB(X_0^{28,29,30,31} \oplus K_0^{28,29,30,31})^2 \oplus SB(X_0^{40,41,42,43} \oplus \\ K_0^{40,41,42,43})^3 \oplus SB(X_0^{104,105,106,107} \oplus K_0^{104,105,106,107})^0\end{aligned}$$

Equation 4:

$$0 = f_4 \oplus K_1^{72} \oplus SB(X_0^{24,25,26,27} \oplus K_0^{24,25,26,27})_3 \oplus SB(X_0^{12,13,14,15} \oplus K_0^{12,13,14,15})_2 \oplus SB(X_0^{j36,37,38,39} \oplus K_0^{36,37,38,39})_2 \oplus SB(X_0^{48,49,50,51} \oplus K_0^{48,49,50,51})_1 \oplus SB(X_0^{88,89,90,91} \oplus K_0^{88,89,90,91})_0 \oplus SB(X_0^{124,125,126,127} \oplus K_0^{124,125,126,127})_0 \oplus SB(X_0^{0,1,2,3} \oplus K_0^{0,1,2,3})_0$$

Equation 5:

$$0 = f_5 \oplus K_1^{73} \oplus SB(X_0^{16,17,18,19} \oplus K_0^{16,17,18,19})^0 \oplus SB(X_0^{56,57,58,58} \oplus K_0^{56,57,58,59})^2 \oplus SB(X_0^{68,69,70,71} \oplus K_0^{68,69,70,71})^1$$

Equation 6:

$$0 = f_6 \oplus K_1^{124} \oplus SB(X_0^{76,77,78,79} \oplus K_0^{76,77,78,79})_3 \oplus SB(X_0^{88,89,90,91} \oplus K_0^{88,89,90,91})_2 \oplus SB(X_0^{64,65,66,67} \oplus K_0^{64,65,66,67})_2 \oplus SB(X_0^{100,101,102,103} \oplus K_0^{100,101,102,103})_1 \oplus SB(X_0^{48,49,50,51} \oplus K_0^{48,49,50,51})_0 \oplus SB(X_0^{52,53,54,55} \oplus K_0^{52,53,54,55})_0$$

Equation 7:

$$0 = f_7 \oplus K_1^{126} \oplus SB(X_0^{80,81,82,83} \oplus K_0^{80,81,82,83})^0 \oplus SB(X_0^{120,121,122,123} \oplus K_0^{120,121,122,123})^2 \oplus SB(X_0^{124,125,126,127} \oplus K_0^{124,125,126,127})^2 \oplus SB(X_0^{4,5,6,7} \oplus K_0^{4,5,6,7})^1 \oplus SB(X_0^{8.9.10.11} \oplus K_0^{8,9,10,11})^3 \oplus SB(X_0^{24,25,26,27} \oplus K_0^{24,25,26,27})^2$$

Gain factor In column of type b, only one bit of the output is needed. In column 26, the first output bit is needed to compute the bit 17_3. In column 14, y_3 is needed to compute 18_1 and in column 10, y_3 is needed to compute 17_3. In column of type c, two bits of the output are needed. In column 19, we need the bits y_1 and y_3 to compute 17_2 and 31_0 respectively. In column 3, we need the bits y_0 to compute 17_1 and 31_0 and y_3 to compute bit 18_0. In column 2, we need to know the bits y_2 and y_3 to compute 17_0 and 31_2 respectively. And as in the previous paper, we can absorb the last bit guess for some of the columns into the next round. Thus here we have a gain factor of 2^{-24} since we guess 8 bits instead of the $6 \times 4 + 8 = 32$ bits to compute those 8 bits without the improvement.

The data and memory complexities don't change and is equal to $2^{127.92}$ and the time complexity of the key recovery part is now :

$$2^{140-8-24-2.48-4.32-2.56} \cdot (2^{123.92-3.5} + 2^4 \cdot 112 \cdot 2^{112} \cdot 2^{-4.17}) \simeq 2^{219.06}.$$

Improving the time complexity of the attack with the best data complexity In [11], the authors have presented another attack on Serpent by also determining the input values of the s-box at column 11 which will reduce the overall correlation and thus allowing to make the data complexity smaller. In this case there will then be no conditions imposed in the differences of state # 2 of Fig. 7 which will increase the active bits in the differential key recovery part. Now there will be one column of type a (23), three columns of type c (2,3 and 19), three columns of type $\mathbf{e}$ (0, 22, 7 and 8), and four columns of type f (12, 22, 26 and 29). The authors compute the gain factor the same way as before, and they obtain for column

29: $2^{-1.607}$, column 26: $2^{-1.192}$, column 23: 2^{-4}, column 22: $2^{-1.35}$, column 19: $2^{-1.41}$, column 12: 2^{-1}, column 8: $2^{-1.678}$, column 7: $2^{-1.41}$, column 3: $2^{-0.415}$, column 2: $2^{-0.415}$ and column 0: $2^{-1.54}$. for a data and memory complexity of $2^{118.40}$, which gives an advantage of 16 bits, we have a time complexity of:

$$2^{128+16-16.023} \cdot (2^{118.4-3.5} + 2^4 \cdot 104 \cdot 2^{104} \cdot 2^{-4.17}) + 2^{256-16} \simeq 2^{242.93}$$

12-round encryptions.

As before, we can use the state-test technique for the columns of type c to reduce the amount of key information guessed during the differential key recovery part. Thus here we guess the bits $11_0, 17_0, 17_1, 17_2, 18_0, 31_0$ and 31_2 which give a gain of factor 2^{-12} instead of $2^{-2.24}$ for column of type c. And we have a time complexity of:

$$2^{128+16-25.777} \cdot (2^{118.4-3.5} + 2^4 \cdot 104 \cdot 2^{104} \cdot 2^{-4.17}) + 2^{256-16} \simeq 2^{240}.$$