Differential Cryptanalysis of PUFFIN and PUFFIN2

Céline Blondeau[1] * and Benoît Gérard[2]**

[1] Aalto University School of Science, Department of Information and Computer Science
[2] Université catholique de Louvain, UCL Crypto Group, ICTEAM Institute.
celine.blondeau@aalto.fi; ben.gerard@uclouvain.be

Abstract. A sound theoretical framework for analyzing multiple differential cryptanalysis was introduced and applied to reduced-round PRESENT at FSE 2011. We propose here to apply it to the cryptanalysis of another lightweight block cipher namely PUFFIN. This cipher security has already been undermined by Leander for a quarter of the keys. In this paper we claim that both PUFFIN and its patched version PUFFIN2 can be broken by differential cryptanalysis faster than by exhaustive search and using less than the full code-book. We also improve the complexities of these attacks using multiple differentials. Particularly, we propose an attack on PUFFIN2 that recovers the 80-bit key in $2^{74.78}$ operations using $2^{52.3}$ chosen plaintexts.
Keywords: multiple differential cryptanalysis, lightweight cryptography, PUFFIN.

1 Introduction

Security and privacy in constrained environment is a challenging topic in cryptography. In this prospect many lightweight ciphers have been designed in the last few years. The most studied one is PRESENT that was proposed at CHES 2007 [1]. The popularity of this cipher may come from its very simple structure (the Substitution Permutation Network or SPN) together with the simplicity of its permutation layer that only consists in wire crossings. This cipher has been extensively studied and its security against statistical cryptanalyses has not been threatened yet. Unfortunately, this is not the case of all proposed lightweight ciphers. Indeed, some specifications include incorrect security analysis claiming resistance against classical statistical attacks namely Matsui's linear cryptanalysis [2] and Biham and Shamir's differential cryptanalysis [3]. A typical illustration is the PUFFIN cipher family. PUFFIN [4] and PUFFIN2 [5] are two recently proposed ciphers operating on 64-bit messages with respective key lengths 128 and 80. An extension of linear cryptanalysis has been applied to PUFFIN by Leander [6] and we propose in this paper to consider the weakness of both ciphers against differential cryptanalysis.

* This work was produced while at INRIA project-team SECRET, France
** Postdoctoral researcher supported by Walloon region MIPSs project.

The main source of incorrectness in the security analysis of PUFFIN and PUFFIN2 comes from the fact that advances in both linear and differential cryptanalysis have not been taken into account. For instance, at EUROCRYPT 2011, Leander [6] presented a new method to attack block ciphers considering the linear hull effect. Applying this method to PUFFIN he obtained an attack that recovers the master key with a data complexity of 2^{58} for a quarter of the keys while the designers claim that obtaining a gain over exhaustive search at least requires 2^{64} plaintext/ciphertext pairs. The same holds for differential attacks since only attacks using one differential are considered and since the probabilities of such differentials are underestimated by only looking at the most probable differential trail. Moreover, it is implicitly considered that the classical last-round attacks will use differentials on $r-1$ rounds (that is 31 out of 32 for PUFFIN and 33 out of 34 for PUFFIN2) while it turns out that we are able to perform attacks with differentials for $r-4$ and $r-5$ rounds of the cipher due to the slow diffusion of both the permutation layer and the key-schedule algorithm.

From a cryptanalytic point of view, the proposed attacks are very similar to the multiple differential cryptanalyses of PRESENT [7, 8]. The main difference comes from the fact that more than the two last rounds are partially inverted. This implies that the key-schedule may be exploited when partially deciphering samples to keep the time complexity small enough. This consideration is the main contribution of this paper regarding the state of the art in differential cryptanalysis. It also raises the question of using algebraic techniques at this point of the attack. This problem is closely related to the use of an algebraic wrong-pair detection as investigated in [9].

Using the theoretical framework presented in [8], we propose different attacks on both ciphers. For instance, we propose parameters to attack PUFFIN which is parametrisez by a 128-bit key with time complexity $2^{108.84}$ and data complexity $2^{49.42}$. More importantly, we also propose an attack on the patched PUFFIN2 version that recovers the 80-bit key in time $2^{74.78}$ using $2^{52.3}$ chosen plaintexts contradicting the security claims of the designers.

The remaining sections are organized as follows. First, in Section 2 we provide a detailed description of both PUFFIN and PUFFIN2 then, in Section 3 we discuss techniques for recovering key bits in a last-round attack. We also discuss techniques for performing this key-recovery part of the attack efficiently when many active S-boxes have to be considered and propose an estimate for the resulting time complexity. In Section 4 we present tools we use to analyze multiple differential attack complexities and provide some lower bounds for differential probabilities on PUFFIN and PUFFIN2. Finally, we detail our choice of parameters for the proposed attacks and provide the corresponding complexities in Section 5 before concluding in Section 6.

2 Description of PUFFIN and PUFFIN2

The lightweight cipher PUFFIN was introduced in [4] then upgraded to PUFFIN2 in [5]. Both ciphers are 64-bit SPN ciphers with round functions composed of a key addition, an S-box layer and a permutation. The order of the different components differs from a version to another but both the S-box and the permutation are the same for PUFFIN and PUFFIN2. The particularity of this cipher is that the permutation and the substitution layer are involutions, meaning that the same primitive is used for both encryption and decription process.

The initial number of key bits in PUFFIN was 128, it has been reduced to 80 in PUFFIN2 while the number of rounds has been increased from 32 to 34. The key-schedule also has been improved since it was linear in PUFFIN and is now highly non-linear in PUFFIN2. Notice that in addition to the 32 rounds of PUFFIN, a sub-key addition and a permutation are performed at the beginning. In PUFFIN2, the S-box layer is applied at the end of the 34 rounds (see Fig. 1). The round functions and key-schedules are detailed in the following subsections.

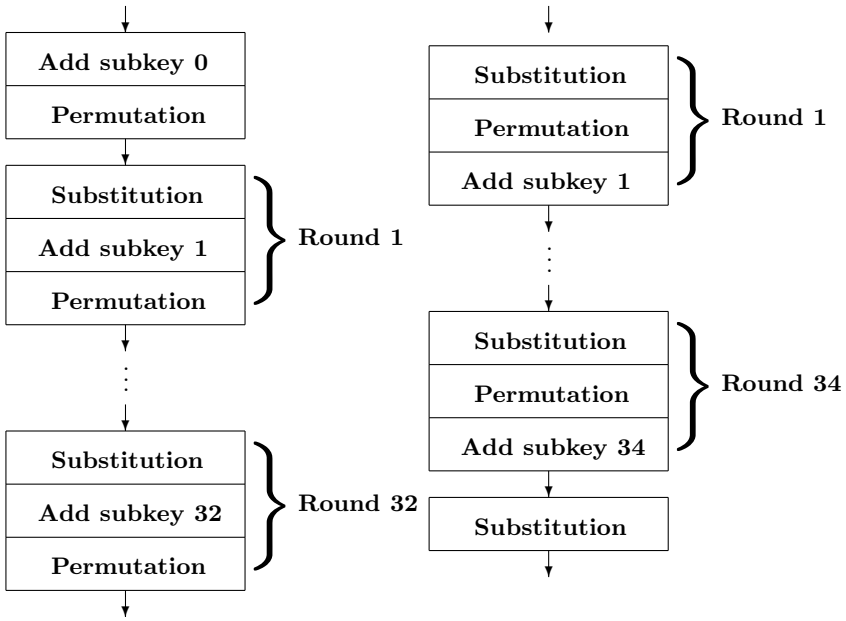

Fig. 1. PUFFIN (left) and PUFFIN2 (right) ciphers.

2.1 Round functions of PUFFIN and PUFFIN2

As mentioned earlier, the round functions are composed of a key addition, an S-box layer and a permutation layer. While in both PUFFIN and PUFFIN2 the state first passes through the S-box layer, the state is exclusively-ORed to

the round sub-key before being permuted in PUFFIN (SAP) when it is first permuted in PUFFIN2 (SPA). Both ciphers are depicted in Fig. 1.

For both versions of the cipher, the substitution layer is composed of 16 applications of a 4x4 S-box given by Table 6 in Appendix B. The permutation layer P is given by Table 7 in Appendix B. A round of PUFFIN is depicted in Fig. 2.

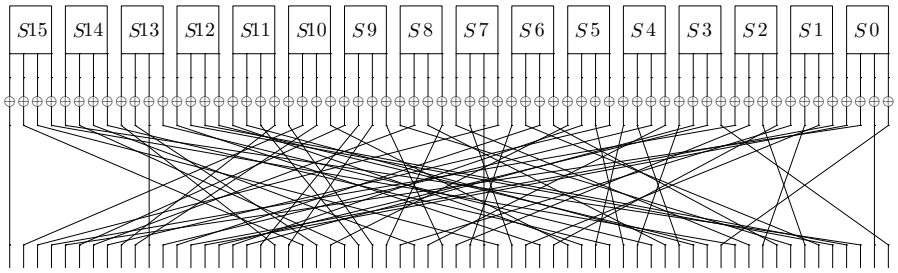

Fig. 2. One round of PUFFIN.

2.2 PUFFIN key-schedule

The key-schedule of PUFFIN is linear and operates on 128-bit master keys to generate 33 round-subkeys as follow.

1. First, a 128-bit state is initialized with the master key.
2. The 64-bit first-round subkey is extracted from the state using the selection table given in Table 8.
3. Steps 4 to 6 are iterated to obtain the remaining subkeys.
4. The state is updated using the permutation given in Table 9.
5. Bits $0, 1, 2$ and 4 are inverted excepted for rounds $2, 5, 6$ and 8.
6. A 64-bit round subkey is extracted from the state using Table 8.

2.3 PUFFIN2 key-schedule

The key-schedule in PUFFIN2 is not linear anymore but nevertheless remains simple since its structure follows the one of an SPN (without key addition of course). The key-size has been reduced here from 128 in PUFFIN to 80 bits. Then the state is processed as follows to generate 34 round sub-keys.

- First, an 80-bit state is initialized with the master key and passed through an S-box layer which consist in an application of the cipher S-box to each group of nibbles of the 80 bits.
- Then the following steps are iterated to obtain the 34 round-subkeys.

1. A 64-bit subkey is extracted by selecting the 64 leftmost bits or the 64 rightmost bits of the state depending on the round number (see Table 1).
2. The 64 bits of the state used to generate the key are passed through the cipher permutation P(64 leftmost or 64 rightmost depending on the round number).
3. The whole 80-bit state is non-linearly transformed by passing through a layer of S-boxes: adjacent bits are grouped by nibbles (4 bits) and the cipher S-box is applied to each of the 20 groups.

	Round numbers
64 leftmost bits	1,2,7,8,11,12,15,16,19,20,23,24,27,28,33,34
64 rightmost bits	3,4,5,6,9,10,13,14,17,18,21,22,25,26,29,30,31,32

Table 1. Bits of the state considered in the key-schedule.

3 Key-recovery in differential cryptanalysis

3.1 Last-round attacks: the differential case

Last-round attacks recover some key bits by peeling off one or more rounds (say r') of the cipher. The idea is to compute a statistic linked to the behavior of the cipher over r rounds thus targeting $r + r'$ rounds. This is done by partially deciphering some relevant part of the available ciphertexts over r' rounds. Hence, for each possible value of the key bits involved into this process, a statistic is obtained. These statistics translate into probabilities or likelihood values that induce at their turn an ordering of the subkey candidates from the most probable value for the correct subkey to the least one.

The main novelty of the attacks proposed in this paper lies in the fact that, by contrast with differential attacks on PRESENT, the number of peeled off rounds is larger (r' can be up to 5 rounds in the PUFFIN's case while it is equal to 2 in differentials attacks on PRESENT [7, 8]). The partial deciphering hence has a huge cost in time if no trick is used to reduce it. The context of differential cryptanalysis is particular in this prospect hence, as mentioned in the title of the section, we will focus on this case. We are first going to briefly recall the flow of differential cryptanalysis to clearly express the problem. To ease the understanding, we restrict ourselves to the particular setting where only one differential is used. The problem is quite similar when using more differentials hence we will discuss the main differences later on.

The cipher is divided into two parts: $E_{r+r'} = E_{r'} \circ E_r$ that is the first r rounds E_r that are considered when searching a good differential and the last r' rounds $E_{r'}$ that are partially inverted during the attack. The functions E have two variables: the first is the key used and the second one the message/internal

state. Pairs available to the attacker have been obtained using the correct key k_* that the attacker aims at recovering: $C = E_{r+r'}(k_*, P)$. The partial decryption using a candidate k is denoted by $E_{r'}^{-1}(k, C)$. For a given differential (δ_0, δ_r), the attack is detailed in Algorithm 1.

Algorithm 1: Last-round differential attack.

Input: a differential (δ_0, δ_r), plaintext/ciphertext pairs $(P, C = E_{r+r'}(k_*, P))$
Output: an ordered list of subkey candidates
Initialize a table D of size 2^{n_k} to 0, $D(k)$ corresponds to the counter for the subkey candidate k;
foreach *plaintext pair* (P, P') *such that* $P \oplus P' = \delta_0$ **do**
    foreach *subkey candidate* k **do**
        if $E_{r'}^{-1}(k, C) \oplus E_{r'}^{-1}(k, C') = \delta_r$ **then**
            $D(k) \longleftarrow D(k) + 1$;
return *candidates ordered according to* $D(k)$*;*

3.2 PUFFIN permutation layer diffusion

The permutation layer of PUFFIN has been designed to be hardware efficient (it uses only wire crossings) and to be an involution. This last point is the main difference with the permutation layer of PRESENT and has a major counterpart that is its slow diffusion. The full-diffusion of a cipher is reached when a bit-flip at any position of the input will influence all output bits[3]. In general, the smaller the number of rounds required to reach full-diffusion is, the more resistant the cipher should be (particularly against last-round attacks and impossible differential cryptanalysis). While using PRESENT permutation full-diffusion is reached after 3 rounds (which is optimal for a bit permutation and 16 4-bit S-boxes), 5 rounds of PUFFIN are required to obtain this property (see Fig. 3 for an example). The danger for last-round attacks is that a bad diffusion implies a small number of active S-boxes when inverting the last rounds. Hence, the attacker is able to increase r' that is decreasing r for the same number of targeted rounds, which in turns, implies the use of better statistical characteristics.

In this paper, we present attacks where only S-box $S11$ is active in the $r+1$-th round. It turns out that best differentials are the ones having only one active S-box in the last round. More particularly, output differences activating S-boxes $S3, S11$ and $S12$ have pretty good probabilities. Then, we looked at the corresponding diffusion patterns and chose $S11$ for the attack. The diffusion pattern for $S11$ is depicted in Fig. 3.

We want to point out that the larger r' is, the larger the differential probabilities for the r-round part are (since covering less rounds) inducing a smaller

[3] Notice that this does not mean that there is not any bias in the distribution of the output differences.

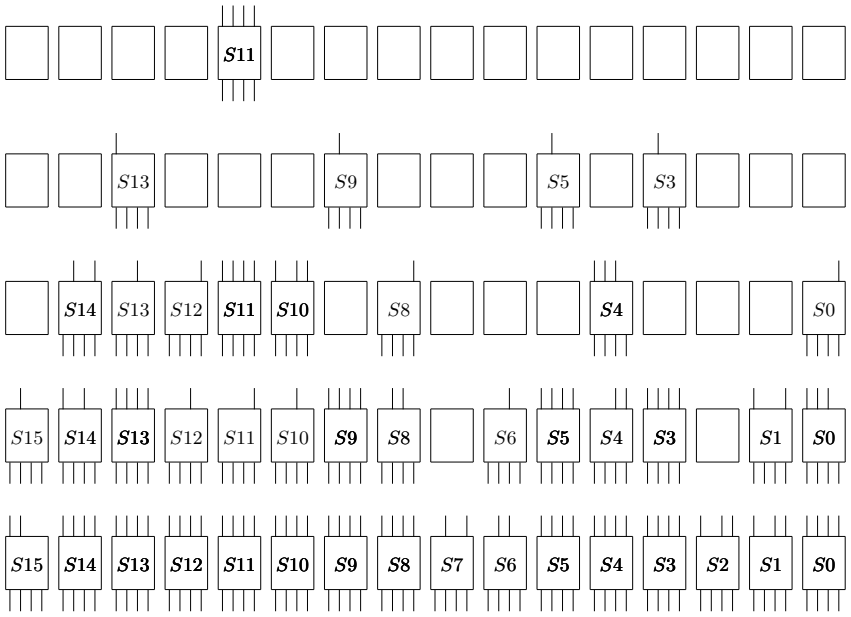

Fig. 3. Diffusion of differences activating only $S11$ over 5 rounds of PUFFIN and PUFFIN2.

data complexity. The counterpart of choosing a larger r' is that the number of key-bits involved increases. The partial decryption phase as presented in Algorithm 1 has a time complexity of $\Theta\left(N_s\, 2^{n_k}\right)$ where N_s is the number of pairs used and n_k the number of involved key bits. Hence, r' should be chosen such that $\log_2(N_s) + n_k$ is smaller than the master-key length.

It is actually possible to speed up Algorithm 1. Many techniques can be found in the literature but here we need to push them beyond their typical use since the number of rounds we invert is large (and so is the number of active S-boxes). Moreover, the analysis of the fastened Algorithm 1 becomes really tricky when using such advanced techniques. Therefore, we propose to look at key-recovery as a tree-traversal problem. Such representation will help in both the understanding of the problem and the derivation of an estimate of the complexity. This is precisely the two points we focus on in the next subsections.

3.3 Using a tree-based description for key-recovery

For a fixed pair of ciphertexts, the problem of incrementing the key-candidate counters can be seen as a tree traversal. Using this expression of the problem makes things clearer and might be helpful for the analysis of the attack complexity.

Counter incrementation as a tree traversal.
The attack uses a differential (δ_0, δ_r) over r rounds of the cipher. For each ciphertext pair (C, C') obtained from a plaintext pair having a difference δ_0, we have to increment by one each candidate counter corresponding to a value k such that $E_{r'}^{-1}(k, C) \oplus E_{r'}^{-1}(k, C') = \delta_r$. Not all the bits of $E_{r'}^{-1}$ are required to increment the counters. Only active S-boxes (as depicted in Fig. 3) need to be inverted. Let us denote by d the number of active S-boxes (maximal number of active S-boxes on the r' rounds for a pair of ciphertexts fulfilling $E_{r'}^{-1}(k, C) \oplus E_{r'}^{-1}(k, C') = \delta_r$.). The tree $\mathcal{T}_{C,C'}$ corresponding to the problem is a tree of height d and branching factor 2^4. This tree has 2^{4d} leaves corresponding to all the possible values for the key-bits XORed before inverting active S-boxes. At some depth in the tree, all the nodes are corresponding to the same active S-box. For each leave, there is a difference δ that results from the partial decryption of C and C' using the key-bits determined by the path of the leave. Note that not all the $4d$-bit values correspond to a master key. Indeed, these bits may be linked by the key-schedule algorithm. Incrementing counters boils down to detecting the leaves for which the $4d$ bits correspond to a real key (there are only 2^{n_k} such leaves) and for which the difference after deciphering is equal to δ_r.

We propose to implement the key-recovery part of the attack as the traversal of $\mathcal{T}_{C,C'}$. Then, to be efficient, we have to take care of not wasting time traversing *useless branches*. What we refer as a *useless branch* is a subtree that does contain no leave leading to a counter incrementation. Such branches may be the result of an inconsistence in the key-schedule (only 2^{n_k} out of 2^{4d} candidates correspond to a real key) or may simply come from the fact that not all the candidates are incremented by a given ciphertext pair.

Using a first filter.
An important remark here is that not all the 2^{n_k} leaves corresponding to valid candidates will produce difference δ_r after decryption. Let us introduce some formalism to express this. Let φ_{δ_r} be the function defined as follows where m is the message-length.

$$\begin{aligned} \varphi_{\delta_r} : \mathbb{F}_2^m \times \mathbb{F}_2^{n_k} &\to \mathbb{F}_2^m \times \mathbb{F}_2^m \\ (x, k) &\mapsto (y, y') = (E_{r'}(k, x), E_{r'}(k, x \oplus \delta_r)) \end{aligned}$$

The set $\mathrm{Im}(\varphi)$ is the set of ciphertext pairs that can be obtained after r' rounds of encryption when starting from a difference δ_r. The first main tool to speed up Algorithm 1 is to use a filter that is a subset $\mathcal{F} \subseteq \mathbb{F}_2^m \times \mathbb{F}_2^m$ such that $\mathrm{Im}(\varphi) \subseteq \mathcal{F}$. Indeed, no candidate counter will be incremented by a ciphertext pair not in $\mathrm{Im}(\varphi)$ thus processing such a pair is useless and time consuming. Such filter $\mathcal{F}$ is optimal when equal to $\mathrm{Im}(\varphi)$. Typically, filters consist of a set of reachable differences or potentially active key-bits obtained using techniques of truncated differential cryptanalysis or impossible differential cryptanalysis. The usefulness of algebraic techniques for this purpose is still an open question [10, 9].

Using round filters.
In the attack on PRESENT proposed in [7] and where r' is equal to 2, the author proposed to use an intermediate filtering step. For a given pair of ciphertexts, the attacker inverts the active S-boxes in the last round for all possible corresponding round-key bit values. He goes on inverting the penultimate round only if the difference obtained after the round inversion belongs to the set of possible differences. Such an additional filtering step has an important impact on the time complexity of the attack.

Using S-box filters.
The number of active S-boxes in the last round is too large in our context to use only round filters. Indeed, inverting the last round for all pairs that have not been discarded is far too time consuming. Hence, the natural idea is to apply a filter after each S-box inversion. This is precisely what we propose in this paper and we will see that it allows us to take large values for r' (up to 5). Using efficient filters drastically fastens the attack but is difficult to analyze. Indeed, ad hoc techniques used up to now becomes more tricky to apply for many reasons and particularly because dependencies between different filters may appear.

Invalid key values.
Considering S-boxes successively, we have to take care of detecting invalid values for the $4d$ key-bits early in the traversal. In Algorithm 1, each of the 2^{n_k} candidate was expanded to a $4d$-bit value then used for partial decryption. Using S-box filters, we try to avoid decrypting using all the candidates and hence we have to check for key-bit consistency during the traversal. This can be done by using r' states corresponding to round subkeys and by updating them relatively to the key-schedule each time key-bits are guessed. Then, some key-bits of previous rounds will be fixed in advance reducing the number of children of the next level of the tree.

Analyzing key-recovery complexity.
We presented techniques to speed up Algorithm 1 and we aim now at analyzing the resulting complexity of the attack. We tried to use the same kind of techniques that can be found in the literature but we faced many difficulties because of all the inherent dependencies between bits when considering more than 2 rounds of a cipher. A typical problem encountered is that the diffusion pattern provides a list of potentially active S-boxes that have to be inverted but, for a fixed ciphertext pair, some of these S-boxes may be inactive. We have to invert them to perform the r'-round decryption but we should not start with them since the corresponding filters will validate all the 4-bit values for the key. Hence, to be efficient, the order of S-boxes may depend on the ciphertext pair. It is not tractable to take this into account using aforementioned evaluation techniques and fixing the same order for all pairs would result in very high complexity. That

is the reason why we propose to analyze the complexity of this part of the attack in an ideal context using the tree-representation of the problem.

3.4 Expected complexity of the fast counter incrementation

In this subsection, we derive an asymptotic estimate for the key-recovery complexity under the strong assumption that we are able to detect all useless branches as soon as possible (that is detecting a useless branch when reaching its root) in constant time. This assumption is optimistic but we will discuss in Appendix A the fact that techniques used to speed up Algorithm 1 take constant time at the cost of table pre-computations and storage and that they allow a quasi-optimal tree traversal.

Now assuming that this hypothesis holds, then the complexity of one tree traversal is $\Theta(d\alpha)$ where α is the number of counters incremented by the processed pair. Let N be the number of available plaintext/ciphertext pairs, then we can form $N_s = N/2$ plaintext pairs and obtain the corresponding ciphertext pairs. We denote by $A_1, \ldots, A_{N_s}$ the number of incremented counters corresponding to the N_s ciphertext pairs. Then, we are interested in computing the complexity of the attack: $\sum_{i=1}^{N_s} d\,A_i$. Since we know very few about the distribution of A_i, we propose to estimate the expected value of the time complexity of the attack. Using the linearity of the expected value, we easily derive

$$\mathbb{E}\left(\sum_{i=1}^{N_s} d\,A_i\right) = N_s \cdot d \cdot \mathbb{E}(A).$$

Supposing that ciphertext pairs are uniformly distributed[4] over $\mathbb{F}_2^m \times \mathbb{F}_2^m$, then, the expected value of the cardinal of $\varphi_{\delta_r}^{-1}(y, y')$ is the ratio of the input by the output space cardinalities:

$$\mathbb{E}\left(\#\varphi_{\delta_r}^{-1}(y, y')\right) = \frac{2^{m+n_k}}{2^{2m}} = 2^{n_k - m}.$$

Hence, the expected value of the time complexity of the fastened version of Algorithm 1 using the tree-based approach is

$$\Theta(N_s\, d\, 2^{n_k - m}). \tag{1}$$

Notice that the N available plaintexts have to be read to form the pairs. Hence, it has to be taken into account when estimating the complexity of the attack. Moreover, the first filter can be applied at the same time as forming pairs using a hash table. The value provided by (1) may be smaller than N but this is due to the fact that after the first filter only very few pairs remain.

We presented a new technique to estimate the complexity of a simple differential cryptanalysis. In the next section, we present tools for analyzing the use of several differentials.

[4] This assumption is realistic since they are obtained by using $r + r'$ rounds of the cipher: if not true, it will induce an attack on $r + r' + 1$ rounds of the cipher.

4 Multiple differential cryptanalysis

In the previous section, we focused on techniques for incrementing candidate counters for a given ciphertext pair. All the statements were instantiated in the particular case of single differential cryptanalysis. Attacks presented here are multiple differential cryptanalyses that is attacks using more than one differential. The optimal way for combining information from many differentials is a work in progress and may hardly depend on the accuracy of differential probability estimates as it is the case for linear cryptanalysis [11–13]. Hence, we chose here a classical approach that consists in combining counters obtained from different differentials using addition. Such attacks have been formalized and studied in [8]. Before providing results on the differential probabilities of the best characteristics on PUFFIN and PUFFIN2, we briefly discuss the influence of the use of many differentials on the complexity given in (1) and recall the results given in the aforementioned paper.

4.1 Time cost of the use of many differentials

The asymptotic time complexity of the key recovery part given by formula (1) have been obtained for a single differential. In differential cryptanalysis, the attacker uses a set Δ of differentials. The set of input differences contained in Δ is denoted by Δ_0. To a given input difference $\delta_0^{(i)}$ in Δ_0 corresponds a set of differentials of the form $(\delta_0^{(i)}, \delta_r^{(i,j)})$. We denote by $\Delta_r^{(i)}$ the set of output differences corresponding to a given input difference $\delta_0^{(i)}$. Then, from well-chosen[5] N plaintext/ciphertext pairs, the attacker can form $N/2$ couples of plaintext for each input difference hence obtaining a total of $N_s = \frac{|\Delta_0| \cdot N}{2}$ samples.

Forming all the samples should have time complexity $\Theta(N_s)$ but it is possible to combine the formation of plaintext pairs to the first filter using a hash table (as mentioned in Section 3.4). It turns out that if the filter is efficient enough, the number of remaining pairs will smaller than N. In the other case, forming pairs will be more time consuming than $\Theta(N)$ but this complexity will be negligible compared to the one of the corresponding key-recovery. Hence, it is not abusive to consider than the time complexity of forming samples is the one of reading available plaintext/ciphertext couples that is $\Theta(N)$.

Then, for a given ciphertext pair, the input difference $\delta_0^{(i)}$ is fixed. This difference has a corresponding set of output differences $\Delta_r^{(i)}$. Since we combine differentials by summing counters, the only difference with Algorithm 1 is that a candidate counter will be incremented if the obtained difference belongs to the set $\Delta_r^{(i)}$. Hence, for a fixed input difference, the number of incremented counters will be multiplied by $|\Delta_r^{(i)}|$ influencing the complexity of a tree traversal. Summing over all possible samples, we obtain a global complexity of

[5] That is choosing plaintexts using the so-called *structures*.

$$\Theta\left(N_s\, d\, 2^{n_k-m}\, \frac{\sum_{i=1}^{|\Delta_0|} |\Delta_r^{(i)}|}{|\Delta_0|}\right). \tag{2}$$

Note that since $N_s = \frac{|\Delta_0|\cdot N}{2}$, the term $N_s\, \frac{\sum_{i=1}^{|\Delta_0|} |\Delta_r^{(i)}|}{|\Delta_0|}$ actually corresponds to $2N\,|\Delta|$.

4.2 Theoretical framework used

Let us first begin with the definition of some notation from the framework developed in [8].

Notation.
The attacker chooses a set Δ of differentials with probabilities $p^{(1)}, \ldots, p^{(|\Delta|)}$. The output differences of the differentials determine the set of active S-boxes involved in the partial deciphering process. The corresponding number of required key bits is denoted by n_k hence the attacker will have to distinguish the correct subkey among 2^{n_k}.

The cornerstone of the theoretical analysis in [8] is an estimate for the subkey-counter cumulative functions that is given in [8, Proposition 1]. This estimate is denoted by G and is parameterized by the number of samples. We will hence denote it by G_{N_s}. The function $G_{N_s}(\tau, p_*)$ is the estimate for the correct-key counter distribution and $G_{N_s}(\tau, p)$ is the one for the wrong keys. Values for p_* and p are equal to

$$p_* = \frac{\sum_i p_*^{(i)}}{|\Delta_0|} \quad \text{and} \quad p = \frac{|\Delta|}{2^{64}|\Delta_0|}.$$

We now recall the main two results that can be found in [8]. The first one is an estimate of the data complexity required for the correct key to be ranked among the ℓ most likely candidates with probability close to one half.

Corollary 1. *[8, Corollary1] Let ℓ be the size of the list of the remaining candidates and let n_k be the number of bits of the key we want to recover. Using the previous notations, the data complexity of a multiple differential cryptanalysis with success probability close to 0.5 can be estimated by*

$$N' = -2 \cdot \frac{\ln(2\sqrt{\pi}\ell\, 2^{-n_k})}{|\Delta_0| D(p_*||p)}, \tag{3}$$

where $D(p_*||p)$ denote the Kullback-Leibler divergence:

$$D(p_*||p) = p_* \log\left(\frac{p_*}{p}\right) + (1-p_*)\log\left(\frac{1-p_*}{1-p}\right)$$

In this result the statement "success probability close to 0.5" may seem unclear and imprecise[6]. The point is that the success probability corresponding to

[6] For a complete understanding, to please refer to [14]

the data complexity N' given in this Corollary may vary a bit around this value. In the case of differential or multiple differential cryptanalysis, our experiments show that using the value of N' given in Corollary 1 leads to a success probability between 0.35 and 0.50[7]. This formula provides an intuition on the impact of the use of many differentials on the data complexity through the denominator $|\Delta_0|D(p_*||p)$.

Then, to precisely adjust the number of samples to reach a given success probability, another result of [8] should be use. This one, presented in Corollary 2, is a tight formula for computing the success probability of an attack when only the ℓ most likely keys are tested and when N_s samples are available.

Corollary 2. *[8, Corollary2] Under the previous notations, the success probability, P_S, of a multiple differential cryptanalysis is given by*

$$P_S \approx 1 - G_{N_s}\left[G_{N_s}^{-1}\left(1 - \frac{\ell - 1}{2^{n_k} - 2}, p\right) - 1, p_*\right] \tag{4}$$

where the pseudo-inverse of G_{N_s} is defined by $G_{N_s}^{-1}(y) = \min\{x|G_{N_s}(x) \geq y\}$.

The tightness of formula (4) have been empirically tested on a reduced version of PRESENT in [8] with very convincing results.

4.3 The differential probabilities

In the first analysis of the security of PUFFIN [4] authors claim that since the best trail on 31 rounds (over the 32 rounds) has a probability equal to 2^{-62}, the cipher is secure against differential cryptanalysis. On the one hand and according to PUFFIN design -i.e. similar to the one of PRESENT- a differential will be composed of so many trails that the probability of a differential is dramatically underestimated when only considering the best differential trail. On the other hand, authors' analysis of the cipher security is based on the assumption that the attacker will only recover key bits from the last round (that is $r' = 1$) while attacks proposed in this paper actually use values for r' up to 5.

Using a Branch and Bound algorithm (see [15–17] for instance), we are able to compute the probabilities of the best differential trails. Combining these trails, we obtain lower bounds on the probabilities of the best differentials. As mentioned in Section 3, we have observed that the best differential trails have a single active S-box in the input and the output difference. Therefore we chose to select differentials such than only one S-box is active in the output difference. Diffusion properties of PUFFIN and PUFFIN2 suggest the use of $S11$ as active S-box among those having good differential properties.

We now provide some results about the differential probabilities for PUFFIN and PUFFIN2. The data complexity of the attack will depend on the number r

[7] Experiments from [14] shown a range from 0.52 to 0.65 in differential settings.

of rounds targeted by differentials, it will also depend on the differentials we will use. Moreover, the estimation of these probabilities is a critical point since a too pessimistic analysis will lead to the underestimation of the attack performances. In order to compare the different values for r' we looked for the best differentials over $r = 27, 28, 29, 30$ and 31 rounds. According to Section 3.2, we focused on r-round differentials (δ_0, δ_r) such that

$$\delta_r \in \Delta_{out} = \{\texttt{0x0000Y00000000000} | Y \in \{\texttt{0x1}, \cdots, \texttt{0xF}\}\}.$$

The best differentials - corresponding to our criteria - we found are given in Table 2. Notice that we did not consider the additional operations performed before the first round in PUFFIN and after the last round in PUFFIN2 since they do not alter the probabilities of differentials.

Table 2. Best differentials on r rounds with output difference activating only $S11$ and their probabilities.

r	δ_0	δ_r	p_*
27	0x0000000000000a000	0x0000400000000000	$2^{-49.71}$
28	0x000b400000000000	0x0000400000000000	$2^{-52.07}$
29	0x0000000000400000	0x0000400000000000	$2^{-53.59}$
30	0x0000400000000000	0x0000400000000000	$2^{-56.35}$
31	0x0000000000007000	0x0000400000000000	$2^{-57.9}$

5 Parameters and performances of proposed attacks

Let us now move to the choice of parameters for the attacks we propose. We want to precise that the attacks mentioned here are not claimed to be optimal since some of our choices relied on heuristics (for instance the choice of $S11$ may not lead to the best possible attack).

5.1 Formulas of the attack complexities

Simple formulas for complexities aim at easing the choice of parameters to balance them. More precisely, in the case of multiple differential cryptanalysis, there are three different steps to consider for the time complexity.

1. Obtaining plaintext/ciphertext pairs: $\Theta(N)$.
2. The key-recovery part with complexity given by formula (2).
3. The final exhaustive search that takes $\Theta(\ell\, 2^{n-n_k})$ where n is the master-key length and ℓ the maximum number of candidates to test.

Then, the time complexity of a multiple differential attack is

$$\Theta\left(N + \ell\, 2^{n-n_k} + N_s\, d\, 2^{n_k-m}\, \frac{\sum_{i=1}^{|\Delta_0|} |\Delta_r^{(i)}|}{|\Delta_0|}\right) \tag{5}$$

For a given set Δ of differentials, fixing r' will determine the values for d the number of active S-boxes and n_k the number of involved key-bits. Then, there are tight links between the data complexity N, the success probability P_S and the number of candidates to test ℓ. This relationship is represented by the formula (3). Since we aim at proposing attacks with success probabilities greater than 0.5, we propose to multiply this formula by a factor 1.5. Experiments in [14] show that for such a value, this probability gets around 0.8.

$$1.5 \cdot N' = -3 \cdot \frac{\ln(2\sqrt{\pi}\ell\, 2^{-n_k})}{|\Delta_0| D(p_*||p)}. \tag{6}$$

This is asymptotic in nature: the estimate is negative for values of ℓ close to 2^{n_k} but it tends toward the correct value of N as $\frac{\ell}{2^{n_k}}$ decreases.

This formula will be of great use since we can substitute it into the aforementioned complexities removing one parameter. Then, for a given set Δ and a fixed r', the problem boils down to balance these complexities using the parameter ℓ.

5.2 First approach: using a single differential

As both time and data complexity depend on the set of differentials, we chose, as a first approach, to study the complexities of differential cryptanalyses using a single differential. For the best differentials (δ_0, δ_r) described in Table 2, we propose to compute the complexities obtained for different values of r' ($r + r' = 32$ for PUFFIN and 34 for PUFFIN2). Once ℓ is fixed, the success probability will accurately be estimated using (4). As it is supposed to, it will vary in a reasonable range around 0.5.

Table 3. PUFFIN: Parameters of simple differential cryptanalyses using the formula for N given by (6).

r'	d	n_k	p_*	ℓ	N	Time C.	P_S
3	13	43	$2^{-53.59}$	$2^{00.1}$	$2^{57.49}$	$2^{85.10}$	0.75
4	27	81	$2^{-52.07}$	$2^{24.6}$	$2^{56.04}$	$2^{76.84}$	0.79
5	43	109	$2^{-49.71}$	$2^{78.6}$	$2^{52.45}$	$2^{101.95}$	0.85

Results in Table 3 and Table 4 confirm the fact that the formula (6) leads to a success probability close to 0.8. The parameters presented in Table 3 and Table 4 show that a differential attack using a single differential is enough to break both PUFFIN and PUFFIN2.

Table 4. PUFFIN2: Parameters of simple differential cryptanalyses using the formula for N given by (6).

r'	d	n_k	p_*	ℓ	N	Time C.	P_S
3	13	35	$2^{-57.90}$	$2^{12.4}$	$2^{61.25}$	$2^{61.35}$	0.75
4	27	59	$2^{-56.35}$	$2^{35.0}$	$2^{59.47}$	$2^{60.07}$	0.63
5	43	74	$2^{-53.59}$	$2^{61.1}$	$2^{55.60}$	$2^{70.21}$	0.87

One of the problems left as an open question for multiple differential cryptanalysis is the optimal choice of the parameters of the attack (typically the set of differences used). Results given here when using a single differential emphasize the fact that this notion of *optimal choice* may hardly depend on the context (Is the full code-book available?, Is the computational power the limiting factor?). Indeed, when moving from $r' = 4$ to $r' = 5$, samples are exchanged for computational effort. Nevertheless, there are parameters that are clearly sub-optimal as $r' = 3$ that, here, leads to an attack outperformed in both time and data complexities by other parameters.

5.3 Proposed attacks

Table 3 and Table 4 show that we can break both PUFFIN and PUFFIN2 using simple differential cryptanalysis. Nevertheless, complexities of these attacks can be improved using several differentials. The set of differential we propose for the attacks is big, that we present in Appendix in Table 10 only the parameters for the attacks with the smallest used differentials. We propose a multiple differential attack on PUFFIN where $r' = 4$ that outperforms the simple attack for $r' = 5$ i.e. that have smaller complexities (for both time and data). We also propose a multiple attack on PUFFIN2 with $r' = 4$ and attacks on both versions for $r' = 5$ to illustrate that the choice of differential sets and values for r' allow trading data complexity for time complexity.

The attacks proposed are summarized in Table 5 and detailed below.

Table 5. Attacks on the PUFFIN ciphers.

version	key bits	rounds	Data C.	Time C.	Success P.	
PUFFIN	128	32	2^{58}	2^{124}	> 0.25	[6]
PUFFIN	128	32	$2^{52.16}$	$2^{95.40}$	0.77	this paper
PUFFIN	128	32	$2^{49.42}$	$2^{108.84}$	0.59	this paper
PUFFIN2	80	34	$2^{55.58}$	$2^{64.66}$	0.58	this paper
PUFFIN2	80	34	$2^{52.30}$	$2^{74.78}$	0.78	this paper

Attacks on PUFFIN.
In the previous section, we presented a simple differential attack on PUFFIN with

data complexity $2^{52.45}$ and time complexity $2^{101.95}$ (case where $r' = 5$). Using a set of 830 differentials having $|\Delta_0| = 359$ input differences on $r = 28$ rounds (that is $r' = 4$), we obtain a multiple differential cryptanalysis ($p_* = 2^{-56.7178}$) which requires $2^{52.16}$ chosen plaintexts that has a time complexity of $2^{95.40}$ and success probability of 0.77 ($\ell = 2^{48.40}$). Complexities of this attack outperform the complexities of the simple differential cryptanalysis of PUFFIN we propose.

Nevertheless, if the bottleneck of the attacker resources is the data complexity and not the computational power, then we may use another set of parameters. For instance, there is a set of 954 differentials over 27 rounds ($r' = 5$) such that $|\Delta_0| = 318$ and that provides a probability $p_* = 2^{-54.6862}$. Taking ℓ equal to $2^{85.9}$, we obtain an attack that requires $2^{49.42}$ chosen plaintexts and that can be performed in time $2^{108.84}$ with a success probability of 0.59. If we compare this attack to the simple differential attack described in Table 3, the data complexity is divided by a factor $2^{3.03}$ while the time complexity is multiplied by $2^{6.89}$.

Examples of parameters we present show that depending on the attack requirements, different trade-offs for the complexities are possible.

Attacks on PUFFIN2.
For PUFFIN2, we also propose two kinds of multiple differential attacks, one that improves the data complexity with a small cost in time complexity (compared to the simple differential attack proposed in Table 4) and another that tries to minimize the data complexity of the attack.

We recall that in the case of $r' = 5$, the simple differential attack proposed in Table 4, can be performed using $2^{55.60}$ plaintexts in $2^{70.21}$ operations. Now, using a set of 115 differentials over 30 rounds (that is $r' = 4$) with $|\Delta_0| = 95$ input differences, we propose a multiple differential cryptanalysis on PUFFIN2 with time complexity $2^{64.66}$, data complexity $2^{55.58}$ and success probability 0.58 ($\ell = 2^{44.60}$ and $p_* = 2^{-59.1178}$). Differentials used for this attack are given in Table 10 for the interested reader to be able to check our results.

We can also perform a multiple differential attack on PUFFIN2 using differentials on 29 rounds ($r' = 5$). Using 210 differentials having $|\Delta_0| = 137$ different input differences, we obtain a probability $p_* = 2^{-57.7949}$ which lead to an attack with data complexity $2^{52.30}$, time complexity $2^{74.78}$ and success probability 0.78 ($\ell = 2^{66.5}$).

6 Conclusion

This work aims at illustrating the impact of the flaws that can be found in the security analysis of recent ciphers. We propose attacks on PUFFIN and PUFFIN2 that are low-cost ciphers designed to be more efficient than PRESENT (and particularly to be involutions). It turns out that they are also less secure since a simple differential cryptanalysis allows to break them. Using a new approach to estimate the cost of key-recovery in differential cryptanalyses, we proposed two attacks on these ciphers. The attack on PUFFIN recovers the full 128-bit key in $2^{95.40}$ operations with probability 0.77 when using $2^{52.16}$ chosen plaintexts and

the attack on PUFFIN2 recovers the full 80-bit key in $2^{65.66}$ operations with probability 0.58 when using $2^{55.58}$ chosen plaintexts.

While PUFFIN and PRESENT are similar, these differential attacks and the linear attack proposed by Leander[6] on PUFFIN and PUFFIN2 do not threaten the security of PRESENT. Differences can be explained by both the substitution and the permutation layers. Indeed, the fact that in PUFFIN, the S-box have properties that allow one-bit input and output differences implies there exists differential trails with one active S-box for each round what is not possible for PRESENT (at least two active S-boxes for one out of two rounds). This property can be removed using a linear transformation of the S-box.

Concerning the permutation layer, it turns out that the optimality of the PRESENT bit-permutation cannot be obtained with an involution. The permutation proposed in PUFFIN only reaches full diffusion after 5 rounds. This has implications in both the trail probabilities and the number of rounds that can be inverted by the attacker. Using involution permuation layer with diffusion on 4 rounds may improve the security of the cipher.

Impact of the use of "better" involutions as basic components for an SPN is an interesting scope for further research.

References

1. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.H.: PRESENT: An ultra-lightweight block cipher. In Paillier, P., Verbauwhede, I., eds.: Cryptographic Hardware and Embedded Systems - CHES 2007. Volume 4727 of LNCS., Springer (2007) 450–466
2. Matsui, M.: Linear cryptanalysis method for DES cipher. In Helleseth, T., ed.: Advances in Cryptology - EUROCRYPT 1993. Volume 765 of LNCS., Springer (1994) 386–397
3. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology **4** (1991) 3–72
4. Cheng, H., Heys, H.M., Wang, C.: PUFFIN: A novel compact block cipher targeted to embedded. In Fanucci, L., ed.: Conference on Digital System Design: ARchitectures, Methods and Tools - DSD 2008, IEEE (2008) 383–390
5. Wang, C., Heys, H.M.: An ultra compact block cipher for serialized architecture implementations. In: Canadian Conference on Electrical and Computer Engineering - CCECE 2009, IEEE (2009) 1085–1090
6. Leander, G.: On linear hulls, statistical saturation attacks, PRESENT and a cryptanalysis of PUFFIN. In Paterson, K., ed.: Advances in Cryptology - EUROCRYPT 2011. Volume 6632 of LNCS., Springer (2011) 303–322
7. Wang, M.: Differential cryptanalysis of reduced-round PRESENT. In Vaudenay, S., ed.: Progress in Cryptology - AFRICACRYPT 2008. Volume 5023 of LNCS., Springer (2008) 40–49
8. Blondeau, C., Gérard, B.: Multiple differential cryptanalysis: Theory and practice. In Joux, A., ed.: Fast Software Encryption - FSE 2011. Volume 6733 of LNCS., Springer (2011) 35–54
9. Wang, M., Sun, Y., Mouha, N., Preneel, B.: Algebraic techniques in differential cryptanalysis revisited. In Parampalli, U., Hawkes, P., eds.: Information Security and Privacy - ACISP 2011. Volume 6812 of LNCS., Springer (2011) 120–141

10. Albrecht, M., Cid, C.: Algebraic techniques in differential cryptanalysis. In Dunkelman, O., ed.: Fast Software Encryption - FSE 2009. Volume 5665 of LNCS., Springer (2009) 193–208
11. Hermelin, M., Cho, J.Y., Nyberg, K.: Multidimensional extension of Matsui's algorithm 2. In Dunkelman, O., ed.: Fast Software Encryption - FSE 2009. Volume 5665 of LNCS., Springer (2009) 209–227
12. Hermelin, M., Cho, J.Y., Nyberg, K.: Statistical tests for key recovery using multidimensional extension of Matsui's algorithm 1. Advances in Cryptology - EUROCRYPT 2009 POSTERSESSION (2009)
13. Cho, J.Y.: Linear cryptanalysis of reduced-round PRESENT. In Pieprzyk, J., ed.: Topics in Cryptology - CT-RSA 2010. Volume 5985 of LNCS., Springer (2010) 302–317
14. Blondeau, C., Gérard, B., Tillich, J.P.: Accurate estimates of the data complexity and success probability for various cryptanalyses. Design, Codes and Cryptography **59** (2011) 3–34
15. Biryukov, A., De Cannière, C., Quisquater, M.: On multiple linear approximations. In Desmedt, Y., ed.: Advances in Cryptology - CRYPTO 2004. Volume 3152 of LNCS., Springer (2004) 1–22
16. Collard, B., Standaert, F.X., Quisquater, J.J.: Improved and multiple linear cryptanalysis of reduced round Serpent. In: Inscrypt'07. Volume 4990 of LNCS., Springer–Verlag (2007) 51–65
17. Blondeau, C., Gérard, B.: Links between theoretical and effective differential probabilities: Experiments on PRESENT. In: TOOLS'10. (2010) http://eprint.iacr.org/2010/261.

A More details on the tree-traversal

In Section 3.3, we modeled the key-recovery part of a differential attack as a tree traversal. Nodes at the same depth in the tree correspond to one of the active S-boxes in the diffusion path. There are 2^4 children corresponding to the possible values for the 4 key bits XORed to the output value of the S-box corresponding to the node. When traversing the tree, passing along an edge refers to XORing the corresponding 4-bit key value and inverting the S-box. When reaching a node, it is essential to detect the current branch (this node and its descendants) as useless if so. A useless branch is a branch where all leaves at depth d do not correspond to counter incrementation. Such branches may appear for different reasons. If the current node is inconsistent with the previous guessed key bits for instance (this case is really easy to detect). The other source of useless branches comes from the fact that the number of incremented counter is far smaller than the 2^{n_k} candidates. Such branches are more difficult to detect even using many filters.

When analyzing this tree traversal in Section 3.4 we made the strong hypothesis that we were able to detect useless branches at the top level (that is as soon as possible). This is a bit optimistic according to what has just been said but using the tricks we are to detail, the proportion of non-discarded useless branches can be kept small. A more precise study of the complexity taking this proportion into account may be of interest here and is left as an open question for further work.

We detail here the algorithm we have in mind for constructing and traversing the tree efficiently (that is detecting most of the useless branches in constant time).

Constructing the tree.
Obviously we are not going to construct the tree since we aim at traverse the tree avoiding useless branches. Nevertheless, there is one degree of freedom in the definition we gave for the tree $\mathcal{T}_{C,C'}$: the order of S-box inversions. Modifying the order of S-boxes will not necessarily help in improving useless branch detection technique but using an *efficient* S-box order, useless branches may be met earlier in the tree traversal. This would reduce the constant hidden in the Θ notation and hence may be carefully looked at when implementing a practical attack but can be omitted for analyzing an attack.

Detecting key-bit inconsistency.
As already mentioned, this can be done by updating a $4d$ state following the key-schedule algorithm. Then, when reaching a new node, the set of child nodes considered is restricted if some key bits have already been guessed. In this context changing S-box order may also be of interest. Indeed, for PUFFIN2, the only non-linear part of the key-schedule is the application of the substitution layer to 64 bits of the current key state. Hence, some key bits directly correspond from one subkey to the other. Starting by guessing those bits may induce more efficient filters at the top of the tree.

Round filtering.
The round filtering process as detail in [7] consist in detected some useless branch after deciphering all active S-boxes in one round. This one may not be optimal due to memory limitation. Hence, they may be reduced to a set of reachable differentials. Applying such a filter can be done efficiently since it will consist in at most 2^{63} differences which can be stored in a relevant structure with search cost logarithmic in its size. Such sieves will be applied only at some depth of the tree hence it is not abusive to consider that this cost is constant in nature. Notice that the first sieve as to be applied to all the N_s samples hence the total cost of the key-recovery step may not be smaller than N_s.

S-box filtering.
An other kind of filters are S-box filters. Instead of applying such filters after having inverted the S-box using all the possible values of key bits, it is more efficient to pre-compute a $2^4 \times (2^4 - 1)$ table containing, for each pair of outputs, the list of keys that lead to a correct input difference. The memory cost of this technique for b-bit S-boxes is $\Theta(d\, 2^{3b})$ hence is negligible compared to the memory used for counters (2^{n_k}) when inverting a large number of S-boxes. Again, the cost of such filtering is constant regarding parameters of the attack.

B Components of PUFFIN

This appendix section contains the details for PUFFIN and PUFFIN2 specification.

B.1 Common components

In Table 6 the S-box used in both PUFFIN and PUFFIN2 is given using hexadecimal values.

Table 6. PUFFIN/PUFFIN2 S-box in hexadecimal, `S1(0x0)` = `0xD`.

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	D	7	3	2	9	A	C	1	F	4	5	E	6	0	B	8

Then, Table 7 specifies the bit-permutation P. The table has to be read as: the input bit at position $a \cdot 8 + b$ will be sent to the position given in the cell at line a and column b in the output. For instance, the 19-th input bit will correspond to the 21-th output bit. As shown in Fig. 3, we number S-boxes (hence bits) from the right to the left (little endian representation).

Table 7. 64-bit permutation P: $input = row \cdot 8 + column$.

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

B.2 Components used in PUFFIN key-schedule

As for Table 7, Table 9 and Table 8 have to be read in the following way. The value in the row a and column b is the output position of the input bit number $16a + b$ or $8a + b$ depending on the table.

Table 8. The 64-bit selection used in the PUFFIN key-schedule: $input = row \cdot 8 + column$.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	2	122	14	57	88	35	97	51
1	56	62	99	69	45	70	93	50
2	82	13	3	21	31	113	83	100
3	11	22	30	64	40	95	119	49
4	44	53	111	121	28	80	29	120
5	96	54	25	63	23	116	18	8
6	110	17	43	85	15	94	41	71
7	1	90	117	123	37	47	42	38

Table 9. The 128-bit permutation used in the PUFFIN key-schedule: $input = row \cdot 16 + column$.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	21	120	125	109	78	80	115	54
1	112	20	28	19	55	75	40	111
2	44	108	94	86	93	43	67	7
3	114	68	5	74	82	4	53	69
4	22	60	105	102	84	123	110	51
5	118	31	99	16	14	33	127	90
6	57	98	119	66	30	97	52	70
7	91	24	37	92	64	1	36	27
8	23	81	87	13	95	117	0	8
9	124	26	126	17	3	9	101	6
10	34	104	47	62	29	76	71	49
11	107	72	11	18	106	10	25	83
12	46	96	116	48	45	32	15	41
13	38	56	113	61	122	100	79	12
14	50	121	63	88	42	59	39	2
15	85	89	58	73	77	103	35	65

B.3 Differentials used to attack PUFFIN2

The following table contains differentials we use in the attack on PUFFIN2 with r' equal to 4. Since the only active box in the outpu difference is the 11-th one, we only mention the corresponding 4-bit output difference in the column $\delta_r|_{S11}$.

Table 10. Differentials used for attacking PUFFIN2 with $r = 30$ and $r' = 4$.

δ_0	$\delta_r\|_{S11}$	p_*	δ_0	$\delta_r\|_{S11}$	p_*
0x0000000000040003	0x4	$2^{-55.55}$	0x0000c00000000000	0x4	$2^{-60.11}$
0x0000000000040003	0x8	$2^{-58.28}$	0x000000000000000d	0x4	$2^{-60.14}$
0x0000000000040003	0x6	$2^{-58.39}$	0x0000000000090001	0x4	$2^{-60.18}$
0x0000000000040003	0x2	$2^{-58.52}$	0x0000000000040001	0x4	$2^{-60.28}$
0x0000000000040003	0xa	$2^{-59.69}$	0x00009a0000000000	0x4	$2^{-60.29}$
0x0000000000040000	0x4	$2^{-55.72}$	0x000100000000000d	0x4	$2^{-60.29}$
0x0000000000040000	0x8	$2^{-58.25}$	0x0001400000000000	0x4	$2^{-60.33}$
0x0000000000040000	0x2	$2^{-58.42}$	0x000000000000d0001	0x4	$2^{-60.42}$
0x0000000000040000	0x6	$2^{-58.55}$	0x0000400000040001	0x4	$2^{-60.44}$
0x0000000000040000	0xa	$2^{-59.65}$	0x000a000000000006	0x4	$2^{-60.45}$
0x0003600000000000	0x4	$2^{-56.79}$	0x00a0000000000000	0x4	$2^{-60.56}$
0x0003600000000000	0x8	$2^{-59.56}$	0x0001d00000000000	0x4	$2^{-60.64}$
0x0003600000000000	0x2	$2^{-59.72}$	0x0000100000050000	0x4	$2^{-60.72}$
0x0003600000000000	0x6	$2^{-59.83}$	0x0000c000000a0000	0x4	$2^{-60.74}$
0x0000000000000001	0x4	$2^{-57.96}$	0x0000000000003000	0x4	$2^{-60.74}$
0x0000000000000001	0x8	$2^{-60.77}$	0x0000500000000000	0x4	$2^{-60.80}$
0x0000000000000001	0x2	$2^{-60.95}$	0x0003400000040000	0x4	$2^{-60.82}$
0x0000000000000001	0x6	$2^{-60.97}$	0x000100000000000a	0x4	$2^{-60.83}$
0x000d400000000000	0x4	$2^{-58.11}$	0x000d000000000000	0x4	$2^{-60.83}$
0x0003000000000006	0x4	$2^{-58.40}$	0x00b0000000000000	0x4	$2^{-60.87}$
0x000300000000000a	0x4	$2^{-58.44}$	0x0000000000004000	0x4	$2^{-60.94}$
0x000b000000000000	0x4	$2^{-58.46}$	0x0000000000040006	0x4	$2^{-60.96}$
0x000a000000000001	0x4	$2^{-58.52}$	0x00001e0000000000	0x4	$2^{-60.99}$
0x0000000000000008	0x4	$2^{-58.63}$	0x0001300000000000	0x4	$2^{-61.08}$
0x000a400000000000	0x4	$2^{-58.74}$	0x0000800000000000	0x4	$2^{-61.09}$
0x0000000000030000	0x4	$2^{-58.78}$	0x0003000000000001	0x4	$2^{-61.11}$
0x0006000000000001	0x4	$2^{-58.88}$	0x0000000000030001	0x4	$2^{-61.12}$
0x0000100000000000	0x4	$2^{-58.92}$	0x0050000000000000	0x4	$2^{-61.13}$
0x0000100000000000	0x8	$2^{-61.62}$	0x00a00000000a0000	0x4	$2^{-61.13}$
0x0000100000000000	0x2	$2^{-61.75}$	0x0000000000060000	0x4	$2^{-61.14}$
0x0000100000000000	0x6	$2^{-61.95}$	0x0080900000000000	0x4	$2^{-61.16}$
0x000a000000000003	0x4	$2^{-58.99}$	0x0001000000040000	0x4	$2^{-61.24}$
0x0001000000000003	0x4	$2^{-59.18}$	0x0050100000000000	0x4	$2^{-61.25}$
0x000100000000000b	0x4	$2^{-59.22}$	0x0000000000000003	0x4	$2^{-61.26}$
0x00000000000e0000	0x4	$2^{-59.33}$	0x00a0300000000000	0x4	$2^{-61.36}$
0x0000200000000000	0x4	$2^{-59.38}$	0x00a0800000000000	0x4	$2^{-61.39}$
0x0000200000000000	0x8	$2^{-61.79}$	0x0000890000000000	0x4	$2^{-61.40}$
0x0000200000000000	0x2	$2^{-61.90}$	0x0000000000000006	0x4	$2^{-61.40}$
0x0000200000000000	0x6	$2^{-62.26}$	0x00d0000000000000	0x4	$2^{-61.44}$
0x0000700000000000	0x4	$2^{-59.54}$	0x0080800000000000	0x4	$2^{-61.46}$
0x0000000000000004	0x4	$2^{-59.61}$	0x0000400000000003	0x4	$2^{-61.46}$
0x00006a0000000000	0x4	$2^{-59.62}$	0x0000400000040003	0x4	$2^{-61.50}$
0x000d000000000001	0x4	$2^{-59.77}$	0x000d000000040000	0x4	$2^{-61.51}$
0x0000000000090000	0x4	$2^{-59.85}$	0x000b000000000001	0x4	$2^{-61.60}$
0x0006400000000000	0x4	$2^{-59.95}$	0x0000000900000000	0x4	$2^{-61.63}$

δ_0	$\delta_r\|_{S11}$	p_*	δ_0	$\delta_r\|_{S11}$	p_*
0x0000030000000000	0x4	$2^{-61.64}$	0x0001000000030000	0x4	$2^{-61.89}$
0x0000d00000080000	0x4	$2^{-61.65}$	0x0000a000000a0000	0x4	$2^{-61.92}$
0x00004000000a0000	0x4	$2^{-61.67}$	0x000600000000000a	0x4	$2^{-61.94}$
0x0001000000000001	0x4	$2^{-61.70}$	0x0008000000000006	0x4	$2^{-61.99}$
0x0000ba0000000000	0x4	$2^{-61.71}$	0x0001400000030000	0x4	$2^{-62.00}$
0x0001300000040000	0x4	$2^{-61.72}$	0x000b000000040001	0x4	$2^{-62.01}$
0x0300000000090000	0x4	$2^{-61.73}$	0x00a0900000000000	0x4	$2^{-62.01}$
0x0050200000000000	0x4	$2^{-61.78}$	0x0005000000000004	0x4	$2^{-62.01}$
0x0000680000000000	0x4	$2^{-61.80}$	0x0000c80000000000	0x4	$2^{-62.05}$
0x0003100000000000	0x4	$2^{-61.83}$	0x00008a0000000000	0x4	$2^{-62.05}$
0x0000600000050000	0x4	$2^{-61.84}$	0x0000000000003001	0x4	$2^{-62.05}$
0x0000000600000000	0x4	$2^{-61.85}$	0x0000600000040003	0x4	$2^{-62.08}$
0x0000400000060000	0x4	$2^{-61.87}$			