Differential Cryptanalysis of Round-Reduced PRINTcipher: Computing Roots of Permutations

Mohamed Ahmed Abdelraheem, Gregor Leander, Erik Zenner

Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark
{M.A.Abdelraheem,G.Leander,E.Zenner}@mat.dtu.dk

Abstract At CHES 2010, the new block cipher PRINTcipher was presented. In addition to using an xor round key as is common practice for round-based block ciphers, PRINTcipher also uses key-dependent permutations. While this seems to make differential cryptanalysis difficult due to the unknown bit permutations, we show in this paper that this is not the case. We present two differential attacks that successfully break about half of the rounds of PRINTcipher, thereby giving the first cryptanalytic result on the cipher.

In addition, one of the attacks is of independent interest, since it uses a mechanism to compute roots of permutations. If an attacker knows the many-round permutation π^r, the algorithm can be used to compute the underlying single-round permutation π. This technique is thus relevant for all iterative ciphers that deploy key-dependent permutations. In the case of PRINTcipher, it can be used to show that the linear layer adds little to the security against differential attacks.

Keywords. symmetric cryptography, block cipher, differential cryptanalysis, permutations

1 Introduction

After the establishment of Rijndael as AES, the need for new block ciphers has greatly diminished. However, given that the future IT-landscape is supposed to be dominated by tiny computing devices such as RFID tags or sensor networks, the need for low cost security has grown substantially. This need opened up the research field of light-weight cryptography. Quite a number of light-weight block ciphers have been proposed in the last couple of years, examples among others are PRESENT [3], HIGHT [7] and KATAN/KTANTAN [4].

PRINTcipher One recent proposal in this direction is the block cipher PRINTcipher presented at CHES 2010. PRINTcipher is an SP-network and comes in two versions, PRINTcipher-48 and PRINTcipher-96 with block sizes of 48 and 96 bits. PRINTcipher is targeted at IC-printing and makes use of the fact that this technology allows to make the circuit implementing the cipher key-dependent. This allows PRINTcipher to be implemented with a considerably smaller circuit compared to other light-weight ciphers. In order to maximize the profit from a key-dependent circuit all round keys in PRINTcipher are identical. To increase the size of the key space beyond the block size, the key in PRINTcipher consists not only of a (constant) round key xored to the state, but also parts of the linear layer are made key-dependent.

Differential Cryptanalysis This attack, invented by Biham and Shamir [2], is one of the most powerful and most general attacks on block ciphers known. The main idea is to encrypt pairs of plaintexts and trace the evolution of their difference through the encryption process. As most modern block ciphers are round based, an attacker usually starts by analyzing one round of the cipher with respect to difference propagation and extends to multiple rounds afterwards. Under well established independence assumptions the probability that a plaintext pair with a given difference α leads to a ciphertext pair with difference β can easily be computed by studying single rounds. Thus, differential attacks are most often based on so-called differential characteristics, that is a sequence of intermediate differences for all rounds together with their associated probabilities.

Our Results In this paper we mount a differential attack on round-reduced versions of PRINTcipher. The main technical problem while doing so is that the differential characteristics are key-dependent, more precisely they depend on the (key-dependent) choice of the linear layer. That is to say, without knowing the key, we do not know the best differential characteristics. At first glance, this seems to complicate a differential attack on PRINTcipher. There is another way to look at this, though. If the differential characteristics are key-dependent, then conversely, knowing the best differential one might be able to deduce information about the key. In general this dependency might be very complex. However, in the case of PRINTcipher we show that given the best differentials, computing the key-dependent linear layer can be reduced to computing roots of permutations in S_{48} or S_{96}. The remaining key bits, that is the constant round key xored to the state, can then be recovered using a standard differential attack, or, at a higher cost, simply by brute force.

Now, computing roots of permutations is a well-studied problem and our attack will profit from known algorithms. However, note that a permutation can have a huge number of roots and this causes two problems. First, this makes algorithms computing all possible roots eventually slow and second, in the case of PRINTcipher this means that many possible linear layers are proposed. We explain how both problems can be overcome.

In particular, our results show that making the linear layer of PRINTcipher key-dependent adds little to no additional security against differential attacks.

Related Work PRINTcipher is not the first block cipher with key-dependent components. Other well known examples are Khufu [10], the Khufu variation Blowfish [12] and Twofish [13]. Along with those proposals, several attempts to cryptanalyze those block ciphers, see for example [5] for a differential attack on Khufu or Vaudenay's attack on round-reduced Blowfish [14] have been published.

2 A Short Description of PRINTcipher

This section holds a short description of PRINTcipher, focusing only on the parts that are of interest for our analysis. For more details we refer to [8]. PRINTcipher-48 (resp -96) is an SP-network with a block size of $b = 48$ (resp $b = 96$) bits and 48 (resp 96) rounds. The key size is 80 bits for PRINTcipher-48 and 160 bits for PRINTcipher-96. It is closely related to the block cipher PRESENT in the sense that both ciphers use small s-boxes and a simple bit permutation as the linear layer. PRINTcipher uses a single 3 bit s-box shown in the following table.

x	0	1	2	3	4	5	6	7
$S[x]$	0	1	3	6	7	4	5	2

In the non-linear layer the current state is split into 16 words of 3 bits for PRINTcipher-48 and into 32 words of 3 bits for PRINTcipher-96 and each word is processed by the s-box in parallel. The linear layer consists of a bit permutation, where bit i of the current state is moved to bit position $P(i)$ where

$$P(i) = \begin{cases} 3i - 2 \bmod b - 1 & \text{for } 1 \leq i \leq b-1, \\ b & \text{for } i = b, \end{cases}$$

where $b \in \{48, 96\}$ is the block size.

The peculiar part of PRINTcipher is to have all rounds identical up to adding a round constant on a small number of bits. Here identical has to be understood as including the round key, in other words, all round keys are identical. As a simple round key xored to the state in each round limits the key size to 48 resp 96 bits, an additional key-dependent permutation layer was introduced. This permutation layer permutes the input bits of each s-box individually. Out of 6 possible permutations on 3 bits, only four are valid permutations for PRINTcipher.

For PRINTcipher-48 the 80-bit user-supplied key k is split into two subkeys $k = \mathrm{sk}_1||\mathrm{sk}_2$ where sk_1 is 48 bits long and sk_2 is 32 bits long. The first subkey sk_1 is xored to the state at the beginning of each round. The second subkey sk_2 is used to generate the key-dependent permutations in the following way. The 32-bits are divided into 16 sets of two bits and each two-bit quantity $a_1||a_0$ is used to pick one of four of the six available permutations of the three input bits. Specifically, the three input bits $c_2||c_1||c_0$ are permuted to give the following output bits according to two key bits $a_0||a_1$.

$a_1\|\|a_0$	
00	$c_2\|\|c_1\|\|c_0$
01	$c_1\|\|c_2\|\|c_0$
10	$c_2\|\|c_0\|\|c_1$
11	$c_0\|\|c_1\|\|c_2$

One round of PRINTcipher-48 is shown in Figure 1.

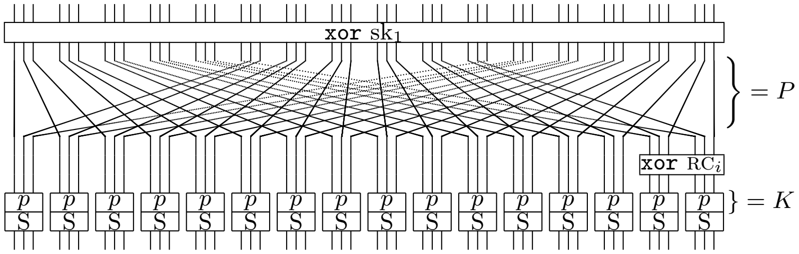


Figure1. One round of PRINTcipher-48 illustrating the bit-mapping between the 16 3-bit S-boxes from one round to the next. The first subkey is used in the first `xor`, the round counter is denoted RC$_i$, while key-dependent permutations are used at the input to each S-box.

3 Using Differential Cryptanalysis To Recover the Permutation Key

A classical differential attack against an SP-network finds an input difference α that produces a certain output difference β with high probability (a so-called *differential*). The attacker then analyses a large number of input pairs (x, x') with $x \oplus x' = \alpha$ and their corresponding output pairs (y, y'), hoping to find the expected difference $\beta = y \oplus y'$. Once this difference actually occurs, the attacker learns something about the internal behaviour of cipher. In particular, he can often use this knowledge to recover parts of the key.

For PRINTcipher, this attack can not be directly applied in a straightforward fashion, since finding good differentials requires the knowledge of the linear layer, which for PRINTcipher is key-dependent and thus unknown. As already pointed out, however, this disadvantage can also be turned into an advantage for the attacker: It can be used to learn something about the part of the key that defines the linear layer.

3.1 Optimal differential characteristic

We start our analysis by proving the following fact about the optimal PRINTcipher characteristic.

Theorem 1. *Given an input difference α of weight one, the unique most probable r-round differential characteristic is*

$$\alpha \to (PK)(\alpha) \to (PK)^2(\alpha) \to (PK)^r(\alpha),$$

which will occur with probability $(1/4)^r$.

		Δy							
		000	001	010	011	100	101	110	111
Δx	000	8	-	-	-	-	-	-	-
	001	-	$\boxed{2}$	-	2	-	2	-	2
	010	-	-	$\boxed{2}$	2	-	-	2	2
	011	-	2	2	-	-	2	2	-
	100	-	-	-	-	$\boxed{2}$	2	2	2
	101	-	2	-	2	2	-	2	-
	110	-	-	2	2	2	2	-	-
	111	-	2	2	-	2	-	-	2

Table1. Difference distribution table for PRINTcipher S-box. Note that the difference table is symmetric. 1-bit to 1-bit differences are marked with boxes.

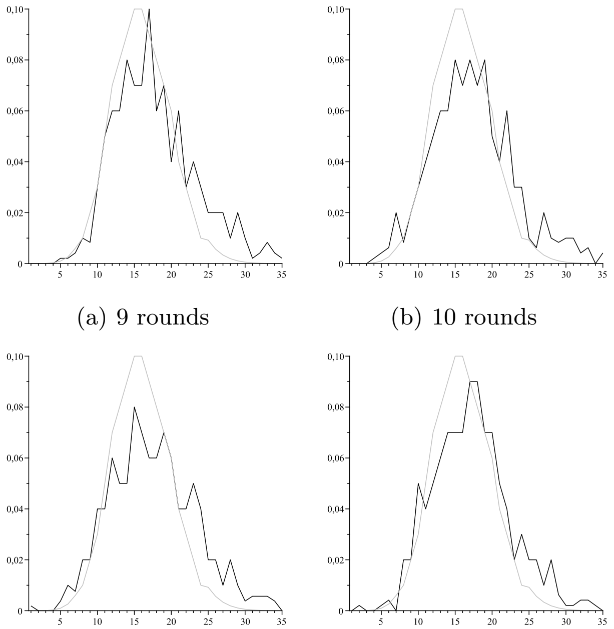
Proof. The difference distribution table for the PRINTcipher S-box (see Table 1) shows that all occuring differences are equally probable (prob. 1/4) and that for every 1-bit input difference, there exists exactly one 1-bit output difference. From this, it follows that starting with a 1-bit input difference, a 1-bit differential trail through r rounds of PRINTcipher occurs with probability $(1/4)^r$. Note also that this trail has the minimum possible number of r active S-boxes and that no other S-box difference is more probable, meaning that this trail is the most probable one.

Also note that the 1-bit output difference always occurs in the same bit position as the 1-bit input difference. This means that if the 1-bit differential occurs, the S-box does not permute the active bit - its position on the differential trail is only influenced by the fixed permutation P and the key-dependent permutation K. Thus, the difference α is indeed mapped to $(PK)^r(\alpha)$, which proves the theorem. □

The probability of the differential characteristic is based on assumptions, in particular the assumption of independent round-keys. This assumption is in particulary questionable for PRINTcipher as all round-keys are identical. Therefore, we ran (limited) tests to see if the theoretical probability of $(1/4)^r$ is actually met. Our experimental data depicted in Figure 2 suggest that indeed the probability is slightly higher than expected.

3.2 Targeting the xor key

In the following, we assume that the attacker has the full code book at his disposal (i.e. 2^{48} plaintext/ciphertext pairs for r rounds of PRINTcipher-48). For every 1-bit input difference $\alpha_1 = (100...0), \alpha_2 = (010...0), \ldots, \alpha_{48}$
$= (000...1)$, the attacker now forms all 2^{47} input pairs with $x \oplus x' = \alpha_i$ and checks whether the output difference also has weight one. If yes, he assumes



(a) 9 rounds

(b) 10 rounds

(c) 11 rounds

(d) 12 rounds

Figure2. Experimental vs. theoretical estimates for the optimal differentials. The x-axis shows the number of pairs yielding the correct output difference within 2^{2r+4} tries. The y-axis shows the relative frequency.

that he has found the above optimal characteristic. It turns out that as long as $r \leq 22$, this is very likely to happen[1].

Every successful 1-bit differential gives the attacker information about the internal behaviour of the cipher which can be used to reconstruct part of the xor key. Consider the first round of the cipher and note that according to [8], the order of S-box and key-dependent permutation can be inversed by adding two constants c and d that do not affect the differential. Thus, we can alternatively consider one PRINTcipher round to consist of key addition, fixed permutation, round constant, adding c, S-box, key-dependent permutation, and adding d. In particular, for the purposes of differential cryptanalysis, we can assume the S-box to follow directly after the key addition.

Now consider a successful differential with input difference $\alpha_1 = (100...0)$. Three key bits (with indices 1, 17 and 33) will affect the bits that go into the first S-box. There are *a priori* 8 possible choices for these bits, generating all

[1] We have 2^{47} pairs and a success probability of $(1/4)^{22} = 2^{-44}$, yielding a success probability close to 1 for any single index i and of ≈ 0.984 for all 48 indices. When increasing the number of rounds to $r = 23$, the success probability drops to 0.865 for any single index and to 0.001 for all 48 indices.

possible 3-bit S-box input pairs with difference α_1. However, as shown in Table 1, only 2 of them will lead to a 1-bit output difference after running through the S-box. Thus, only 1/4 of all keys meet the condition for the first S-box, reducing the key entropy by 2 bit. Thus, finding 16 successful 1-bit to 1-bit differentials (one for each S-box) will reduce the key entropy by 32 bit, leaving a brute-force effort of 2^{48} steps. This work factor could be reduced further, but without greatly affecting the overall running time, which is dominated by the 2^{48} steps of computing the full code book anyway.

The false positive problem: The above description is a simplification since it does not take false positives into account. For every 1-bit differential, trying out 2^{47} plaintext pairs will yield $2^{47} \cdot \frac{48}{2^{48}} = 24$ false positives on average, i.e. 1-bit output differences that occur accidentially and not as a result of the correct differential. The question remains how they can be distinguished from the cases where the 1-bit output differences really result from the desired differential. It turns out that for 22 rounds, the probability that all 48 differentials are met at least three times is 0.514, meaning that in more than half of the cases, the correct 1-bit difference should be recognizable by occuring more often than the false positives, which very rarely occur more than twice.

3.3 Targeting the linear layer

As it turns out, there is also a different way of using the above differential to cryptanalyse PRINTcipher. Remembering that according to Theorem 1, *every* 1-bit to 1-bit characteristic is optimal and describes the mapping $\alpha \rightarrow (PK)^r(\alpha)$, the following corollary immediately follows:

Corollary 1. *Learning all optimal characteristics is the same as learning* $(PK)^r$.

If the attacker has the full code book available, he can form 2^{47} plaintext pairs for every 1-bit input difference. The probability that at least one example of all 48 1-bit differentials is found is 0.984, and as stated above, the probability that they all can be distinguished successfully from false positives is 0.514. Thus, for up to $r = 22$ rounds of PRINTcipher-48, the attacker can learn the permutation $(PK)^r$.

If he can find the r-th root of this permutation, then he has derived PK and thus the linear layer key K. Once this has been done, the xor key can be retrieved bitwise, using a simple divide-and-conquer attack similar to the one described in Subsection 3.2. It turns out that here too, the overall running time is dominated by computing the code book, i.e. the attack requires about 2^{48} computational steps.

This type of differential attack is the dual to the one targeting the xor key and is relevant for all SPN-like ciphers that use key-dependent permutations. For this reason, it is not only interesting for the analysis of PRINTcipher, but also for the understanding of key-dependent permutations in general. In the rest of this paper, we will thus discuss the computation of permutation roots in more detail.

4 Finding (PRINTcipher)-Roots of a Permutation

From the previous section, we see that our problem of finding the permutation key can be reduced to the problem of finding the r-th roots of a given permutation in the symmetric groups, S_{48} and S_{96}, where r is the number of rounds.

Any permutation can be expressed as a product of disjoint cycles, and it is this representation that is most useful when computing roots. In particular, the permutation found through differential cryptanalysis can be expressed as a product of disjoint cycles in S_{48} and S_{96}.

Before describing how to find a root for a permutation in general, we outline the basic ideas. For this, let us first see what happens when we raise a single cycle to the power r.

Let $c = (c_0, c_1, \ldots, c_{l-1})$ be a cycle of length l in S_n. Then c^2 will remain a single cycle when l is odd, namely, $c^2 = (c_0, c_2, \ldots, c_{l-1}, c_1, c_3, \ldots, c_{l-2})$, and will be decomposed into 2 cycles when l is even, namely, $c^2 = (c_0, c_2, \ldots, c_{l-2})(c_1, c_3, \ldots, c_{l-1})$. In general, depending on l, c^r will either remain a single cycle or be decomposed into a number of cycles having the same length (see Lemma 1). Each element c_i will be in a cycle, say, $(c_i, c_{i+r}, c_{i+2r}, \ldots, c_{i+(k-1)r})$, where $i + kr \equiv i \pmod{l}$ and $i + jr$ is reduced modulo l for each j. So in order to find the r-th root we have two cases, the first one is when c^r is a single cycle, and here c^r equals exactly $(c_0, c_r, c_{2r}, \ldots, c_{(l-1)r})$. The second case is when c^r consists of a number of disjoint cycles, and here we combine these disjoint cycles into a single cycle in a certain way in order to get c (see the proof of Theorem 2). To illustrate this, let us find the square root of the permutation $\sigma^2 = (1, 3, 2)(4, 6, 7)(5)(8)$ in S_8. According to the above explanation, we know that cycles of the same length are either a decomposition of a single cycle in the root σ or a reordering of a single cycle in the root σ. Considering cycles of length 1, (5) and (8), it is obvious that they arise from either $(5)(8)$ or $(5, 8)$.

Thus, there are two possibilities for cycles of length 1 in σ. Cycles of length 3, $(1, 3, 2)$ and $(4, 6, 7)$, are either a decomposition of a single cycle in σ, this could be $(1, 4, 3, 6, 2, 7)$, $(1, 6, 3, 7, 2, 4)$ or $(1, 7, 3, 4, 2, 6)$; or a reordering of disjoint cycles in σ and this could only be $(1, 2, 3)(4, 7, 6)$. Summarizing, there are four possibilities for cycles of length 3 in σ. So the total number of square roots for the permutation σ^2 is 8.

4.1 The General Case

The procedure for constructing an r-th root for a permutation, described in [15], is based on the following basic fact in the theory of symmetric groups which can be easily deduced from the previous explanation.

Lemma 1. *Let $C \in S_n$ be a cycle of length l and let r be a positive integer. Then C^r consists of $gcd(l, r)$ disjoint cycles, each of length $\frac{l}{gcd(l,r)}$.*

The following theorem is due to A. Knopfmacher and R. Warlimont [15, p. 148]. We recall its proof, as the proof describes how to construct an r-th root.

Throughout the rest of this paper, we use the notation l-cycle to mean a cycle of length l.

Theorem 2. *[15,1] Let $r = p_1^{i_1} p_2^{i_2} \dots p_n^{i_n}$, where $p_1, p_2, \dots, p_n$ are the prime factors of r. A permutation $Q \in S_n$ has an r-th root, iff for every integer $l \geq 1$, the number of l-cycles in Q is divisible by $((l, r)) := \prod_{\{j:p_j|l\}} p_j^{i_j}$.*

Proof. ($\Leftarrow$): to prove this, we construct an r-th root, R, of Q. Let a_l be the number of l-cycles in Q. Let $g = ((l, r))$. Then $a_l = gm$, where m is an integer, so we can divide the l-cycles of Q into m groups where each group consists of g l-cycles. Assume that we have the cycles, $c_{ij} = (c_{ij}^{(0)}, c_{ij}^{(1)}, \dots, c_{ij}^{(l-1)})$ where $1 \leq i \leq g$ and $1 \leq j \leq m$. For each j, we construct a cycle of length gl, say $R_j = (c_{1j}^{(0)}, c_{2j}^{(0)}, \dots, c_{gj}^{(0)}, c_{1j}^{(d)}, c_{2j}^{(d)}, \dots, c_{gj}^{(d)}, \dots$
$, c_{1j}^{((l-1)d)}, c_{2j}^{((l-1)d)}, \dots, c_{gj}^{((l-1)d)})$, where $d = \frac{r}{g}$ and sd is reduced modulo l for each $1 \leq s \leq l-1$. Now R_j is a cycle of length gl, so according to the previous lemma, R_j^r consists of $gcd(gl, r)$ cycles of length $\frac{gl}{gcd(gl,r)}$. Now, since $g = ((l, r))$, then $gcd(l, \frac{r}{g}) = 1$ and so $gcd(gl, r) = g$, which means that R_j^r consists of g cycles of length l, namely, $c_{1j}, c_{2j}, \dots, c_{gj}$. So $\prod_{j:1 \leq j \leq m} R_j$ is an r-th root for the l-cycles of Q. Repeating the same procedure for all l will yield an r-th root of Q. For the proof of ($\Rightarrow$), see [1]. □

In [6,9], a procedure to find all the roots of Q is described. Going back to the previous theorem, we see that the main property that enables us to construct an r-th root for the l-cycles of Q is having $gcd(gl, r) = g$. Repeating the same procedure for all the g's that satisfy $gcd(gl, r) = g$ will allow us to find all the possible roots that can come from the l-cycles. Note that g is bounded by a_l (the number of l-cycles). To find all the roots, for each group consisting of l-cycles in Q, we proceed as follows.

First we construct the set $G_r(l, a_l) = \{g_i : gcd(g_i l, r) = g_i \text{ and } 1 \leq g_i \leq a_l\}$. Now, this tells us that the roots have cycles of length $g_i l$, but we do not know how many of them. For this, we solve the following Frobenius equation for $x_i \geq 0$:

$$g_1 x_1 + g_2 x_2 + \dots + g_k x_k = a_l \qquad \text{where } k = |G| \tag{1}$$

This equation will usually have more than one solution. Each solution corresponds to a possible cycle structure of the roots. For instance, the solution $x = (x_1, x_2, \dots, x_k)$, tells us that each corresponding root for the l-cycles of Q consists of x_i cycles of length $g_i l$ for $1 \leq i \leq k$.

The efficiency of computing all roots is of course bounded by the total number of roots. If a permutation has a huge number of roots, computing all of them is very time consuming. It is therefore of interest to know the number of roots in advance.

In [9], using the above information about the cycle structure of permutations that have an r-th root, the following explicit formula[2] for calculating the number of all the possible roots is provided.

[2] A more complicated formula was previously found by Pavlov in [11].

Theorem 3. *[9] Let r be a positive integer and $Q \in S_n$. Let a_l be the number of l-cycles in Q, where $1 \leq l \leq n$. Let $X(l, a_l)$ be the set of all the possible solutions of equation* (1)*. Then the number of r-th roots of Q is*

$$\prod_{a_l \neq 0} a_l!\Big(\sum_{x \in X(l,a_l)} \prod_{i=1}^{k} \frac{l^{(g_i-1)x_i}}{g_i^{x_i} x_i!} \Big) \tag{2}$$

where $x = (x_1, x_2, \ldots, x_k)$ and $\{g_i : 1 \leq i \leq k\}$ are the elements of $G_r(l, a_l)$.

To get a feeling of how many roots of a permutation can be expected for the case of PRINTcipher-48, let us take the following permutation in S_{48}, suppose we have

$$\begin{aligned} \tau^{24} =&(1,7,47)(2,19,45)(3,48,17)(4,9,38)(5,16)(6,33,32)(8,28)(10,35)\\ &(11,27,18)(12,20)(14,19,41)(15,46)(21,26,30)(22,34)(23,36)(25,\\ &42,39)(40,44)(13)(24)(31)(37)(43) \end{aligned} \tag{3}$$

So we have $a_1 = 5, a_2 = 8, a_3 = 9$ and $a_l = 0$ for $4 \leq l \leq 48$. $G(1, a_1) = \{1, 2, 3, 4\}$, $X(1, a_1) = \{(0, 1, 1, 0), (1, 0, 0, 1), (1, 2, 0, 0), (2, 0, 1, 0), (3, 1, 0, 0), (5, 0, 0, 0)\}$, $G(2, a_2) = \{8\}$, $X(2, a_2) = \{(1)\}$ and $G(3, a_3) = \{3, 6\}$, $X(3, a_3) = \{(3, 0), (1, 1)\}$. Plugging these values into equation (2), we find that the number of roots is $\simeq 2^{51.3}$. Moreover, the case where τ^{22} is the *Identity* has $\simeq 2^{192}$ roots in S_{48}.

Note that out of all 48!(96!) permutations only a tiny fraction of $2^{32}(2^{64})$ permutations actually correspond to a valid key in PRINTcipher-48(96). We can therefore expect that in the above example out of the $\simeq 2^{51.3}$ only a very small number will actually correspond to a PRINTcipher-permutation. In particular there is only one root for equation (3) that corresponds to a PRINTcipher permutation.

The main purpose of the next section is to describe a method that filters out wrong candidates as soon as possible, allowing to considerably speed up the computation of all valid PRINTcipher-roots.

4.2 PRINTcipher-Roots

As discussed in the last section, computing all the roots of $(PK)^r$ in order to find the right permutation key is inefficient. In this section we describe a method that finds the permutation roots PK belonging to the $2^{32}(2^{64})$ possible permutations in PRINTcipher-48(96). Throughout the rest of this paper, we only discuss PRINTcipher-48 and unless mentioned explicitly, the assumption is that everything about PRINTcipher-48 follows for PRINTcipher-96 with a slight modification.

Our method uses the fact that when we apply the fixed permutation, P, for all $1 \leq i \leq 16$, the 3 bits $i, i + 16$ and $i + 32$ go to the ith Sbox, where depending on the permutation key, they are permuted to only four out of the six possible permutations. So the result of applying the fixed permutation, P, and then applying the keyed permutation, K, on a 48 bits plain text, is a permutation PK that satisfies the following two properties:

1. *Property* 1: For all $1 \leq i \leq 48$, $PK(i)$ equals one of the following three possible values depending on K,

$$PK(i) = \begin{cases} 3i - 2 \pmod{48} & \text{if } 3i - 2 \neq 48) \\ 3i - 1 \pmod{48} & \text{if } 3i - 1 \neq 48) \\ 3i \pmod{48} & \text{if } 3i \neq 48) \end{cases}$$

2. *Property* 2: Only 4 out of the 6 possible 3-bit permutations are valid, namely, $PK(i)$, $PK(i+16)$ and $PK(i+32)$ are permuted to one of the four possible permutations, i.e., for all $1 \leq i \leq 48$, the following two permutations are not allowed:
   (a) $PK(i) = 3i - 1$, $PK(i + 16) = 3i$ and $PK(i + 32) = 3i - 2$.
   (b) $PK(i) = 3i$, $PK(i + 16) = 3i - 2$ and $PK(i + 32) = 3i - 1$.

Definition 1. *A* PRINTcipher *permutation root is any permutation on 48 elements satisfying both Property 1 and Property 2.*

Definition 2. *A* PRINTcipher *permutation(cycle) is any permutation (cycle) on less than 48 elements satisfying both Property 1 and Property 2.*

To explain these definitions, consider the following two cycles $(14, 41, 25, 26, 30, 42, 29, 39, 21)$ and $(14, 42, 30, 41, 25, 26, 29, 39, 21)$. We want to investigate whether these cycles are PRINTcipher cycles or not. The latter cycle satisfies the two properties and so it is a PRINTcipher cycle, in other words it can be part of a PRINTcipher permutation root, PK. The former cycle satisfies only *Property* 1 but not *Property* 2 since we have $PK(14) = 41$ and $PK(30) = 42$ and therefore $PK(42) = 40$, and this is one of the two disallowed permutations (see item (a) in *Property* 2) and so it cannot be part of a valid PRINTcipher permutation root, PK. Sometimes we can have a permutation consisting of two or more cycles having same or different lengths, that satisfies *Property* 1 but not *Property* 2. For example, the following permutation, $(1, 2, 6, 17)(5, 15, 44, 34)$, satisfies *Property* 1 but not *Property* 2 as we have $PK(2) = 6$ and $PK(34) = 5$ and therefore $PK(18) = 4$, which is an invalid PRINTcipher permutation (see item (b) in *Property* 2).

Since the same cycles and permutations can be written in different ways, our method adopts the notion that starts writing each cycle by its smallest element and lexicographically order the disjoint cycles of the same length of a permutation in order to avoid repetitions in the permutation roots of $(PK)^r$. Our method consists of two algorithms: the first one constructs a PRINTcipher cycle of length gl and the second one uses the first algorithm to construct k combined disjoint cycles, each of length gl. In what follows, we shall give a detailed description of the two algorithms and end this section by showing how to use Algorithm 2 to find the whole PRINTcipher permutation roots of $(PK)^r$.

Finding single PRINTcipher cycles Given a_l cycles of length l, the following algorithm constructs all the possible PRINTcipher cycles of length gl beginning with an element called *first* specified in the input (must be one the the first elements in one of these a_l cycles). The algorithm performs a depth first search to find all the other possible $g-1$ cycles with minimal elements larger than *first* and can be combined with the cycle containing *first* as described in Theorem 2 in order to form a PRINTcipher cycle (or just reorder the given cycle in the case $g = 1$ as described previously).

Algorithm 1 finds a PRINTcipher cycle of length gl
find-cycle(cycle, current, g, l-cycles)

```
Require: $l$-cycles numbered from 1 to $a_l$ where $a_l \geq g$
Require: current = first, cycle = first
Ensure: cycle is a PRINTcipher cycle of length $gl$
for count=0 to 2 do
  next = 3 × current - count
  if next $\in$ $l$-cycles then
    if next > first and next.cycleno $\neq$ first.cycleno and cycle.length < $g$ then
      if next.cycleno $\neq$ the cycleno of all the elements of cycle then
        Add next to cycle
        current = next
        Perform again this algorithm on cycle, find-cycle(cycle, current, g, l-cycles)
      end if
    else if cycle.length = $g$ and next.cycleno = first.cycleno then
      Complete the construction of cycle by combining the $g$ different cycles to get a single cycle of length $gl$ as shown in the proof of Theorem 1 (when $g = 1$, reorder the cycle containing first as described previously and assign it to cycle)
      if cycle satisfies Property 2 then
        cycle is a PRINTcipher cycle of length $gl$
      end if
    end if
  end if
end for
```

Plugging all the 2-cycles of equation (3) and setting *first* $= 5$ and $g = 8$ will produce the following PRINTcipher cycle of length 16

$$(5, 15, 44, 36, 12, 35, 8, 22, 16, 46, 40, 23, 20, 10, 28, 34).$$

Algorithm 1 enables us to find a PRINTcipher permutation consisting of only one cycle of length gl but note that some of the x_i's in equation (1) can be more than 1. So we need another algorithm which can find a PRINTcipher permutation consisting of k disjoint cycles where $k \geq 1$.

Finding *k* combined PRINTcipher cycles Given a_l cycles of length l, the following algorithm constructs a permutation beginning with an element called *first* specified in the input (must be the first element in one of these a_l cycles) and consisting of k combined and disjoint cycles ordered lexicographically. It basically performs a recursive depth first search. The recursive algorithm begins by invoking Algorithm 1 which outputs single cycles of length gl beginning with *first*. It then proceeds from each cycle found by Algorithm 1 and concatenates it with the previously $i-1$ concatenated disjoint cycles found after the ith recursive call and if the concatenation satisfies *Property* 2, it recursively calls itself a number of times, each time with a different *first* element to begin the required permutation with as this will enable us to find all the possible $i+1$ disjoint cycles, on a reduced number of l-cycles (exactly $a_l - gi$ cycles) consisting of all the l-cycles except the gi cycles involved on the i concatenated disjoint cycles (in each invocation *first* is set to the smallest element on one of the currently available $a_l - gi$ cycles). Each recursive call stops when $i = k$, or when Algorithm 1 returns nothing, or when each concatenation of i cycles does not satisfy *Property* 2.

Using Algorithm 1 for all the possible g's along with all the possible *first* values and setting $a_1 = 48$, we can find all the possible PRINTcipher cycles. For instance, when $g = 1$ and $a_1 = 48$, Algorithm 1 returns four 1-cycles when trying all the possible values for *first*, namely, (1), (24), (25) and (48). When $g = 2$ and $a_2 = 48$, we found that there are six possible 2-cycles, namely, $(6, 18)$, $(7, 19)$, $(12, 36)$, $(13, 37)$, $(30, 42)$ and $(31, 43)$. When $g = 3$, we found there are eight possible 3-cycles. This information enables us to reduce the size of the cycle structure of the roots by removing any structure containing more than four 1-cycles, six 2-cycles and eight 3-cycles. It also enables us to easily find some roots, for example, knowing all PRINTcipher cycles of length 1 and 2, we can easily find that $(24)(13, 37)(30, 42)$ is a PRINTcipher permutation that is a root for the 1-cycles of equation (3).

Moreover, using Algorithm 2 we find that we cannot have a permutation consisting of more than 6 disjoint cycles of length 5 in PRINTcipher-48 and not more than 9 cycles of length 4, 12 cycles of length 5, 13 cycles of length 6 and 12 cycles of length 7 in PRINTcipher-96. This will generally reduce the number of solutions of equation (1) and therefore the size of the cycle structure which will speed up the process of finding PRINTcipher permutations roots.

Finding PRINTcipher permutations Now, when given a_l cycles of length l, Algorithm 2 enables us to find PRINTcipher permutations beginning with a specified element and consisting of k cycles, each of length gl. But in order to find the rth permutation roots for all the l-cycles we use Algorithm 2 together with the elements of the sets $G(l, a_l)$ and $X(l, a_l)$. Each entry $x_j = (x_{j1}, x_{j2}, \ldots, x_{jk}) \in X(l, a_l)$ where $k = |G(l, a_l)|$, represents the cycle structure of many rth roots for the l-cycles and it might correspond to few or none PRINTcipher permutations, so for each $x_j \in X(l, a_l)$, we try to find all the possible PRINTcipher permutations beginning with a specific element called

Algorithm 2 finds a PRINTcipher permutation that has k disjoint gl-cycles *find-k-cycles(C, current, k, g, l-cycles)*

```
Require: $l$-cycles numbered from 1 to $a_l$ where $a_l \geq g$
Require: current = first
Require: $C = \{\}$
Ensure: $k$ disjoint PRINTcipher $gl$-cycles, or return $\{\}$ if there is no $k$ disjoint
    PRINTcipher cycles
Invoke Alg. 1 on the current $l$-cycles
if number of cycles found by Alg. 1 $> 0$ then
    if the number of disjoint cycles in $C$ consists of $k - 1$ disjoint cycles then
        for each permutation cycle found by Alg. 1 do
            $C = C \cup cycle$
            if $C$ satisfies Property 2 then
                return $C$
            else
                return $\{\}$
            end if
        end for
    else
        for each cycle found by Alg. 1 do
            $C = C \cup cycle$
            if $C$ satisfies Property 2 then
                Delete all the $l$-cycles involved in $C$ from the $a_l$ cycles of length $l$
                for each $cycle \in$ currently available $l$-cycles do
                    {Perform again this algorithm on the current $l$-cycles to find the other
                    $k - 1$ cycles}
                    current = first element in cycle
                    find-k-cycles(C, current, k, g, l-cycles)
                end for
            end if
        end for
    end if
else
    return
end if
```

first (must be the first element in one of these a_l cycles) and that can be roots for the l-cycles by applying Algorithm 2 through all the nonzero entries of x_j. Trying all the possible values for *first* gives us all PRINTcipher permutations that are roots for all the l-cycles.

Now, assume that we find all the possible PRINTcipher permutations for each l, say σ_{l_i}, for $1 \leq i \leq \eta_l$ where η_l is the number of permutation roots of the l-cycles of $(PK)^r$, so all the possible products $\prod_{a_l>0} \sigma_{l_i}$ where $1 \leq l \leq 48$ and $1 \leq i \leq \eta_l$, represent the PRINTcipher permutation roots which are the possible values for PK and by brute forcing these PK values we can recover the permutation key, K.

Let us try to find PRINTcipher permutations that are roots for the nine 3-cycles in equation (3). We have $G(3, a_3) = \{3, 6\}$ and $X(3, a_3) = \{(3, 0), (1, 1)\}$. We start with, $x_1 = (3, 0)$, here we only need to apply Algorithm 2 using any possible *first* because the 3 disjoint cycles of length 9 would come from all the 9 cycles. Setting *first* $= 1$ and applying Algorithm 2 doesn't give us 3 disjoint cycles of length 9, so we conclude that there is no root having the cycle structure x_1. So we go to the next cycle structure, $x_2 = (1, 1)$, we start with $x_{21} = 1$ and use Algorithm 2 on all the possible *first* values. Setting *first* $= 1, 2, 3, 4, 6$ and 11 doesn't yield a single cycle of length 9, while *first* $= 14$ yields the cycle $(14, 42, 30, 41, 25, 26, 29, 39, 21)$, we save it and continue to the next element $x_{22} = 1$ where we use Algorithm 2 on the 6 cycles that are not involved in the previous found cycle. Now we want to construct a cycle of length 18, so all the 6 cycles would be involved in it, setting *first* $= 1$, yields $(1, 2, 4, 11, 33, 3, 7, 19, 9, 27, 32, 48, 47, 45, 38, 18, 6, 17)$. Concatenating this cycle with the previous found cycle, we get $(14, 42, 30, 41, 25, 26, 29, 39, 21)(1, 2, 4, 11, 33, 3, 7, 19, 9, 27, 32, 48, 47, 45, 38, 18, 6, 17)$ which satisfies *Property* 2. This means that it is a PRINTcipher permutation that is a root for all the 3-cycles in equation (3). Now, we have found the roots for all the l-cycles in equation (3). Concatenating them together gives us the following PRINTcipher permutation root: $(1, 2, 4, 11, 33, 3, 7, 19, 9, 27, 32, 48, 47, 45, 38, 18, 6, 17)(5, 15, 44, 36, 12, 35, 8, 22, 16, 46, 40, 23, 20, 10, 28, 34)(14, 42, 30, 41, 25, 26, 29, 39, 21)(13, 37)(30, 42)(24)$.

5 Experimental Verifications

To demonstrate the efficiency of our attack we implemented the above algorithms. Experiments show that $(PK)^r$ could yield more than one PRINTcipher root when $(PK)^r$ contains several 1-cycles, but in most cases there was exactly one PRINTcipher root.

To derive bounds for the number of PRINTcipher permutations roots, we computed the number of all PRINTcipher permutation roots for $(PK)^r =$ *Identity* where $2 \leq r \leq 22$. This seems the worst case that could happen for any r since $a_1 = 48$, which is a_1's largest value, and as shown in Table 2, the number of PRINTcipher roots when $r = 22$ is $2^{22.04}$. These roots are found within less than 3 hours on a standard PC.

Furthermore, we tried 10^4 random PRINTcipher-48 permutation keys excluding the ones that yield $(PK)^r = \textit{Identity}$. Note that, for a random key, the probability for the worst case is $\frac{2^{22.04}}{2^{32}} = 0.001$ for 22 rounds and less than that for $r < 22$. These experiments took a few seconds on average on a standard PC and they show that most of the time there is a unique PRINTcipher permutation root. Table 2 shows the number of keys (n_k), out of the 10^4 random keys, that yield more than one PRINTcipher permutation root. It also shows the number of PRINTcipher permutation roots in the worst case (n_w) for each number of rounds.

r	$\log_2 n_k$	$\log_2 n_w$	r	$\log_2 n_k$	$\log_2 n_w$	r	$\log_2 n_k$	$\log_2 n_w$
2	-	-	9	7.66	8.58	16	10.80	18.95
3	-	-	10	8.33	11.90	17	8.71	-
4	6.11	2	11	7.94	9.31	18	11.16	20.67
5	2	-	12	11.46	17.39	19	8.77	-
6	9.30	4.17	13	8.47	-	20	10.68	21.54
7	3.70	-	14	9.10	16.27	21	9.18	18.73
8	9.59	10.07	15	9.77	16.63	22	9.59	22.04

Table2. Results of the 10^4 trials and the worst case for $2 \leq r \leq 22$, $n_k \equiv$ the number of keys that yield more than one PRINTcipher permutation root, $n_w \equiv$ the number of PRINTcipher permutation roots in the worst case.

6 Conclusions

We have described two differential attacks against 22 rounds of PRINTcipher-48, requiring the full code book and about 2^{48} computational steps. While this is far from breaking the full 48 rounds of the cipher, it is the best currently known result against the cipher. Similar results can be obtained for the 96-bit version of the cipher.

One of the attacks is a new technique targeting the key-dependent permutations used in PRINTcipher. Since such key-dependent permutations are currently not well-studied, the attack is of importance to past and future designs that use them. We introduced a novel technique for computing permutation roots, making it possible to retrieve the key-dependent single-round permutation π given nothing but the r-round permutation π^r and the cipher description. While our technique so far applies only to the case where the linear layer is a (key-depended) bit permutation, future designers of cryptographic primitives using key-dependent permutations should be aware of this technique when choosing parameters like round numbers or S-box layout for their algorithms.

References

1. Scott Annin and Trenton Jansen. On kth roots in the symmetric and alternating groups. *Pi Mu Epsilon Journal*, 12(10):581–589, 2009.
2. Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.
3. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsø. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
4. Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.
5. Henri Gilbert and Pascal Chauvaud. A chosen plaintext attack of the 16-round Khufu cryptosystem. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO '94, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 359–368. Springer, 1994.
6. Anja Groch, Dennis Hofheinz, and Rainer Steinwandt. A practical attack on the root problem in braid groups. In *Algebraic methods in cryptography*, volume 418, pages 121–132. American Mathematical Society, 2006.
7. Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A new block cipher suitable for low-resource device. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006.
8. Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A block cipher for IC-printing. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.
9. Jesús Leaños, Rutilo Moreno, and Luis M. Rivera-Martínez. A note on the number of m-th roots of permutations. *Arxiv preprint arXiv:1005.1531*, 2010.
10. Ralph C. Merkle. Fast software encryption functions. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO '90, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 476–501. Springer, 1991.
11. A. I. Pavlov. On the number of solutions of the equation $x^k = a$ in the symmetric group S_n. *Mathematics of the USSR-Sbornik*, 40(3):349–362, 1981.
12. Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In Ross J. Anderson, editor, *Fast Software Encryption 1993, Proceedings*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 1994.
13. Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher. Submitted as candidate for AES. Available: `http://www.schneier.com/paper-twofish-paper.pdf` (2010/02/05).
14. Serge Vaudenay. On the weak keys of Blowfish. In Dieter Gollmann, editor, *Fast Software Encryption 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 27–32. Springer, 1996.
15. Herbert S. Wilf. *Generatingfunctionology.* Academic Press, 1993.