Study of the invariant coset attack on PRINTcipher: more weak keys with practical key recovery

Stanislav Bulygin[1,2] and Michael Walter[1]

[1] Center for Advanced Security Research Darmstadt - CASED
Mornewegstraße 32, 64293 Darmstadt, Germany
Stanislav.Bulygin@cased.de

[2] Technische Universität Darmstadt, Department of Computer Science
Hochschulstraße 10, 64289 Darmstadt, Germany
michael.walter@swel.com

Abstract. In this paper we investigate the invariant property of PRINTcipher first discovered by Leander *et al.* in their CRYPTO 2011 paper. We provide a thorough study of the question, showing that there exist 64 classes of weak keys for PRINTcipher–48 and many more for PRINTcipher–96. We show that for many classes of weak keys the key recovery can be done in a matter of minutes in the chosen plaintext scenario. In fact, at least 2^{48} weak keys can be recovered in less than 20 minutes per key on a single PC using only a few chosen plaintexts. We provide detailed treatment of the methods and put them in a more general context that opens new interesting directions of research for PRESENT-like ciphers.

Keywords: PRINTcipher, invariant coset attack, mixed integer linear programming, weak keys, chosen plaintext attack, key recovery

1 Introduction

Lightweight cryptography gained its importance due to emergence of many applications that involve using small and resource constraint devices like RFID tags, smart cards, and sensors. Conventional cryptographic algorithms turned out to be too massive to be implemented on such devices. Therefore the need for new cryptographic primitives arose in the community. In particular, the whole arsenal of lightweight block ciphers has been developed in recent years to satisfy the needs for secure usage of small devices. The block cipher PRESENT is one outstanding example that gained popularity [1]. Other block ciphers, such as KATAN and KTANTAN family [2], LED block cipher [3] and many others were presented recently. Following the design principle of PRESENT, several block ciphers with even more lightweight structure have been proposed. PRINTcipher [4] and EPCBC [5] are immediate examples, as well as SPONGENT hash family [6].

PRINTcipher is a block cipher proposed at CHES 2010 [4] and is really pushing the design solutions for lightweight ciphers to their limits. It has been designed with the technology of integrated circuit (IC) printing in mind that provides an opportunity to print cryptographic circuitry on different surfaces, such as thin films, that can be attached to a product to give it cryptographic protection. PRINTcipher has been an object of numerous attacks since then. Methods of linear and differential cryptanalysis [7,8,9] as well as algebraic cryptanalysis [10] have been proposed to analyze PRINTcipher. Also certain results on side channel analysis of PRINTcipher appeared [10,11]. Notably, cryptanalytic methods proposed so far were not

able to break more than $\frac{2}{3}$ of PRINTcipher's rounds[1].
At CRYPTO 2011 Leander *et al.* proposed a very powerful attack, which they called the *invariant coset attack* [12]. Using this attack it is possible to break the *full* PRINTcipher (both the 48- and the 96-bit versions) for a large portion of weak keys using only a few chosen plaintexts. Note, however, that despite the fact that distinguishing a weak key can be done in unit time, the complete key recovery still is a challenging task in practice (see specific figures in Section 3.4). The authors of [12] presented two families of weak keys for PRINTcipher–48, each having 2^{51} keys as well as two families for PRINTcipher–96.
This paper has initially been motivated by the problem of finding optimal propagations of known bits to facilitate algebraic attacks using e.g. SAT-solvers following the ideas of [10]. With the methods employed, however, we ended up providing a thorough study of the invariant coset attack. In particular, we were able to recover *all* 64 classes of weak keys for PRINTcipher–48. The same method can be applied to PRINTcipher–96 as well. We observed that among the found 64 classes, many classes are composed of weak keys that can be recovered in a matter of minutes on a single PC. Thus we provide a key recovery for many weak keys that can be done very efficiently for a cipher that is otherwise quite secure. Following [12] we analyze protecting techniques against the attack and show that the situation for PRINTcipher–96 is not obvious. Nevertheless, it is still possible to counter the attacks quite easily. Summing up, in this paper we provide a complete study of the invariant coset attack of [12] with methods that have value on their own and potentially can be employed to obtain interesting results on other similar ciphers.
The outline of the paper is as follows. In Section 2 we briefly recall the definition of PRINTcipher and outline its properties that are relevant for the further exposition; we also recall the attack of [12]. Section 3 presents methods of obtaining and using invariant cosets (or *invariant projected subsets* as we call them). Section 3.1 presents the general background, as well as the first methods to find invariant projected subsets. Section 3.2 develops the idea further and proposes an alternative method based on Mixed Integer Linear Programming that turns out to be highly efficient in practice. Section 3.3 summarizes the results for PRINTcipher–48 obtained by our methods, listing all invariant projected subsets (their *defining sets* to be precise), computing the number of the corresponding weak keys in each class and complexity of key recovery, protecting measures, and outlining results for PRINTcipher–96. Section 3.4 is explicitly considering the situation when the weak key recovery is particularly fast, providing specific figures that estimate practical key recovery using only very moderate computational resources. Finally, Section 4 presents a retrospective of the proposed methods and puts them in a more general context providing possible further directions for research.

2 The block cipher PRINTcipher

2.1 Description of the cipher

PRINTcipher [4] is a substitution-permutation network, which design is largely inspired by the block cipher PRESENT [1]. The main differences to PRESENT is the absence of the key schedule (all round keys are the same and are equal to the master key) and key-dependent S-Boxes. PRINTcipher comes in two variations: PRINTcipher-48 encrypts 48-bits blocks with an 80-bit key and has 48 rounds, PRINTcipher-96 encrypts 96-bit blocks with a 160-bit key

[1] The best attack of [9] can break 31 out of 48 rounds of PRINTcipher–48 for the fraction of 0.036% of keys using almost the entire code book.

and has 96 rounds. Here we present a short overview of the cipher, referring the reader to [4] for a more detailed description and analysis.

The encryption process of PRINTcipher–n for $n = 48, 96$ is organized as in Algorithm 1.

Algorithm 1 Encryption function of PRINTcipher–n

Require:
- n-bit plaintext p
- $\frac{5}{3}n$-bit key $k = (sk_1, sk_2)$, where sk_1 is n bits and sk_2 is $\frac{2}{3}n$ bits

Ensure: n-bit ciphertext c

```
Begin
state := p
for i = 1, ..., n do
    state := state ⊕ sk_1
    state := Perm(state)
    state := state ⊕ RC_i
    state := SBOX(state, sk_2)
end for
c := state
return c
End
```

Some comments to Algorithm 1 follow. The linear diffusion layer $Perm$ implements a bit permutation similar to PRESENT and is given by the map P:

$$P(i) = \begin{cases} 3i \bmod n-1 & \text{for } 0 \le i \le n-2, \\ n-1 & \text{for } i = n-1, \end{cases} \tag{1}$$

so that the $i-$th bit of the state is moved to the position $P(i)$. The round counter RC_i for $i = 1, \ldots, n$ is a 6-bit binary representation of i that is placed in the last two triplets. The S-Box layer $SBOX$ is a layer of $n/3$ 3-bit S-Boxes, where each S-Box is chosen according to the value of the two corresponding bits of the subkey sk_2. Therewith, there are 4 possible S-Boxes at each position called V_0, V_1, V_2, V_3 in [4]. One may also consider such an S-Box as a composition of a key dependent bit permutation that acts on groups of three bits and then followed by the layer of fixed S-Boxes, each one being a 3-bit S-Box with the truth table as in Table 1. This S-Box, called V_0 in [4], is preceded by a key-dependent permutation defined by Table 2. In Table 2 the three input bits are permuted according to the two consecutive key bits from the subkey sk_2 called a_0 and a_1. Figure 1 provides an illustration for one encryption round of PRINTcipher-48.

Table 1. Truth table for the S-Box V_0.

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

We need properties of 1-masks that exist for keyed S-Boxes of PRINTcipher. Table 3 shows that each possible 1-mask on the input bits (i.e. when two input bits are known at some positions) has a corresponding 1-mask for output bits (i.e. two output bits are known). In this table $+/-$ notation shows which bits of the mask are known ($+$) and ones which are

Table 2. Key depended permutation

a_1	a_0	Permutation
0	0	(0,1,2)
0	1	(0,2,1)
1	0	(1,0,2)
1	1	(2,1,0)

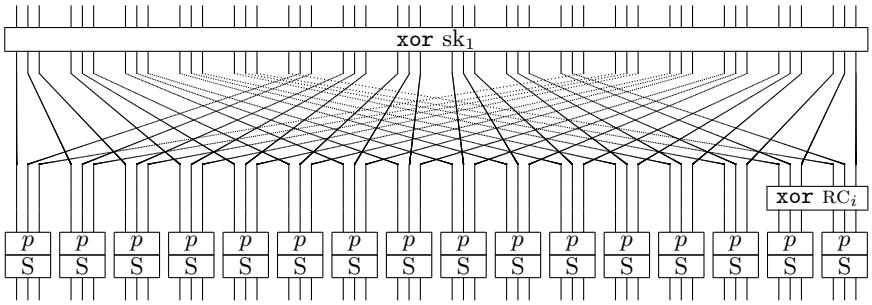


Fig. 1. Round function of PRINTcipher-48, cf. Figure 1 of [4].

not $(-)$. For example, the first row says that if we have a mask where the first two bits of both input and output to a keyed S-Box are known, then both S-Boxes with $a_0 = 0$ satisfy the mask. Moreover, the two known output bits of the mask, as well as the two input bits, must be 0. Table 3 plays an important role in studying the invariant coset attack of [12].

Table 3. 1-masks for PRINTcipher S-Boxes

Input mask	Output mask	Values of (a_0, a_1)	log(# valid a-pairs)	input values	output values
$++-$	$++-$	0*	1	00*	00*
$++-$	$+-+$	10	0	10*	1*1
$++-$	$-++$	11	0	11*	*10
$+-+$	$++-$	10	0	0*0	00*
$+-+$	$+-+$	$a_0 = a_1$	1	0*1 or 1*0	1*1
$+-+$	$-++$	01	0	1*1	*10
$-++$	$++-$	11	0	*00	00*
$-++$	$+-+$	01	0	*10	1*1
$-++$	$-++$	*0	1	*11	*10

2.2 Invariant coset attack of Leander *et al.*

In their work [12] Leander *et al.* showed that for PRINTcipher there exist subsets of plaintexts which, when encrypted with keys from certain other subsets, end up in the same subset. Thus they showed an invariant property for PRINTcipher under certain weak keys. The subsets they presented are of quite special form: they are linear cosets. The invariant coset for the plaintexts is of the form $U + d$ for some linear subspace $U \leq \mathbb{F}_2^n$ and some vector $d \in \mathbb{F}_2^n$; the weak keys are from the coset $U + c + d$ for some other vector $c \in \mathbb{F}_2^n$. Notably, the U's

are linear subspaces of a special kind. Namely, they are linear subspaces where all vectors have the value 0 at certain positions. The vectors c and d then "adjust" the zeros, so that the invariant property holds. The authors of [12] show that distinguishing the weak keys so obtained can be done using only a few chosen plaintexts, thus presenting a powerful attack. In the *invariant coset attack* they heavily use the property of 1-masks presented in Table 3. In [12] two classes of weak keys for PRINTcipher–48 are presented, both having 2^{51} weak keys. In this paper we study the invariant coset attack further exploiting many more new classes of weak keys.

3 Obtaining and using invariant projected subsets

In this section we describe methods of obtaining all invariant cosets for both versions of PRINTcipher. These invariant cosets are of the same type as in [12], i.e. we only consider cosets that are described by the "projection" equations specifying values of vectors at certain positions. In Section 3.1 we present the first method based on orbits of $\mathbb{Z}_n$ under the permutation $\Pi_{sk_2} = P \circ \pi_{sk_2}$ for some permutation key sk_2. In Section 3.2 we suggest another method that is based on describing invariant cosets via polytopes in $\mathbb{Z}^n$ and show its connection to the previous method. In Section 3.3 we provide specific results, listing all classes of weak keys (defining sets of invariant projected subsets to be precise) for PRINTcipher–48 and outlining results for PRINTcipher–96. We provide the number of elements in each class, thus providing an estimate on the overall number of weak keys. We also provide complexity of the key recovery for each class. In Section 3.4 we show that for a considerable number of keys, the key recovery can be done in practical time using only very modest computational resources.

3.1 Invariant projected sets via orbits

In this subsection we investigate the structure of invariant cosets of PRINTcipher from the point of view of orbits under the action of the augmented permutation layer Π_{sk_2}.
Let $n \in \{48, 96\}$ be the block size and $l \in \{80, 160\}$ be the key size of PRINTcipher-n. Next, let P be the bit permutation defined by (1) and let π_{sk_2} be a permutation defined as a concatenation of permutations of bit triplets according to the permutation key $sk_2 \in \{0,1\}^{l-n}$. Then $\Pi_{sk_2} = P \circ \pi_{sk_2}$ is the augmented permutation. So now the S-Box layer S does not depend on the key. Note that we will often be interested in the augmented permutation Π_{sk_2} rather than the augmented linear map $P \circ RC_i \circ \pi_{sk_2}$, since the round counter will not affect positions our subsets are projected on. We do a distinction, though, when the role of RC_i is important. Now, let S_n be the symmetric group defined on n elements. Obviously $\Pi_{sk_2} \in S_n$, i.e. Π_{sk_2} is a permutation of the n positions in a block. It is a well-known fact that each element of S_n is a product of disjoint *cycles* and such a product is unique up to permutation of its factors. Recall that a *cycle* is a permutation of the form $(i_1 \dots i_t)$ for some $0 \leq i_1 < \dots < i_t \leq n-1$, i.e. i_1 gets permuted to i_2, i_2 to i_3 and so on, and i_t permutes to i_1; other elements stay intact.

Definition 1. *Let $sk_2 \in \{0,1\}^{l-n}$ and $a \in \mathbb{Z}_n$, then an* orbit *of a w.r.t sk_2 is a subset $O_a^{sk_2} := \{i \in \mathbb{Z}_n | \exists j \geq 0 : i = \Pi_{sk_2}^{(j)}(a)\} \subset \mathbb{Z}_n$. Here $\Pi_{sk_2}^{(j)}(a) := \Pi_{sk_2}(\dots \Pi_{sk_2}(a) \dots)$ is the composition of Π_{sk_2} j times.*

We have a natural correspondence between the cycles in the representation of Π_{sk_2} and orbits w.r.t sk_2. Indeed, if $(i_1 \dots i_t)$ is a cycle in the representation of Π_{sk_2}, then $O_{i_1}^{sk_2} = \dots =$

$O_{i_t}^{sk_2}$ is the orbit of i_1 (as well as of $i_2, \ldots, i_t$) w.r.t sk_2.
Next we define the type of subsets we will be dealing with.

Definition 2. *A* projected subset $U \subset \mathbb{F}_2^n$ *is defined as*

$$U := \{u = (u_0, \ldots, u_{n-1}) \in \mathbb{F}_2^n | u_{i_1} = a_1, \ldots, u_{i_t} = a_t \text{ for some } t \geq 0 \text{ and } 0 \leq i_1 < \cdots < i_t \leq n-1 \text{ and some } a = (a_1, \ldots, a_t) \in \mathbb{F}_2^t\}.$$

We define $def(U)$ *to be the subset of indexes* $i_1, \ldots, i_t$ *with the above property and for such a subset we define* $val(U) = a$. *We also define* ind_U *on* $def(U) = \{i_1, \ldots, i_t\}$ *as* $ind_U(i_j) = j$. *We call* $def(U)$ *the* defining set *of* U.

Definition 3. *Let* $V \subset \mathbb{Z}^n$ *with* n *divisible by 3, then* V *is a* $\bar{1}-$subset *iff* $\forall\ 0 \leq j < n/3 : |\{3j, 3j+1, 3j+2\} \cap V| \neq 1$.

The above definition calls a subset of positions a $\bar{1}-$subset iff in each consecutive triplet of positions there are 0, 2, or 3 elements from that subset.

Definition 4. *Let* U *be a projected subset with the defining set* $def(U)$. *For* $s \in \{2, 3\}$ *define*

$$U_s := \bigcup_{\substack{0 \leq j < n/3 \\ |\{3j, 3j+1, 3j+2\} \cap def(U)| = s}} \big[\{3j, 3j+1, 3j+2\} \cap def(U)\big].$$

Now let $E_{sk_1, sk_2, i} = XOR_{sk_1} \circ P \circ RC_i \circ \pi_{sk_2} \circ S$ be the round function of PRINTcipher-n for the round $1 \leq i \leq n$.
The next proposition collects properties of projected subsets, in case they define invariant cosets for PRINTcipher.

Theorem 1. *If* $E_{sk_1, sk_2, i}(U) = U$ *for all* $1 \leq i \leq n$ *for some* $sk_1 \in \mathbb{F}_2^n, sk_2 \in \mathbb{F}_2^{l-n}$ *and some projected subset* $U \subset \mathbb{F}_2^n$, *then:*

a. $\Pi_{sk_2}(def(U)) = def(U)$;
b. $def(U) \cap \{n-6, n-5, \ldots, n-1\} = \oslash$;
c. $def(U)$ *is a union of orbits w.r.t* sk_2;
d. Both $def(U)$ *and* $P(def(U))$ *are* $\bar{1}$*-subsets such that* $\forall\ 0 \leq j < n/3 : |\{3j, 3j+1, 3j+2\} \cap def(U)| = |\{3j, 3j+1, 3j+2\} \cap P(def(U))|$.

Proof. a. We have $XOR_{sk_1}(U) = V$, where V is also a projected subset with $def(V) = def(U)$. Next, $(P \circ RC_i \circ \pi_{sk_2})(V) = W$, where W is again projected with $def(W) = \Pi_{sk_2}(def(V)) = \Pi_{sk_2}(def(U))$. Finally, we have at the end of the round i that $U' = S(W)$, which can be written as

$$(U'_0, \ldots, U'_{n/3-1}) = (S(W_0), \ldots, S(W_{n/3-1})),$$

by dividing the state into groups of three bits and applying the S-Box S ($= V_0$, see Section 2) to each triplet. Now note that since W is projected, so is $W_i \subset \mathbb{F}_2^3$ for each i. Also we know that we must have $U' = U$ and so U' as well as all U'_i are projected. So the question is now, under which conditions does the S-Box S map a projected subset of $\mathbb{F}_2^3$ to a projected subset. Note that S is now the "keyless" S-Box V_0 with the truth table as in Table 1 that is defined by $a_0 = a_1 = 0$. Moreover, observe that for S only the 1-masks

$$++- \to ++-, \quad +-+ \to +-+, \quad -++ \to -++$$

are possible, see rows 1, 5, 9 of Table 3. Now, obviously, $\mathbb{F}_2^3$ is mapped to $\mathbb{F}_2^3$ and it is projected. Since there are no other 1-masks for S and no 2-masks, we conclude that $def(U'_i) = def(W_i)$ for all i. Therewith, $def(U) = def(U') = def(W) = \Pi_{sk_2}(def(U))$ as claimed.
b. Assume there exists $j \in \{n-6, \ldots, n-1\} \cap def(U)$. So j is an output position to one of the last two S-Boxes. Note that the value $val(U)_{ind_U(j)}$ is fixed due to the invariant property. Next, j belongs to some $def(U'_h)$ for h either $n/3-2$ or $n/3-1$. In case $|def(U'_h)| = 2$ we know that the input of a 1-mask is defined by the output. This is impossible, since the action of RC_i for i such that $+1$ is added to an input of the S-Box h produces a different input value than the one where addition $+0$ at that input position. The case $|def(U'_h)| = 3$ is obvious: we then have two different inputs to S that should yield the same output, which is impossible.
c. Assume there exists an orbit O w.r.t. sk_2 such that there exist $i \in O \cap def(U)$ and $j \in O \setminus def(U)$. By the definition of O we have that there exists $\alpha : j = \Pi^{(\alpha)}_{sk_2}(i) \in \Pi^{(\alpha)}_{sk_2}(def(U)) = def(U)$, where the last equality is due to (a.). So we have that $j \in def(U)$, which is a contradiction.
d. Follows from the proof of $[a.]$, the part discussing the subsets U'_i.

Note that in the case Theorem 1 holds, we have $E_{sk_1,sk_2,i}(V+d) = V+d$ for a certain linear subspace $V \leq \mathbb{F}_2^n$ and $d \in \mathbb{F}_2^n$. We see now that $V+d$ is an *invariant coset* as per [12]. In order to follow our terminology, we prefer the term *invariant projected subset*.
Now we go in the opposite direction and answer the question, when a certain subset of $\mathbb{F}_2^n$ is a defining set for an invariant projected subset.

Theorem 2. *Let $T \subset \mathbb{Z}_n$ with n divisible by 3 and*

1. *$T \cap \{n-6, \ldots, n-1\} = \oslash$;*
2. *T is a $\bar{1}-$subset of $\mathbb{Z}_n$.*
3. *$P(T)$ is a $\bar{1}-$subset of $\mathbb{Z}_n$.*
4. *$\forall\ 0 \leq j < n/3 : |\{3j, 3j+1, 3j+2\} \cap T| = |\{3j, 3j+1, 3j+2\} \cap P(T)|$.*

Then there exists an invariant projected subset $U \subset \mathbb{F}_2^n$ with $def(U) = T$. Moreover, $\{val(U)_{ind_U(i)} : i \in U_2\}$ is uniquely determined.

Proof. We want to show that there exist $sk_1 \in \mathbb{F}_2^n$ and $sk_2 \in \mathbb{F}_2^{l-n}$ such that $E_{sk_1,sk_2,i}(U) = U$ for all $1 \leq i \leq n$ for some projected subset U with $def(U) = T$.
First, define U as a projected subset with $def(U) = T$. We will specify $val(U)$ later. For the construction, we browse through triplets corresponding to inputs of S-Boxes. Due to properties (2.) and (3.) we are guaranteed that for each keyed S-Box we have 0, 2, or 3 active bits in both input (positions from $P(T)$) and output (positions from T) and the numbers of active bits in input and output to a given S-Box are the same (property (4.)). Now let us look closely at the case $|\{3j, 3j+1, 3j+2\} \cap T| = |\{3j, 3j+1, 3j+2\} \cap P(T)| = 2$ for some $0 \leq j < n/3$. This exactly corresponds to a 1-mask situation. Here Table 3 comes in hand. From this table we see that each possible combination of two active input/output bits for an S-Box is satisfied for some values of $sk_{2,2j}$ and $sk_{2,2j+1}$. Moreover, output values for active bits are now fixed. These fixed values are assigned to corresponding bits of $val(U)$ (recall that $def(U) = T$ for the subset U we want to construct). For input values there exists ambiguity in the case of $+-+ \to +-+$ mask. In this case take $sk_{2,2j} = sk_{2,2j+1} = 0$ and therewith the input mask will be $0 * 1$. The positions of T just considered correspond to U_2 of $def(U) = T$.
In the case $|\{3j, 3j+1, 3j+2\} \cap T| = |\{3j, 3j+1, 3j+2\} \cap P(T)| = 3$ we assign corresponding 3 bits of $val(U)$ arbitrary values, assign arbitrary values to $sk_{2,2j}$ and $sk_{2,2j+1}$. Therewith, we

obtain input values of the j-th S-Box that correspond to the three active bits of $P(T)$. The positions of T just considered correspond to U_3 of $def(U)$. From the proof of Theorem 1, (a.) we have that $def(U) = U_2 \cup U_3$. At this point $val(U)$ is completely defined and so is U.
In the case $|\{3j, 3j+1, 3j+2\} \cap T| = |\{3j, 3j+1, 3j+2\} \cap P(T)| = 0$ we just assign arbitrary values to $sk_{2,2j}$ and $sk_{2,2j+1}$.
We observe that input bits to keyed S-Boxes (corresponding to $P(T)$) match active bits of T and thus output bits of the S-Boxes. Note that all these active bits are assigned specific values at this point. Therewith we can uniquely determine values of sk_1 at positions given by $T = def(U)$. Other values of sk_1 can be assigned arbitrarily.
Now it is clear that we have $E_{sk_1,sk_2,1}(U) = U$ for so constructed U, sk_1, sk_2. Property (1.) guarantees then that $E_{sk_1,sk_2,i}(U) = U$ for all i.

Remark 1. Note that for a subset $T \subset \mathbb{Z}_n$ that satisfy (1.)–(4.) from Theorem 2 it holds that T is a union of orbits w.r.t sk_2 for some $sk_2 \in \mathbb{F}_2^{l-n}$ or, equivalently, $T = \Pi_{sk_2}(T)$ for some sk_2.

As a result we have the following one-to-one correspondence shown in Figure 2.

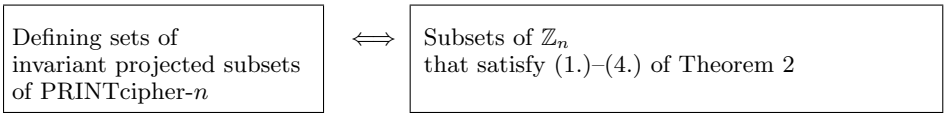


Fig. 2. One-to-one correspondence between defining sets of invariant projected subsets and subsets from Theorem 2.

Now that we have the correspondence in Figure 2 and Remark 1, we may identify two possible ways of finding defining sets for invariant projected subsets and therewith classes of weak keys.

1. Start with an sk_2-candidate and look for orbits w.r.t. this sk_2 that are not active in the last 6 bits and then check conditions (2.)–(4.) for each such orbit.
2. Start with a candidate $\bar{1}-$subset T and check if conditions (1.)–(4.) hold.

We present the solution from (1.) in Algorithm 2. The algorithm uses a subroutine Inv_sk_2 that is described in Algorithm 3. Algorithm 3 uses the function $check_\bar{1}_property$ that checks properties (2.)–(4.) from Theorem 2. Regarding the complexity of the algorithm, note that we need to search through $\mathbb{F}_2^{l-n-4}$ for the part of the subkey sk_2 that defines all S-Boxes except for the last two. Now in the for-loop of Algorithm 2 the defining factor of complexity is calling Inv_sk_2, see Algorithm 3. In its turn, the complexity of Algorithm 3 is defined by the average number of orbits in $\mathbb{Z}^n$ w.r.t. a candidate sk_2. For $n = 48, l = 80$, we have that $l - n - 4 = 28$ and the algorithm terminates in a reasonable time. However, for $n = 96, l = 160$, we have $l - n - 4 = 60$ and it is infeasible to get all possible $def(U)$ in a reasonable time.

Solution (2.) from the above list obviously admits a brute force approach that has complexity similar to the one of Algorithm 2. We develop this approach further in the next subsection showing an existence of a more elegant solution.

Algorithm 2 Algorithm for finding defining sets of invariant projected subsets via orbits

Require:
- Block length n of PRINTcipher-n

Ensure: List of defining sets of invariant projected subsets of PRINTcipher-n

Begin
$l := 5/3 \cdot n$
$inv_sets := []$
for $sk_2 \in \{0,1\}^{l-n-4}$ **do**
  Append sk_2 with random values to a vector in $\{0,1\}^{l-n}$
  Construct $\Pi_{sk_2} = P \circ \pi_{sk_2}$
  Add $Inv_sk_2(n, \Pi_{sk_2})$ to inv_sets
end for
return inv_sets
End

Algorithm 3 Subroutine Inv_sk_2

Require:
- Block length n
- Permutation Π

Ensure: List of subsets that are unions of orbits under Π and satisfy (1.)–(4.) of Theorem 2.

Begin
$orbits := []$
$result := []$
$work_set := \{0, 1, \ldots, n-1\}$
while $work_set \neq \oslash$ **do**
  Take $a \in work_set$
  $orbit_a :=$the orbit of a w.r.t Π as a subset of $\mathbb{Z}_n$
  Add $orbit_a$ to $orbits$
  $work_set := work_set \setminus orbit_a$
end while
for $candidate \in 2^{orbits}$ **do**
  Concatenate elements in $candidate$ to form a subset of indexes $=: c$
  if $check_\bar{1}_property(c)$ and $c \cap \{n-6, \ldots, n-1\} = \oslash$ **then**
    Append c to $result$
  end if
end for
return $result$
End

3.2 Invariant projected sets via polytopes in $\mathbb{Z}^n$

In this subsection we provide an alternative method of finding invariant projected subsets. The method is based on providing a one-to-one correspondence between defining sets of the invariant projective subsets of PRINTcipher–n and points of a certain polytope in $\mathbb{Z}^n$. One can then efficiently find these points by applying techniques of Mixed Integer Linear Programming (MILP).

Theorem 3. *Let IP_n be a subset in $\{0,1\}^n \subset \mathbb{Z}^n, 3|n$, defined as a subset of those $x = (x_0, \ldots, x_{n-1}) \in \{0,1\}^n$ that satisfy*[2]

$$\begin{aligned} x_{n-6} = \cdots = x_{n-1} &= 0, \\ \text{for all } 0 \le j < n/3 &: \\ x_{3j} + x_{3j+1} + x_{3j+2} &= x_{P^{-1}(3j)} + x_{P^{-1}(3j+1)} + x_{P^{-1}(3j+2)} \\ x_{3j} + x_{3j+1} &\ge x_{3j+2} \\ x_{3j} + x_{3j+2} &\ge x_{3j+1} \\ x_{3j+1} + x_{3j+2} &\ge x_{3j}. \end{aligned} \tag{2}$$

For each $v \in IP_n$ define $T_v := \{i | v_i = 1\} \subset \mathbb{Z}_n$. Then for every $v \in IP_n$ the set T_v satisfies:

1. $T_v \cap \{n-6, \ldots, n-1\} = \oslash$;
2. T_v *is a* $\bar{1}-$*subset of* $\mathbb{Z}_n$.
3. $P(T_v)$ *is a* $\bar{1}-$*subset of* $\mathbb{Z}_n$.
4. $\forall\ 0 \le j < n/3 : |\{3j, 3j+1, 3j+2\} \cap T_v| = |\{3j, 3j+1, 3j+2\} \cap P(T_v)|$.

Proof. For the proof we need the following lemma

Lemma 1. *For a pair of vectors $a = (a_0, a_1, a_2)$ and $b = (b_0, b_1, b_2)$ from $\mathbb{Z}^3 \cap \{0,1\}^3$ that satisfy*

$$b_0 + b_1 + b_2 = a_0 + a_1 + a_2, \quad b_0 + b_1 \ge b_2, \quad b_0 + b_2 \ge b_1, \quad b_1 + b_1 \ge b_0,$$

only the following cases for Hamming weights of a and b are possible: $wt(a) = wt(b) = i$ for $i = 0, 2, 3$.

Proof. The claim follows by direct inspection.

Now property (1.) follows directly from the condition in the first line of (2). It is not hard to see that the above Lemma easily yields claimed properties (2.)–(4.).

Now combining Theorem 2 and Theorem 3 we obtain the one-to-one correspondence as in Figure 3.

So now the question is how to compute all elements of IP_n for $n = 48, 96$. One possible solution is to use the MILP. Recall that the problem of the MILP is to optimize (i.e. either maximize or minimize) a linear objective function under linear constraints with solutions lying in $\mathbb{Z}^p \times \mathbb{R}^{n-p}$. In our case the feasible solutions are all in $\{0,1\}^n$ as per (2) and we thus have a binary integer program – a certain kind of MILP. Therefore, we can use an MILP solver with the additional requirement that the solutions should lie in $\{0,1\}^n$. Also, in our

[2] Note that the arithmetic is integer.

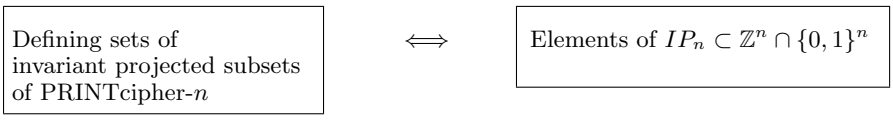


Fig. 3. One-to-one correspondence between defining sets of the invariant projected subsets and elements of IP_n

case we do not actually have an optimization problem, but rather an enumeration problem. However, we can utilize the MILP by using an objective function that is constant on the subset defined by (2) and then solve this "optimization" problem with the constraints defined by (2) for all "optimal" solutions, i.e. find IP_n as the feasible set for this optimization problem. In our specific case, we may set as an objective function $F(x_0, \ldots, x_{n-1}) = x_{n-1}$, so that $F(x) = 0\ \forall x \in IP_n$[3]. See the next subsection for specific results of applying this technique.

3.3 Obtaining the weak key classes

In this subsection we provide detailed results for PRINTcipher–48 that can be obtained by the methods from Sections 3.1 and 3.2. We also provide brief overview of the results for PRINTcipher–96. To implement the search algorithms from Section 3.1 we used SAGE computer algebra system [13]. For the method from Section 3.2 we use the MILP solver CPLEX[4] through the SAGE interface.

Table 4 presents the results. Overall, for PRINTcipher–48 we have 64 different defining sets of invariant projected subsets, which give rise to 64 classes of weak keys. The ones that were presented in [12] are marked in bold. In the course of this subsection we will explain how we obtained the numbers in the last two columns of Table 4: the number of weak keys in each class and the corresponding work factor of the key recovery.

How to distinguish the weak keys? Now let us detail how to distinguish the weak keys belonging to a class defined by a certain $def(U)$ in the chosen plaintext scenario. Consider the set of chosen plaintexts

$$P_{def(U)} = \{p \in \{0,1\}^n | p_i = val(U)_{ind_U(i)}\ \forall i \in U_2\},$$

where U_2 is defined in Definition 4. We know the values $\{val(U)_{ind_U(i)} : i \in U_2\}$ as Theorem 2 suggests. Now the attacker chooses an arbitrary $p \in P_{def(U)}$ and encrypts p with a key $k \in \{0,1\}^l$ obtaining $c = E_k(p)$. If $c_i = p_i$ for all $i \in def(U)$, then k belongs to the class of weak keys defined by $def(U)$[5]. Note that we only prescribe positions belonging to U_2 to define $P_{def(U)}$, but when considering the ciphertext, we take the entire $def(U)$ into consideration, see also the proof of Theorem 2.

[3] This is a purely technical trick, since in theory we could take any constant function as an objective function. The way CPLEX solver works, however, we could not simply use a constant function, but rather a function that is constant on the points of IP_n

[4] IBM ILOG CPLEX 12.1 under the academic license.

[5] Note that we actually need several chosen plaintexts, since with probability $2^{-|def(U)|}$ one has $p_i = c_i, i \in def(U)$ where the key k can be any key. So in order to distinguish we need $l/|def(U)|$ pairs. For $l = 80$ it is at most 5 pairs, see also [12].

Table 4. Defining sets of invariant projected subsets of PRINTcipher–48

No.	$def(U)$	$\#k$	$\log WF_{key_recovery}$
1	[0, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 22, 23, 24, 25, 27, 28, 29, 33, 34, 36, 37, 39, 41]	45	31
2	[0, 1, 3, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 33, 34, 36, 37, 39, 41]	42	30
3	[0, 2, 3, 5, 6, 8, 9, 11, 12, 13, 16, 17, 19, 20, 22, 23, 24, 25, 27, 28, 29, 33, 34, 36, 37, 39, 41]	37	30
4	[0, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 22, 24, 25, 27, 28, 29, 33, 34, 36, 37, 39, 41]	41	30
5	[0, 2, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 33, 35, 36, 37, 39, 41]	37	29
6	[0, 2, 3, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 20, 22, 23, 24, 25, 27, 28, 29, 33, 35, 36, 37, 39, 41]	36	30
7	[0, 1, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 37, 39, 41]	37	29
8	[0, 1, 3, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 20, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 37, 39, 41]	37	30
9	[0, 2, 3, 5, 6, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 33, 34, 36, 38, 39, 41]	46	32
10	[0, 2, 3, 5, 7, 8, 9, 11, 12, 13, 16, 17, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 33, 34, 36, 38, 39, 41]	39	30
11	[0, 1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 22, 23, 24, 25, 27, 28, 29, 33, 34, 36, 38, 39, 41]	44	31
12	[0, 2, 3, 5, 6, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 20, 21, 22, 24, 25, 27, 28, 29, 33, 34, 36, 38, 39, 41]	42	31
13	[0, 2, 3, 5, 7, 8, 9, 11, 12, 13, 16, 17, 19, 20, 21, 22, 24, 25, 27, 28, 29, 33, 34, 36, 38, 39, 41]	35	29
14	[0, 1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 22, 24, 25, 27, 28, 29, 33, 34, 36, 38, 39, 41]	40	30
15	[0, 1, 3, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 23, 24, 25, 27, 28, 29, 33, 34, 36, 38, 39, 41]	39	30
16	[0, 2, 3, 5, 6, 8, 9, 11, 12, 13, 16, 17, 19, 20, 21, 23, 24, 25, 27, 28, 29, 33, 34, 36, 38, 39, 41]	35	30
17	[0, 2, 3, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 22, 23, 24, 25, 27, 28, 29, 33, 35, 36, 38, 39, 41]	38	31
18	[0, 2, 4, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 22, 23, 24, 25, 27, 28, 29, 33, 35, 36, 38, 39, 41]	38	30
19	[0, 2, 3, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 22, 24, 25, 27, 28, 29, 33, 35, 36, 38, 39, 41]	34	30
20	[0, 2, 4, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 22, 24, 25, 27, 28, 29, 33, 35, 36, 38, 39, 41]	34	29
21	[0, 2, 3, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 29, 33, 35, 36, 38, 39, 41]	35	31
22	[0, 2, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 23, 24, 25, 27, 28, 29, 33, 35, 36, 38, 39, 41]	34	29
23	[6, 8, 9, 11, 12, 13, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 39, 41]	50	42
24	[0, 1, 3, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 39, 41]	40	31
25	[0, 1, 4, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 39, 41]	39	30
26	[0, 1, 3, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 22, 24, 25, 27, 28, 29, 34, 35, 36, 38, 39, 41]	36	30
27	[0, 1, 4, 5, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 22, 24, 25, 27, 28, 29, 34, 35, 36, 38, 39, 41]	35	29
28	[0, 1, 3, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 39, 41]	36	31
29	[0, 1, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 39, 41]	34	29
30	[0, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 33, 34, 37, 38, 39, 41]	48	32
31	[0, 2, 3, 4, 7, 8, 9, 11, 12, 13, 16, 17, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 33, 34, 37, 38, 39, 41]	41	30
32	[0, 1, 3, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 33, 34, 37, 38, 39, 41]	45	31
33	[0, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 20, 21, 22, 24, 25, 27, 28, 29, 33, 34, 37, 38, 39, 41]	44	31
34	[0, 2, 3, 4, 7, 8, 9, 11, 12, 13, 16, 17, 19, 20, 21, 22, 24, 25, 27, 28, 29, 33, 34, 37, 38, 39, 41]	37	29
35	[0, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 19, 20, 21, 23, 24, 25, 27, 28, 29, 33, 34, 37, 38, 39, 41]	37	30
36	[0, 2, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 33, 35, 37, 38, 39, 41]	40	30
37	[0, 2, 3, 4, 7, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 22, 23, 24, 25, 27, 28, 29, 33, 35, 37, 38, 39, 41]	40	31
38	[0, 2, 3, 4, 7, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 22, 24, 25, 27, 28, 29, 33, 35, 37, 38, 39, 41]	36	30
39	[0, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 29, 33, 35, 37, 38, 39, 41]	37	31
40	[0, 1, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 34, 35, 37, 38, 39, 41]	40	30
41	[0, 1, 3, 4, 7, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 22, 23, 24, 25, 27, 28, 29, 34, 35, 37, 38, 39, 41]	43	31
42	[0, 1, 3, 4, 7, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 22, 24, 25, 27, 28, 29, 34, 35, 37, 38, 39, 41]	39	30
43	[0, 1, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 29, 34, 35, 37, 38, 39, 41]	39	31
44	**[0, 1, 4, 5, 12, 13, 16, 17, 24, 25, 28, 29, 36, 37, 40, 41]**	**51**	**48**
45	[0, 1, 3, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 33, 34, 36, 37, 40, 41]	40	30
46	[0, 2, 3, 5, 6, 7, 9, 11, 12, 13, 16, 17, 19, 20, 22, 23, 24, 25, 27, 28, 29, 33, 34, 36, 37, 40, 41]	34	29
47	**[0, 1, 3, 4, 5, 9, 11, 12, 13, 16, 17, 24, 25, 27, 28, 29, 33, 35, 36, 37, 40, 41]**	**51**	**43**
48	[0, 2, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 33, 35, 36, 37, 40, 41]	36	29
49	[0, 2, 3, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 20, 22, 23, 24, 25, 27, 28, 29, 33, 35, 36, 37, 40, 41]	35	30
50	[0, 1, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 37, 40, 41]	35	29
51	[0, 1, 3, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 20, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 37, 40, 41]	35	30
52	[0, 1, 3, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 21, 23, 24, 25, 27, 28, 29, 33, 34, 36, 38, 40, 41]	38	30
53	[0, 2, 3, 5, 6, 7, 9, 11, 12, 13, 16, 17, 19, 20, 21, 23, 24, 25, 27, 28, 29, 33, 34, 36, 38, 40, 41]	33	29
54	[0, 2, 3, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 29, 33, 35, 36, 38, 40, 41]	35	31
55	[0, 2, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 21, 23, 24, 25, 27, 28, 29, 33, 35, 36, 38, 40, 41]	34	29
56	[0, 1, 3, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 40, 41]	35	31
57	[0, 1, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 21, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 40, 41]	33	29
58	[6, 7, 9, 11, 12, 13, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 40, 41]	48	42
59	[0, 2, 3, 4, 6, 7, 9, 11, 12, 13, 16, 17, 19, 20, 21, 23, 24, 25, 27, 28, 29, 33, 34, 37, 38, 40, 41]	35	29
60	[0, 1, 3, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 33, 34, 37, 38, 40, 41]	43	31
61	[0, 2, 3, 4, 6, 7, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 29, 33, 35, 37, 38, 40, 41]	37	31
62	[0, 2, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 33, 35, 37, 38, 40, 41]	39	30
63	[0, 1, 3, 4, 6, 7, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 29, 34, 35, 37, 38, 40, 41]	38	31
64	[0, 1, 4, 5, 6, 7, 9, 11, 12, 13, 16, 17, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 34, 35, 37, 38, 40, 41]	38	30

How many elements are there in each class? Next we are interested in the question of computing the number of weak keys in each of the 64 classes from Table 4. The number of weak keys can be obtained by observing that each key is of the form (sk_1, sk_2). Now denote by $\#sk_i, i = 1, 2$ the logarithm of the size of the sk_i-part of the weak keys defined by some $def(U)$.
First let us compute $\#sk_2$. Using Table 3 we can provide a procedure for computing $\#sk_2$, see Algorithm 4. The algorithm goes through the position triplets and explores which triplets have two elements of $def(U)$ in them (these are elements of U_2). For these we have the 2–2 case for the corresponding S-Box (a 1-mask). Active positions correspond to "+" in the output mask and inactive positions to "−". The input mask is formed analogously, observing that bits of $P(def(U))$ are active in the input to the keyed S-Box layer $\pi_{sk_2} \circ S$. Now we add the corresponding values from the column 4 of Table 3, which are denoted by $Table_masks[in, out]$. In case we have a 3–3 or a 0–0 situation we know that we may assign sk_2 bits arbitrarily for that S-Box, i.e. we have +2 in the count of the sk_2-part of the weak keys number. As a result, we have $\#sk_2 = Number_sk2(n, def(U))$. where $Number_sk2(n, def(U))$ is the output of Algorithm 4.
Once the values of the sk_2-part of the key are assigned as above, the 2–2 S-Boxes fix values for $X_i, i \in U_2$ and $Y_j, j \in P^{-1}(P(def(U))_2)$, where $Y = XOR_{sk_1}(X)$. Note that $|P^{-1}(P(def(U))_2)| = |U_2|$. Now we may assign arbitrary values to the outputs of the 3–3 S-Boxes. So $X_i, i \in U_3$ attain all possible values. Once outputs of the 3–3 S-Boxes are fixed, so are the inputs. Therewith $Y_j, j \in P^{-1}(P(def(U))_3)$ get specific values that depend uniquely on the values of the sk_2-bits and outputs of the 3–3 S-Boxes. Now X_i and Y_i have their values for $i \in def(U)$ and there are $2^{|U_3|}$ ways to assign these values. Considering that other $sk_{1,j}, j \in \mathbb{Z}_n \setminus def(U)$ may obviously attain arbitrary values, we obtain $\#sk_1 = |U_3| + n - |def(U)| = n - |U_2|$. Summing up, we have that the total number of weak keys in the class defined by $def(U)$ is

$$2^{\#k} = 2^{\#sk_1 + \#sk_2} = 2^{n - |U_2| + Number_sk2(n, def(U))}. \tag{3}$$

Remark 2. Interestingly enough, the classes of weak keys in Table 4 may very well intersect. For example, the two classes no. 44 and no. 47 from [12] have a non-trivial intersection. It has 2^{45} elements. Therewith, the actual number of weak keys found in [12] is $2^{52} - 2^{45} \approx 2^{51.989}$.

Due to Remark 2 we cannot claim that the overall number of weak keys is simply the sum of numbers in the third column of Table 4, which is $2^{52.51}$; it provides an upper bound, however. As we can see, this upper bound does not differ significantly from the number of keys found in [12], since Leander *et al.* recovered the two largest classes out of possible 64.

What is the complexity of the weak key recovery? Once a weak key is *distinguished*, the attacker wants to *recover* the remaining key bits that do not follow immediately from the distinguishing phase.

Remark 3. Note that "*" 's in the 3rd column of two rows in Table 3 mean that *any* value will work, e.g. both S-Boxes with $(a_0, a_1) = (0, 0)$ and $= (0, 1)$ satisfy the mask $++- \to ++-$ independently of the value of the unknown bits in the mask. This, in particular, means that in the key recovery phase we can assign any value to these bits and thus these bits do not contribute to the work factor.

Algorithm 4 $Number_sk_2$

Require:
- Block length n of PRINTcipher-n
- $def(U)$ for some U an invariant projected subset

Ensure: $\#sk_2$ for $def(U)$

```
Begin
result := 0
for 0 ≤ j < n/3 do
  if |{3j, 3j + 1, 3j + 2} ∩ def(U)| = 2 then
    Form +/- mask for out = |{3j, 3j + 1, 3j + 2} ∩ def(U)|
    Form +/- mask for in = |{3j, 3j + 1, 3j + 2} ∩ P(def(U))|
    result := result + Table_masks[in, out]
  else
    result := result + 2
  end if
end for
return result
End
```

Due to Remark 3 the actual size of the search space is not the number of weak keys, which is computed via (3), but is smaller due to the fact that sometimes the value of the second sk_2-bit is irrelevant in the chosen plaintext scenario. Let $*(def(U))$ be the number of S-Boxes such that there is a "*" in Table 3 w.r.t. a given $def(U)$. The work factor of recovering sk_2 is thus reduced by $2^{*(def(U))}$. Moreover, as has been noted above, knowing 3–3 S-Boxes (ones the corresponding sk_2-bits are fixed) one obtains inputs to these S-Boxes automatically, since in the recovery phase the ciphertext c is given and so are all output bits to the 3–3 S-Boxes. So we simply remove the factor $2^{|U_3|}$ from the above computation of $\#sk_1$ and obtain that the work factor of the key recovery is

$$WF_{key_recovery} = 2^{n-|def(U)|+Number_sk2(n,def(U))-*(def(U))}. \tag{4}$$

In order to recover the "*"-bits of sk_2 we take another pair plaintext/ciphertext, where a plaintext is now arbitrary, and recover the remaining bits with work factor $2^{*(def(U))}$. So the total work factor is the sum of $2^{*(def(U))}$ and (4). Since the former term is dominated by the latter, it can be neglected. Column 4 of Table 4 presents numbers obtained using (4).

Example 1. In this example we would like to provide a detailed workout of the computations discussed in this subsection. Moreover, this example should help to understand better the proof of Theorem 1.
For this example let us take class no. 5 from Table 4. So

$$def(U) = \{0, 2, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 33, 35, 36, 37, 39, 41\}.$$

In the table below, we provide input/output masks, input/output values, as well as the corresponding values that are obtained during the run of Algorithm 4 and the value $*(def(U))$ for each S-Box; the corresponding values of key bits in sk_2 are given as well:

S-Box	in_mask	out_mask	num_sk2	star	(a_0, a_1)	in_vals	out_vals
0	$++-$	$+-+$	0	0	(1,0)	10*	1*1
1	$-++$	$-++$	1	1	($\forall$,0)	*11	*10
2	$++-$	$+-+$	0	0	(1,0)	10*	1*1
3	$-++$	$+-+$	0	0	(0,1)	*01	1*1
4	$+-+$	$++-$	0	0	(1,0)	0*0	00*
5	$+-+$	$-++$	0	0	(0,1)	1*1	*10
6	$++-$	$++-$	1	1	(0,$\forall$)	00*	00*
7	$-++$	$-++$	1	1	($\forall$,0)	*11	*10
8	$++-$	$++-$	1	1	(0,*)	00*	00*
9	$+++$	$+++$	2	0	(E,F)	ABC	XYZ
10	$---$	$---$	2	0	(*,*)	***	***
11	$++-$	$+-+$	0	0	(1,0)	10*	1*1
12	$++-$	$++-$	1	1	(0,$\forall$)	00*	00*
13	$++-$	$+-+$	0	0	(1,0)	10*	1*1
14	$---$	$---$	2	0	(*,*)	***	***
15	$---$	$---$	2	0	(*,*)	***	***
Sum			13	5			

In this table the symbol $\forall$ means that any value of the corresponding sk_2-bit works; "ABC" and "XYZ" mean that we obtain these values in the actual attack, moreover it holds that $S(ABC) = XYZ$. Note that "XYZ" itself can attain arbitrary values and "ABC" is determined by these. The values we have no control over are denoted with "*" and can be arbitrary.
Now let us apply formula (3). We have that $\#k = \#sk_1 + \#sk_2 = n - |U_2| + Number_sk2(n, def(U))$. Then $n = 48, |U_2| = 24$ and $Number_sk2(n, def(U)) = 13$ as can be seen from the table above. So we have $\#k = 48 - 24 + 13 = 37$, see Table 4. Next, formula (4) says that $\log WF_{key_recovery} = n - |def(U)| + Number_sk2(n, def(U)) - *(def(U))$. We have that $|def(U)| = 27$ and $*(def(U)) = 5$ as the table above indicates. As a result we obtain $\log WF_{key_recovery} = 48 - 27 + 13 - 5 = 29$, compare with Table 4.
Next, let us see how the values from the table above are distributed in one round:

$in =$	1*1	*10	1*1	1*1	00*	*10	00*	*10	00*	XYZ	***	1*1	00*	1*1	***	***
$sk_1 =$	0*0	*11	1*1	K*0	01*	*11	00*	*11	0B*	XYZ	***	0*0	01*	0*L	***	***
$XOR =$	1*1	*01	0*0	A*1	01*	*01	00*	*01	0B*	000	***	1*1	01*	1*C	***	***
$P =$	10*	*11	10*	*01	0*0	1*1	00*	*11	00*	ABC	***	10*	00*	10*	***	***
$sk_2 =$	10	$\forall$0	10	01	10	01	0$\forall$	$\forall$0	0$\forall$	EF	**	10	0$\forall$	10	**	**
$out =$	1*1	*10	1*1	1*1	00*	*10	00*	*10	00*	XYZ	***	1*1	00*	1*1	***	***

In the table the values in *out* and *in* are the same and are taken from the *out_vals* column of the first table. Then, the values in P correspond to the inputs to S-Boxes (or, equivalently, to the outputs of the layer P) and are taken from the column *in_vals* of the first table. We then permute the values in P with P^{-1} to get the output of the XOR_{sk_1} operation. Having that, we may compute many values of sk_1 right away. Note that $K = 1 + A, L = 1 + C$ and are determined by the values of "XYZ".
For a specific example, let us take $XYZ = 000$ and $EF = 00$. We have then that $ABC = 000, K = L = 1$. Therewith we get

$$U = 1*1\ *10\ 1*1\ 1*1\ 00*\ *10\ 00*\ *10\ 00*\ 000\ ***\ 1*1\ 00*\ 1*1\ ***\ ***,$$

such that $def(U)$ is exactly the one in question and

$$val(U) = 1,1,1,0,1,1,1,1,0,0,1,0,0,0,1,0,0,0,0,0,0,1,1,0,0,1,1.$$

The corresponding weak keys are of the form (sk_1, sk_2), where

$$sk_1 = 0*0\ \ *11\ 1*1\ 1*0\ 01*\ \ *11\ 00*\ \ *11\ 00*\ \ 000\ \ ***\ \ 0*0\ 01*\ \ 0*1\ \ ***\ \ ***,$$

$$sk_2 = 10\ \ \ *0\ \ 10\ \ 01\ \ 10\ \ 01\ \ 0*\ \ \ *0\ \ 0*\ \ \ 00\ \ \ **\ \ 10\ \ 0*\ \ \ 10\ \ \ **\ \ \ **$$

It is not hard to see that the set of weak keys above is different from the ones presented in [12]. Indeed, for example the keys with

$$sk_1 = 0\mathbf{0}0\ \ *11\ 1*1\ 1*0\ 01*\ \ *11\ 00*\ \ *11\ 00*\ \ 000\ \ ***\ \ 0*0\ 01*\ \ 0*1\ \ ***\ \ ***$$

do not belong to the class defined by $def(U)$ no. 44, since there $sk_{1,1} = 1$ and the keys with

$$sk_1 = 0*0\ \ *11\ 1*1\ 1*0\ 01*\ \ *11\ 00*\ \ *11\ 00*\ \ 000\ \ ***\ \ 0*0\ 01*\ \ 0\mathbf{0}1\ \ ***\ \ ***$$

do not belong to the class defined by $def(U)$ no. 47, since there $sk_{1,40} = 1$, see [12].

Protecting against the attack Note that due to existence of the round counter RC_i in the last 6 bits (corresponding to the S-Boxes 14 and 15), our invariant projected subsets are not active in the last two S-Boxes. As can be seen from Table 4 there exist no $def(U)$ that avoids the last *three* S-Boxes. So, as has already been pointed out in [12], spreading out the round counter over the last three S-Boxes (two bits of the counter per S-Box) protects against the attack. Note, however, that this choice is not as obvious as it may seem. For example, a SPONGENT-like solution [6], where S-Boxes 0,1 and 14,15 are used for the round counter (or any three of them) does not provide a secure solution, since classes no. 23 and 58 avoid them, providing at least 2^{50} weak keys. Still, the "SPONGENT" solution obviously defeats the classes no. 44 and 47 of [12].

Results for PRINTcipher–**96** In order to get all invariant projected subsets for PRINTcipher–96, the method of Section 3.2 has to be used, since the one of Section 3.1 is not feasible any more (see the end of that subsection). We do not provide detailed results for PRINTcipher–96 here. We just mention that PRINTcipher–96 has 115,669 classes of weak keys. We could obtain these after 10 minutes of computations using CPLEX for solving the MILP problem as per Section 3.2.
Note that in the case of PRINTcipher–96 there *exist* invariant projected subsets that avoid the last three S-Boxes. In fact, there are 28 such classes. So the countermeasure suggested in [12] does not work. Still, there is a collection of combinations of three S-Boxes that cannot be avoided by any invariant projected subset. The S-Boxes 0,1,23 is one possible solution among many others. So, in order to counter the attack, one has to spread out the round counter over these S-Boxes. These results are obtained by replacing the condition on the last two S-Boxes in the first line of (2) with the 0-conditions on the variables x_i that correspond to a choice of three S-Boxes in question.

3.4 Fast key recovery for many weak keys of PRINTcipher–48

As we may see from Table 4, column 4, the work factor for the key recovery of many classes is very low. Therewith, weak key recovery for these classes may be done very fast in the chosen plaintext scenario. We can estimate practical time for the key recovery using figures for a software implementation of PRINTcipher–48 reported in [4]. There the authors claim 72–95 CPU cycles per byte for PRINTcipher–48. For our estimates we take 95 cycles per byte as a pessimistic figure. Now let us assume doing key recovery using a 64-bit processor with 2.4 GHz frequency, e.g. AMD Athlon 64 4000+, whereby only one core is doing the computation. We may estimate that 1 CPU cycle takes $1/(2.4)\cdot 10^{-9})$ seconds. In our estimates we only compute the time needed to do the necessary number of trial encryptions, not counting overhead[6] that appears in actually implementing the brute force attack. Still, it is quite clear that such an overhead is negligible compared to the time needed for the trial encryptions. So now we have

$$T_{key_recovery} = WF_{key_recovery} \cdot 95 \cdot \frac{1}{2.4 \cdot 10^9} \cdot 6 \text{ sec},$$

where the "6" is the number of bytes in 48 bits. Table 5 shows the results obtained using the above formula. The table lists the time to do the brute force search assuming the worst case where an attacker has to look through the entire search space. In the table we also indicate the number of different $def(U)$'s, i.e. classes of weak keys for which corresponding timings for the key recovery are obtained, as well as the lower and upper bounds on the number of weak keys with the corresponding time recovery. For the lower bound we took the number of keys in the largest class and as an upper bound the sum of keys in all classes with the same timing. As we may see the bounds are not far from each other, so we have a pretty good idea about the actual number of weak keys in each case.

Table 5. Expected timings for weak key recovery for PRINTcipher–48

$\log(T_{key_recovery},$ sec)	# classes	$\log(\#_{lower})$	$\log(\#_{upper})$
7	15	37	39.4
8	25	42	44
9	18	45	46.9
10	2	48	48.3
20	2	50	50.3
21	1	51	51
26	1	51	51

As a result of Table 5 we see that at least 2^{48} weak keys can be recovered in less than 20 minutes and at least 2^{37} weak keys can be recovered in about 2 minutes or less using just one core of a "usual" PC. Note that recovering keys from the class no. 44, which is the main example in [12], would take around 2 years and the one from Appendix A in [12] around 1 month. Therewith, along with known classes we showed existence of classes that allow very fast key recovery.

[6] E.g. computing the bits $sk_{1,i}, i \in U_3$ given p, c, and an sk_2 candidate.

4 Related and future work

Finally, in this section we would like to briefly discuss the initial motivation of this work and show its relation to the material of the paper.
Initially we were interested in the question of how to maximize the number of internal bits known when describing the encryption function algebraically given a plaintext and a part of a key. That is, having a plaintext and knowing (or guessing) certain key bits, which (and how many) internal bits can we learn from that.
Let us consider a PRESENT-like primitive interleaving a permutation layer P and a substitution layer, in this order. Note that constant additions can be ignored here, since constants are publicly known and do not constrain information flow. Observe that the value of a bit of the state is known if it is either known initially (or guessed) or if it is learned through propagation from known values in the previous round. Our goal is to maximize the number of known values using a Mixed Integer (or, strictly speaking, Binary Integer) Linear Program. For this we can introduce two variables for each bit i in each round j: $x_{i,j} = 1$ iff the j-th bit is known after round i, and $g_{i,j} = 1$ iff it is initially known (or guessed). The objective function is simply the sum of all $x_{i,j}$. Restricting the number of the initially known bits to a certain threshold is similarly straightforward and can be done by introducing a constraint requiring the sum of all $g_{i,j}$ to be smaller than this threshold. The non-trivial part of the model is including the propagation and relating x- and g-variables. For this, let us assume a very strong S-Box, where the outputs of an S-Box are only known if the full input is known (for PRINTcipher that would be the 3 – 3 case). So, let $\{x_{i,P^{-1}(j_0)}, x_{i,P^{-1}(j_1)}, \cdots, x_{i,P^{-1}(j_{l-1})}\}$ be the set of variables of the input bits of a certain S-Box in round i, where l is the block length of an S-Box. Then $\{x_{i+1,j_0}, x_{i+1,j_1}, \cdots, x_{i+1,j_{l-1}}\}$ is the set of variables of the output bits of that S-Box. The model can now be completed by introducing the following set of constraints for every S-Box:

$$x_{i+1,j_t} \leq x_{i,P^{-1}(j_s)} + g_{i+1,j_t} \quad \text{for all } t, s \in \{0, 1, \cdots, l-1\}.$$

If other masks are to be considered, these constraints need to be adjusted accordingly. One approach to do this is to consider each possible mask as a vertex of a high dimensional 0-1 polyhedron (the dimension depending on l, e.g. for PRINTcipher this would be $2 \cdot l = 2 \cdot 3 = 6$). The polyhedron so constructed can be converted to its half-space representation, which can then be used to model the information flow.
Applying this approach to PRINTcipher (incorporating constraints accommodating the key addition), surprisingly or may be not really, ended up producing invariant projected subsets as results, in particular also those that were known from [12]. Refinement of the approach led to the results of Section 3.2 and to the overall study undertaken in Section 3. We would like to mention the paper [14] where techniques of MILP were used to facilitate differential and linear cryptanalysis.
As a result, the methods presented in this paper, especially the ones of Section 3.2, should be seen in a broader context of obtaining desirable cryptanalytic properties. The method turned out to be extremely useful in the question of the complete study of the invariant coset attack from [12]. Still, the optimization method outlined here may be used with its initial purpose in mind. Note that having more bits known internally in the course of the encryption process facilitates other cryptanalytic methods, such as algebraic cryptanalysis [15,16]. Basically, one has more variables resolved "for free" in an algebraic system describing a primitive in question. Especially PRESENT-like ciphers seem to be good targets for this method: PRESENT itself

[1], SPONGENT hash family [6], EPCBC block cipher [5]. SP-network LBlock [17] also has all properties in its description to apply the technique. It would be interesting to see how the proposed optimization technique works in this context. This is a research in progress and is an interesting research direction combining techniques like algebraic system solving (Gröbner bases, SAT-solvers) and optimization techniques (e.g. MILP).

Acknowledgements

The first author is supported by the German Science Foundation (DFG) grant BU 630/22-1.

References

1. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe, "PRESENT – An Ultra-Lightweight Block Cipher". In P. Pailier, I. Verbauwhede (Eds.) CHES 2007, LNCS 4727, pp.450–466.
2. C. de Canniére, O. Dunkelman, M. Knezević, "KATAN and KTANTAN A Family of Small and Efficient Hardware-Oriented Block Ciphers". In C. Clavier, K. Gaj (Eds.) CHES 2009, LNCS 5747, pp. 272-288.
3. J. Guo, T. Peyrin, A. Poschmann, M. Robshaw, "The LED Block Cipher". In B. Preneel, T. Takagi (Eds.) CHES 2011, LNCS 6917, pp. 326–341.
4. L. Knudsen, G. Leander, A. Poschmann, M.J.B. Robshaw, "PRINTcipher: A Block Cipher for IC-Printing". In S. Mangard and F.-X. Standaert (Eds.) CHES 2010, LNCS 6225, pp.16–32, 2010.
5. H. Yap, K. Khoo, A. Poschmann, M. Henricksen, "EPCBC – A Block Cipher Suitable for Electronic Product Code Encryption". In D. Lin, G. Tsudik, X. Wang (Eds.) CANS 2011, LNCS 7092, pp. 76–97.
6. A. Bogdanov, M. Knezević, G. Leander, D. Toz, K. Varici, I. Verbauwhede, "SPONGENT: A Lightweight Hash Function". In B. Preneel, T. Takagi (Eds.) CHES 2011, LNCS 6917, pp. 312–325.
7. M.A. Abdelraheem, G. Leander, E. Zenner, "Differential cryptanalysis of round-reduced PRINTcipher: Computing roots of permutations". In A. Joux (Ed.) FSE 2011, LNCS 6733, pp. 1–17.
8. M, Agren, T. Johansson, "Linear Cryptanalysis of PRINTcipher - Trails and Samples Everywhere". In D.J. Bernstein, S. Chatterjee (Eds.) INDOCRYPT 2011, LNCS 7107, pp. 114–133.
9. F. Karakoc, H. Demirci, A. Emre Harmanci, "Combined Differential and Linear Cryptanalysis of Reduced-Round PRINTcipher". In A. Miri, S. Vaudenay (Eds.) SAC 2011, LNCS 7118.
10. S. Bulygin, J. Buchmann, "Algebraic Cryptanalysis of the Round-Reduced and Side Channel Analysis of the Full PRINTcipher–48". In D. Lin, G. Tsudik, X. Wang (Eds.) CANS 2011, LNCS 7092, pp. 54–75.
11. X. Zhao, T. Wang, S. Guo, "Fault Propagate Pattern Based DFA on SPN Structure Block Ciphers using Bitwise Permutation, with Application to PRESENT and PRINTcipher", ePrint, available at `http://eprint.iacr.org/2011/086.pdf`
12. G. Leander, M.A. Abdelraheem, H. AlKhzaimi, E.Zenner, "A Cryptanalysis of PRINTcipher: The Invariant Coset Attack". In P. Rogaway (Ed.) CRYPTO 2011, LNCS 6841, pp. 206–221.
13. S. William Stein et al., "SAGE Mathematics Software". The Sage Development Team, 2008. Available at `http://www.sagemath.org` 593–599.
14. N. Mouha, Q. Wang, D. Gu, B. Preneel, "Differential and Linear Cryptanalysis using Mixed-Integer Linear Programming," Inscrypt 2011, Lecture Notes in Computer Science, Springer, 2011.
15. C. Cid, S. Murphy, M.J.B. Robshaw, "Algebraic Aspects of the Advanced Encryption Standard", Springer-Verlag, 2006.
16. G.V. Bard,"Algebraic Cryptanalysis", Springer, 2009.
17. W. Wu, L. Zhang, "LBlock: A Lightweight Block Cipher". In J. Lopez, G. Tsudik (Eds.) ACNS 2011, LNCS 6715, pp. 327–344.