Many Weak Keys for PRINTcipher: Fast Key Recovery and Countermeasures

Stanislav Bulygin[1,2], Michael Walter[3], and Johannes Buchmann[1,2]

[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, buchmann@cdc.informatik.tu-darmstadt.de

[3] University of California, San Diego, Department of Computer Science and Engineering
9500 Gilman Drive, Mail code 0404, La Jolla, CA 92093-5004, USA
miwalter@eng.ucsd.edu

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 complete study and show that there exist 64 families of weak keys for PRINTcipher–48 and as many as 115,669 for PRINTcipher–96. Moreover, we show that searching the weak key space may be substantially sped up by splitting the search into two consecutive steps. We show that for many classes of weak keys, key recovery can be done with very small time complexity in the chosen/known plaintext scenario. This shows that the cipher is actually much more vulnerable to this type of attacks than was even thought previously. Still, effective countermeasures exist against the attack. The method of finding all weak key families has value on its own. It is based on Mixed Linear Integer Programming and can be adapted to solving other interesting problems on similar 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 such as RFID tags, smart cards, and sensor networks. 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 of secure usage of small devices. The block cipher PRESENT is one outstanding example that gained popularity [1]. Other block ciphers, such as

E. Dawson (Ed.): RSA 2013, LNCS 7779, pp. 189–206, 2013.
© Springer-Verlag Berlin Heidelberg 2013

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. 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 for a large portion of keys[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 significant portion of 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 for weak key families presented in [12] would take quite a considerable time in practice. 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. However, it remained unknown whether other families existed and what was a systematic way of finding them.

In this paper we thoroughly investigate the invariant property of PRINTcipher. We suggest a systematic method of obtaining all invariant cosets of the cipher and corresponding families of weak keys. In particular, we are able to recover *all* 64 families of weak keys for PRINTcipher–48 and *all* 115,669 families for PRINTcipher–96. It is possible to speed the key recovery process compared to a simple brute force of the remaining key bits. In the case of PRINTcipher–48 we identified many weak keys that can be recovered in a matter of minutes on a single PC. This shows that the cipher is actually much more vulnerable to this type of attacks than was even thought previously. The key recovery procedure is improved compared with the simple brute force by a factor of 8 to 2048 depending on the weak key class. For PRINTcipher–96 the gain can be as large as 2^{27}. Similarly to [12] we analyze countermeasures against the attack. Since our analysis of the invariant property is complete we may argue about security of the cipher against the invariant coset attack. In particular, *we claim that with countermeasures suggested in this paper both versions of PRINTcipher remain secure ciphers.*

The method of finding weak key families is based on Mixed Integer Linear Programming (MILP). It appears to be a novel technique to use MILP in such a context. Noteworthy is also that this method can be adapted to solving other interesting problems on similar ciphers and therefore has value on its own.

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

[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.

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 on defining sets of invariant projected subsets and their characterization. These defining sets then give rise to families of weak keys. Section 3.2 presents a method of finding all defining sets based on MILP that turns out to be highly efficient in practice. Section 3.3 summarizes the results for PRINTcipher–48 and PRINTcipher–96 obtained by our methods, computing the number of keys in each weak key family and complexity of the key recovery, as well as protecting measures. Finally, Section 4 provides a retrospective of the proposed methods and puts them in a more general context providing possible further directions for research. In Appendices A and B the reader may find the table with results for PRINTcipher–48 as well as a worked out example illustrating computations from Section 3.3 resp.

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 with PRESENT is the absence of a key schedule (all round keys are the same and are equal to a 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 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–48 follows a classical SP-network structure, see Figure 1 for the round function. Both versions have similar structure. The key $k = (sk_1, sk_2)$ is divided into two parts. The subkey sk_1 is used for XORing with the state and sk_2 defines the S-Box layer, see below. The linear diffusion layer is implemented by 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 RC_i for $i = 1, \ldots, n$ are 6-bit long round constants obtained via a certain LFSR and are placed in the last two position triplets. The S-Box layer 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 (triplets) 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

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

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)

three input bits are permuted according to the two consecutive bits from the subkey sk_2 called a_0 and a_1.

We need properties of masks that exist for keyed S-Boxes of PRINTcipher. By an α–β *mask* we understand a situation when fixing some α out of 3 S-Box input values yields fixed β output values regardless of the values at other input/output positions. There is obviously a 3–3 mask, since knowledge of three input bits of an S-Box yields all three output bits. There is also obviously a 0–0 mask. It turns out that for certain values of sk_2-bits, PRINTcipher's S-Box has 2–2 masks for all $3 \cdot 3 = 9$ possible combinations of input and output positions. Table 3 shows that each possible 2-mask on the input bits has a corresponding 2-mask for output bits. In this table $+/-$ notation shows which bits of the mask are fixed $(+)$ and ones which are 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 fixed, then both S-Boxes with $a_0 = 0$ satisfy the mask. Moreover, the two fixed 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]. Notably, there are also 2–1, 2–0, 1–0 masks and no others. Clearly, the 2–2 and 3–3 masks are the most desirable for cryptanalytic properties.

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

Table 3. Behavior of the 2–2 masks in S-Boxes of PRINTcipher

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

up in the same subset of plaintexts. 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. An invariant coset for the plaintexts is of the form $U + d$ for some linear subspace $U \subseteq \mathbb{F}_2^n$ 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 2–2 masks presented in Table 3.

3 Obtaining and Exploiting 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 investigate defining sets of invariant cosets (or invariant projected subsets, as we call them), i.e. sets of positions the subsets are projected onto. We provide their characterization that is used in Section 3.2 to provide a method for finding all possible defining sets using techniques of Mixed Integer Linear Programming. In Section 3.3 we describe the structure of weak key classes that stem from the invariant projected subsets, compute a number of elements in each class and compute time complexity of weak key recovery. Notably, we show that it is possible to speed up the search process by separating the search space into two steps.

3.1 Defining Sets of Invariant Projected Subsets

Definition 1. *A* projected subset $U \subset \mathbb{F}_2^n$ *is defined as*

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

We define $def_U \subset \mathbb{Z}_n$ *to be the subset of indexes* $i_1, \ldots, i_t$ *with the above property and for such a subset we define a vector* $val_U \in (\mathbb{F}_2 \cup \{'*'\})^n$ *as follows:* $val_U[i_j] = a_j, 1 \leq j \leq t$ *and* $val_U[i] =' *', i \in \mathbb{Z}_n \setminus def_U$. *We call* def_U *the* defining set *of* U.

Definition 2. *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 3. *Let $T \subseteq \mathbb{Z}_n$ with n divisible by 3. For $i \in \{1,2,3\}$ define*

$$T_i := \bigcup_{\substack{0 \le j < n/3 \\ |\{3j,3j+1,3j+2\} \cap T| = i}} \big[\{3j, 3j+1, 3j+2\} \cap T\big].$$

In other words, we divide the set $\{0,\ldots,n-1\}$ into triplets and then look which triplets have exactly i elements of T in them and collect these elements into T_i. Let U be a projected subset with the defining set def_U. For short notation we denote $U_i := (def_U)_i$ for $i = 1,2,3$.

Now let $E_{sk_1,sk_2,r} = S_{sk_2} \circ RC_r \circ P \circ XOR_{sk_1}$ be the round function of PRINTcipher-n for the round $1 \le r \le n$, where sk_1 and sk_2 are parts of the secret key as defined in Section 2.1.

Theorem 1. *Let $T \subset \mathbb{Z}_n$ with n divisible by 3 and*

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

Then there exists a projected subset $U \subset \mathbb{F}_2^n$ with $def_U = T$ such that $E_{sk_1,sk_2,r}(U) = U$ for some $sk_1 \in \mathbb{F}_2^n$ and $sk_2 \in \mathbb{F}_2^{2n/3}$ and any $r \ge 1$. The set $\{val_U[i] : i \in U_2\}$ is uniquely determined by T.
Vice versa: if for a projected subset $U \subset \mathbb{F}_2^n$ holds $E_{sk_1,sk_2,r}(U) = U$ for some $sk_1 \in \mathbb{F}_2^n$ and $sk_2 \in \mathbb{F}_2^{2n/3}$ and any $r \ge 1$, then def_U satisfies conditions (1.)–(3.) above.

Proof. ($\Rightarrow$) : We want to show that there exist $sk_1 \in \mathbb{F}_2^n$ and $sk_2 \in \mathbb{F}_2^{2n/3}$ such that $E_{sk_1,sk_2,r}(U) = U$ for all $r \ge 1$ for some projected subset U with $def_U = T$. For the construction of U we need to specify the values in val_U as well as the corresponding values of sk_1 and sk_2. Note that due to properties (2.) and (3.), the set of positions $P(T)$ is also a $\bar{1}$-subset. Now observe that positions of $P(T)$ define input masks to the S-Box layer S_{sk_2} and those from T define output masks of that layer. Indeed, we start with the positions from T at the beginning of the round. So in order to have an invariant property, we have to end with T as well. Due to properties (2.) and (3.) we have that only 0–0, 2–2 and 3–3 cases are possible for the S-Box masks (see Section 2.1). Note that in the case of a 2–2 mask most of the time the corresponding values of the key sk_2 are fixed and determined by T, see Table 3. In the case of ambiguity, we may take arbitrary value for the corresponding bit of sk_2. Also from Table 3 we see that for the 2–2 masks S-Box output values are uniquely defined in all cases, the same for input values except for the mask $+-+ \to +-+$. In this case arbitrary but fixed choice of the corresponding bits of sk_2 resolves the ambiguity. After the 2–2 cases are resolved (either through mandatory assignment or arbitrary but now fixed choice) we move to the 0–0 and 3–3 masks. Here both outputs of S-Boxes and the corresponding values of sk_2 can be assigned arbitrary values (sk_2 bits are equal in this case). Note that assignment of output values in the case of a 3–3

mask provides an assignment of values in $\{val_U[i] : i \in U_3\}$. We have that inputs and outputs to the S-Boxes at positions $P(T)$ and T resp. are now defined. This yields, via the operation XOR_{sk_1}, unique values for the key sk_1 at positions T. Moreover, the values in val_U are now fixed too. Therewith the set U is now fully defined by $def_U = T$ and such constructed val_U. Note that since outputs of S-Boxes at positions $T_2 = U_2$ (see Definition 3) are uniquely determined by the 2–2 masks, we have that the values $\{val_U[i], i \in U_2\}$ are uniquely determined. Condition (1.) makes sure that the round counter RC_i is avoided (i.e. there we have only 0–0 mask) and therewith there is no chance to change input/output values of S-Boxes, and so invariant property is preserved.
($\Leftarrow$) : For the reverse direction, note that def_U should satisfy (2.) and (3.), since for the invariant property we need an α–α mask and there are only 0–0, 2–2, and 3–3 of this sort, see Section 2.1. Again, the invariant property must not be spoiled by the round counter, thus the condition (1.) should be satisfied.

We refer to the example in Appendix B for assisting in understanding the notation and how the invariant property works. Note that in the case Theorem 1 holds, we have $E_{sk_1,sk_2,r}(V + d) = V + d$ for a certain linear subspace $V \subseteq \mathbb{F}_2^n$ and $d \in \mathbb{F}_2^n$. This $V + d$ is an *invariant coset* as per [12]. In order to follow our terminology, we prefer the term *invariant projected subset*. Also note that if $U = V + d$ is an invariant projected subset, then def_U is a disjoint union of U_2 and U_3: $def_U = U_2 \sqcup U_3$, see Definition 3. As a result of Theorem 1 we have a one-to-one correspondence between defining sets of invariant projected subsets and subsets of $\mathbb{Z}_n$ that satisfy (1.)–(3.) of Theorem 1.

3.2 Defining Sets via Polytopes in $\mathbb{Z}^n$

Now having the characterization of defining sets, we provide an efficient method of finding all subsets of $\mathbb{Z}_n$ satisfying conditions (1.)–(3.) of Theorem 1. 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 2. *Let IP_n be a subset in $\{0,1\}^n \subset \mathbb{Z}^n$, where n is divisible by 3, defined as a subset of those $x = (x_0, \ldots, x_{n-1}) \in \{0,1\}^n$ that satisfy*[2]

$$\begin{aligned} &x_{n-6} = \cdots = x_{n-1} = 0, \\ &\textit{for all } 0 \le j < n/3 : \end{aligned}$$

$$\begin{aligned} &x_{3j} + x_{3j+1} + x_{3j+2} = x_{P^{-1}(3j)} + x_{P^{-1}(3j+1)} + x_{P^{-1}(3j+2)}, \\ &x_{3j} + x_{3j+1} \ge x_{3j+2}, \quad x_{3j} + x_{3j+2} \ge x_{3j+1}, \quad x_{3j+1} + x_{3j+2} \ge x_{3j}, \\ &\sum_{i=0}^{n-1} x_i > 0. \end{aligned} \tag{2}$$

[2] Note that the arithmetic is integer.

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.)–(3.) of Theorem 1. Conversely, for each set T that satisfies (1.)–(3.) of Theorem 1 there exists $v \in IP_n$ such that $T = T_v$.

Proof. For the proof we need the following observation, which is proved by a direct inspection. Namely, 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

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

only the following cases for Hamming weights of a and b are possible: $wt(a) = wt(b) = i$ for $i = 0, 2, 3$. Now property (1.) follows directly from the condition in the first line of (2). It is not hard to see that the above observation easily yields claimed properties (2.) and (3.). The last line of the conditions (2) makes sure that we do not include the trivial all-zero solution.

It is not difficult to see that the converse statement is also true.

Combining Theorem 1 and Theorem 2 we obtain a one-to-one correspondence between defining sets of invariant projected subsets and vectors from $IP_n \subset \{0,1\}^n$.

So the question now 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 linear 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 case we do not actually have an optimization problem, but rather a constraint satisfaction problem that likewise can be solved by an MILP solver.

3.3 Families and Classes of Weak Keys

In this subsection we provide detailed results for PRINTcipher that can be obtained by the method from Section 3.2. We used SAGE computer algebra system [13] together with the MILP solver CPLEX[3] through the SAGE interface[4].

The results as presented in Tables 4 and 5 for both versions of PRINTcipher. In the tables we also indicate how our work improves the initial result of [12].

Each *family* is composed of potentially several *classes* of weak keys. By a class we understand a set of weak keys that ensure the invariant property for certain plaintext subsets, see more on that below. The *family*, in turn, unites all classes of weak keys stemming from invariant projected sets with the same defining set. We can provide only an upper bound on the number of all weak keys, since weak key classes can intersect, see "How many weak keys are there in a family defined

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

[4] See `http://sagemath.org/doc/reference/sage/numerical/mip.html?highlight=linear%20programming`

Table 4. Results and comparison for PRINTcipher–48

	Our analysis	Leander et al. [12]
# weak key families / all found?	64/**Yes**	2/No
upper bound on # weak keys	$2^{52.5}$	2^{52}
fastest time for key recovery in CP/KP scenario	2^{24}	2^{38}

Table 5. Results and comparison for PRINTcipher–96

	Our analysis	Leander et al. [12]
# weak key families / all found?	115,669/**Yes**	2/No
upper bound on # weak keys	$2^{117.7}$	2^{102}
fastest time for key recovery in CP/KP scenario	2^{30}	2^{76}

by def_U?" below. We also provide the minimum of the time complexities of all possible *classes* of weak keys.

Table 6 in Appendix A presents detailed results for PRINTcipher–48. Defining sets of weak key families found in [12] are marked in bold there.

Description of Weak Key Classes. From Theorem 1 we know that once we have a subset of positions $T \subset \mathbb{Z}_n$ that satisfies conditions (1.)–(3.) there exists a key $k = (sk_1, sk_2)$ and a projected subset U with $def_U = T$ such that $E_{sk_1,sk_2,r}(U) = U$ for all $r \geq 1$. Moreover, the set of projected values $\{val_U[i] : i \in U_2\}$ is uniquely determined by T. Now we would like to have more: we want to have a description of all *classes* of weak keys (i.e. those that preserve the invariant property) that correspond to $T = def_U$. These classes form a *family*.

For this we first need to fix values for all elements in $\{val_U[i] : i \in def_U\}$. Values $\{val_U[i] : i \in U_2\}$ are uniquely determined by T as per Theorem 1. Then choose a vector $v_3 \in \mathbb{F}_2^{|U_3|}$ and assign $val_U[i_j] = v_3[j], 1 \leq j \leq |U_3|$, where $U_3 = \{i_1, \ldots, i_{|U_3|}\}$ with $i_1 < \cdots < i_{|U_3|}$. So now the invariant projected subset U is fully defined. Let us show that there exists a *class* of weak keys $WK(def_U, v_3)$ such that for any $(sk_1, sk_2) = k \in WK(def_U, v_3) : E_{sk_1,sk_2,r}(U) = U$ for all $r \geq 1$. Let us describe separately the two parts sk_1 and sk_2 of a key $k = (sk_1, sk_2) \in WK(def_U, v_3)$.

We start with sk_2. Define for $0 \leq j \leq 3$:

$$S_j := \{0 \leq i < n/3 : |\{3i, 3i+1, 3i+2\} \cap def_U| = j\}.$$

So S_j collects those S-Boxes that have j bits both in input and output masks (i.e. a j–j mask). We construct the sk_2-part of k as follows.

- Bits $sk_2[2i]$ and $sk_2[2i+1]$ for $i \in S_3$ can attain arbitrary values. There are $\frac{2}{3}|U_3|$ such bits (a 3–3 mask).
- For $i \in S_2$, the bits $sk_2[2i]$ and $sk_2[2i+1]$ get their values according to column 3 of Table 3 by examining the corresponding parts of U_2 and $(P(def_U))_2$ to get $+/-$ masks. The star sign '*' means that the corresponding bit of sk_2 can attain arbitrary value. Denote by $U^* \subset \mathbb{Z}_{2n/3}$ the set of positions of sk_2 that correspond to '*'s. In the case of the $+-+ \to +-+$ mask assign arbitrary value to $sk_2[2i] = sk_2[2i+1]$. Let us denote the set of S-Boxes yielding the $+-+ \to +-+$ mask by $U^= \subset S_2 \subset \mathbb{Z}_{n/3}$.

- Bits $sk_2[2i]$ and $sk_2[2i+1]$ for $i \in S_0$ can attain arbitrary values (a 0–0 mask). There are $\frac{2}{3}n - \frac{2}{3}|U_3| - |U_2|$ such bits. Indeed, from the length of sk_2, which is $\frac{2}{3}n$, we subtract positions from S_3 ($= \frac{2}{3}|U_3|$) and S_2 ($= |U_2|$).

Now determine the sk_1 part. Let the choice of sk_2 be done as above and fixed.

- Denote $Y = XOR_{sk_1}(X)$. Note that $\{val_U[i] : i \in def_U\}$ are fixed, therefore $X_i, i \in def_U$ have fixed values. Now, since the choice of sk_2 bits for active S-Boxes (i.e. those with $i \in S_2 \cup S_3$) has been fixed, we have that inputs to S-Boxes at positions $P(def_U)$ have fixed values. Therefore $Y_i, \in P^{-1}(P(def_U)) = def_U$ are fixed as well. As a result $sk_1[i] = X_i \oplus Y_i, i \in def_U$ are now determined.
- Bits $sk_1[i], i \in \mathbb{Z}_n \setminus def_U$ can attain arbitrary values. There are $n - |def_U|$ such bits.

An explicit example explaining the notation is given below:

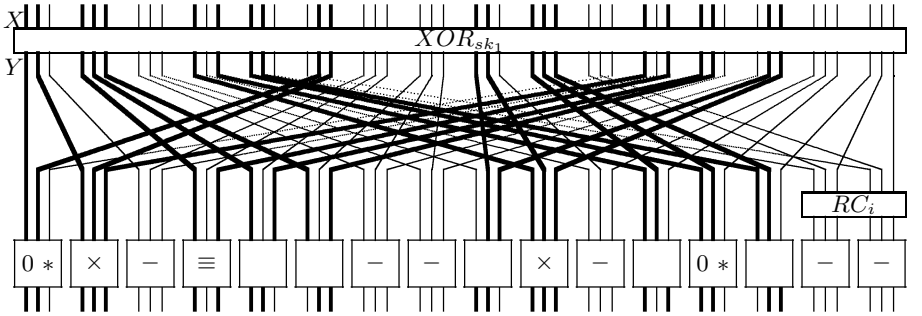


Fig. 1. Defining set no. 14, Table 6
Bold: positions of def_U and their transition through the round
$\times$: S-Boxes 1 and 9 belonging to S_3
$-$: S-Boxes 2,6,7,10,14, and 15 belonging to S_0
$\equiv$: S-Box 3 belongs to $U^= \subset S_2$
$*$: sk_2-positions of $U^* = \{1, 25\} \subseteq \mathbb{Z}_{32}$

Having the above construction, we obtain the following proposition.[5]

Proposition 1. *Let $T \subset \mathbb{Z}_n$ satisfy (1.)–(3.) of Theorem 1. Let $v_3 \in \mathbb{F}_2^{|T_3|}$ and $U \subset \mathbb{F}_2^n$ be a projected subset with $def_U = T$ and val_U composed of unique values for $\{val_U : i \in U_2\}$ and $\{val_U(U_3[i]) = v_3[i] : 1 \leq i \leq |U_3|\}$. Then for the set $WK(def_U, v_3)$ constructed as above holds*

$$E_{sk_1,sk_2,r}(U) = U, r \geq 1,$$

for any $(sk_1, sk_2) \in WK(def_U, v_3)$.

[5] For space reasons, we must refer to the full version of the paper for a formal proof.

How to Distinguish the Weak Keys? Now let us detail how to distinguish in the *chosen plaintext scenario* the weak keys belonging to $WK(def_U, v_3)$ for some defining set def_U of an invariant projected set U with a fixed choice of bits $\{val_U[i] : i \in U_2\}$ and $\{val_U[U_3[i]] = v_3[i] : 1 \le i \le |U_3|\}$. The attacker chooses an arbitrary $p \in U$ and encrypts p with a key $(sk_1, sk_2) = k \in \mathbb{F}_2^{5n/3}$ obtaining $c = E_{sk_1,sk_2,n}(p)$. If $c[i] = p[i]$ for all $i \in def_U$, i.e. $c \in U$, then k is a candidate element of $WK(def_U, v_3)$. To be sure we 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 $\#CP = \lceil \frac{5}{3}n/|def_U| \rceil$ plaintexts from U and their corresponding encryptions. For $\frac{5}{3}n = 80$ it is at most 5 chosen plaintexts, see also [12].

How Many Weak Keys Are There in a Family Defined by def_U? For a given def_U there are $2^{|U_3|}$ possibilities to assign values for the vector v_3. Therefore, each family of weak keys has $2^{|U_3|}$ classes. We want to determine

$$\Big| \bigcup_{v_3 \in \mathbb{F}_2^{|U_3|}} WK(def_U, v_3) \Big|.$$

The following bounds for the number above hold:

$$2^{|U_3|}|WK(def_U, 0)| \ge \Big| \bigcup_{v_3 \in \mathbb{F}_2^{|U_3|}} WK(def_U, v_3) \Big| \ge |WK(def_U, 0)|. \tag{3}$$

Indeed, if we fix vector $v_3 = 0$, then the number of elements in all classes is at least the number of elements in one class and is at most $2^{|U_3|}$ times the number of elements in that class, since

$$|WK(def_U, a)| = |WK(def_U, b)| \quad \forall a, b \in \mathbb{F}_2^{|U_3|}.$$

We cannot simply say that $|\cup_{v_3 \in \mathbb{F}_2^{|U_3|}} WK(def_U, v_3)|$ is equal to the upper bound, since it may happen that $WK(def_U, a) \cap WK(def_U, b) \neq \oslash$ for some $a \neq b$. Note, however, that for many def_U's the value $|\cup_{v_3 \in \mathbb{F}_2^{|U_3|}} WK(def_U, v_3)|$ *does* attain the upper bound. We skip the details due to space constraints.

Now as to the computation of $|WK(def_U, 0)|$ (or, equivalently $|WK(def_U, a)|$ for any other $a \in \mathbb{F}_2^{|U_3|}$), we just follow the lines of the argument preceding Proposition 1. Namely, if we compute the number of arbitrarily assigned key bits, we need to add the numbers of bits of $sk_1[i], i \in \mathbb{Z}_n \setminus def_U$ $(= n - |def_U|)$, from $sk_2[2i], sk_2[2i+1]$ for $i \in S_0$ and $i \in S_3$ $(= \frac{2}{3}n - \frac{2}{3}|U_3| - |U_2|$ and $= \frac{2}{3}|U_3|$ resp.), as well as the number of bits of $U^* \subset \mathbb{Z}_{2n/3}$ $(= |U^*|)$ and one bit per $U^= \subset \mathbb{Z}_{n/3}$ $(= |U^=|)$. Summing up, we obtain

$$\log_2 |WK(def_U, 0)| = \frac{5}{3}n - |def_U| - |U_2| + |U^*| + |U^=|. \tag{4}$$

Therewith the upper bound (and often the actual value) is

$$\log_2 \Big| \bigcup_{v_3 \in \mathbb{F}_2^{|U_3|}} WK(def_U, v_3) \Big| \ge \frac{5}{3}n - 2|U_2| + |U^*| + |U^=|. \tag{5}$$

Summing up the numbers obtained via (5) over all def_U we have an upper bound on the overall number of weak keys: it is $2^{52.51}$ for PRINTcipher–48 and $2^{117.7}$ for PRINTcipher–96.

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 are not deduced immediately from the distinguishing phase. Of course, we may simply use several known plaintexts and brute force the keys in $WK(def_U, v_3)$, but we can actually do better: we can separate the key recovery process in two consecutive steps, each having smaller time complexity.

Step 1 (chosen plaintext). Using chosen plaintexts from the distinguishing phase brute force inactive key bits, i.e. $sk_1[i], i \in \mathbb{Z}_n \setminus def_U$ and $sk_2[2i], sk_2[2i+1], i \in S_0$. For the actual implementation of this step we assign arbitrary values to $sk_2[2i], sk_2[2i+1], i \in S_3$, $sk_2[2i] = sk_2[2i+1], i \in U^=$, $sk_2[i], i \in U^*$ and compute the corresponding bits of sk_1 to get candidate keys for testing.

Note that after assigning arbitrary values to certain key bits, we rely on the assumption that the remaining "cipher" behaves as a random permutation. Since the overall number of plaintext bits in several chosen plaintexts exceeds the number of key bits we brute force, we expect, as usual, that we have a unique solution for these key bits. Due to certain degeneration properties of PRINTcipher it is not always true, however. Without going into technical details, we just state that there exist certain positions at which bits of sk_1 may attain arbitrary values in this chosen plaintext scenario. Therefore we assign in this step some arbitrary values to these bits, call them *weird bits*, and brute force them in the next step. Denote the number of such weird bits w.

Now the computation of the work factor is similar to the one for the number of weak keys as above. We have the work factor of Step 1:

$$\log_2 WF_1 = n - |def_U| + \frac{2}{3}n - \frac{2}{3}|U_3| - |U_2| - w = \frac{5}{3}(n - |def_U|) - \frac{1}{3}|U_2| - w. \quad (6)$$

Note that the number of chosen plaintexts from the distinguishing phase is indeed enough, since the remaining "block length" $n - |def_U|$ times $\#CP$ exceeds $\log_2 WF_1$ for all def_U.

Step 2 (known plaintext). This is as far as we can go with chosen plaintexts. We cannot distinguish bits $sk_2[2i], sk_2[2i+1], i \in S_3 \cup U^=$ and $sk_2[i]$ for $i \in U^*$ having only these plaintexts. Therefore, for the second phase we take *one* known plaintext that is not in U and brute force these key bits. Note, however, that now the sk_1-bits $sk_1[P^{-1}(3i)]$, $sk_1[P^{-1}(3i+1)], sk_1[P^{-1}(3i+2)]$ corresponding to $i \in S_3$ cannot be determined from one round as in the case of chosen plaintexts where the invariant property holds. Similar situation is with the bits $sk_1[P^{-1}(3i)], sk_1[P^{-1}(3i+2)]$ with $i \in U^=$. So we have to brute force these bits as well. We also brute force the weird bits in this step. We have

$$\log_2 WF_2 = \frac{5}{3}|U_3| + |U^*| + 3 \cdot |U^=| + w. \quad (7)$$

Combining (6) and (7) we have that the overall work factor for the key recovery is

$$\log_2 WF \approx \max\{\log_2 WF_1, \log_2 WF_2\} =$$
$$\max\left\{\frac{5}{3}(n - |def_U|) - \frac{1}{3}|U_2| - w, \frac{5}{3}|U_3| + |U^*| + 3 \cdot |U^=| + w\right\}. \quad (8)$$

Now it is interesting to point out that a simple brute force approach would just search through a weak key class in the known plaintext scenario and its work factor would be the number of elements given by (4). We define a *gain* of our key recovery procedure over the simple brute force as a difference between (4) and (8), which gives a logarithm of the speed-up factor. Gains for all classes in the case of PRINTcipher–48 are given in Appendix A. For PRINTcipher–96 the largest gain is 27 yielding a speed-up factor of 2^{27} which is quite substantial.

See the example in Appendix B for a detailed work-out of the above computations.

Remark 1. The key recovery procedure in chosen/known plaintext scenario as described above was implemented for PRINTcipher–48 and tested for weak key classes that allow practical time key recovery. We could always correctly and uniquely recover the secret key.

Countermeasures against the Attack. Note that due to existence of the round constant RC_i in the last 6 bits (corresponding to the last two S-Boxes no. $n/3-2$ and $n/3-1$), our invariant projected subsets should not be active in these two S-Boxes. As can be seen from Table 6, for PRINTcipher–48 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 constant over the last three S-Boxes (two bits of the constant 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 placing the round constant (or any three of them) does not provide a secure solution, since classes no. 23 and 58 from Table 6 avoid them, providing at least 2^{50} weak keys. However, the "SPONGENT" solution obviously defeats the classes no. 44 and 47 found in [12].

Note that opposed to the 48-bit case, in the case of PRINTcipher–96 there *exist* defining sets that avoid the last three S-Boxes. In fact, there are 28 such sets. So the countermeasure suggested in [12] for PRINTcipher–48 does not work for all families here. Still, there is a collection of combinations of three S-Boxes that cannot be avoided by any defining set. The S-Boxes 0,1,23 is one possible solution among many others. So, in order to defeat the attack, one has to spread out the round counter over these S-Boxes.

Important note: One may argue that the invariant attack in its complete form as described in this paper is of not much value, since there still exist simple countermeasures that defeat all families of weak keys. Note, however, that since our analysis is *complete*, i.e. we have found *all* weak key families, we may argue about security of the cipher against the invariant attack. Whereas having only partial results of [12] it is not possible. In fact, as we have seen above, countermeasures for PRINTcipher–96 that may be hinted from [12] are not effective for many families of weak keys.

4 Related and Future Work

In this paper we have undertaken a complete study of the invariant coset attack initially presented at CRYPTO 2011 by Leander *et al.* By explicitly providing characterization of defining sets of invariant projected subsets and weak key families and classes we were able to recover all families of keys that are weak in the sense of the invariant coset attack. We also showed that both versions of the cipher can be made immune to this attack at no additional cost. The latter conclusion was only possible to make due to completeness of our analysis.

Note also that methods of this paper, such as finding all defining set as per Section 3.2, are interesting on their own. It can be shown that similar approach can be employed in other contexts, e.g. for finding optimal guessing strategies for algebraic cryptanalysis. These directions put the methods in a more general context that deserves further investigation.

Acknowledgements. The first author is supported by the German Science Foundation (DFG) grant BU 630/22-1. The second author is supported in part by the NSF grant CNS-1117936. We would like to thank Gregor Leander for his critical comments that helped to improve the paper considerably. We are also thankful to Mohamed Ahmed Abdelraheem for providing the C implementation of PRINTcipher that was used in the implementation of the attacks. Finally, we thank anonymous referees for their numerous valuable suggestions and remarks.

References

1. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
2. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
3. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
4. Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A Block Cipher for IC-Printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010)
5. Yap, H., Khoo, K., Poschmann, A., Henricksen, M.: EPCBC - A Block Cipher Suitable for Electronic Product Code Encryption. In: Lin, D., Tsudik, G., Wang, X. (eds.) CANS 2011. LNCS, vol. 7092, pp. 76–97. Springer, Heidelberg (2011)
6. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: spongent: A Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
7. Abdelraheem, M.A., Leander, G., Zenner, E.: Differential Cryptanalysis of Round-Reduced PRINTcipher: Computing Roots of Permutations. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 1–17. Springer, Heidelberg (2011)

8. Ågren, M., Johansson, T.: Linear Cryptanalysis of PRINTcipher – Trails and Samples Everywhere. In: Bernstein, D.J., Chatterjee, S. (eds.) INDOCRYPT 2011. LNCS, vol. 7107, pp. 114–133. Springer, Heidelberg (2011)
9. Karakoç, F., Demirci, H., Harmancı, A.E.: Combined Differential and Linear Cryptanalysis of Reduced-Round PRINTcipher. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 169–184. Springer, Heidelberg (2012)
10. Bulygin, S., Buchmann, J.: Algebraic Cryptanalysis of the Round-Reduced and Side Channel Analysis of the Full PRINTCipher-48. In: Lin, D., Tsudik, G., Wang, X. (eds.) CANS 2011. LNCS, vol. 7092, pp. 54–75. Springer, Heidelberg (2011)
11. Zhao, X., Wang, T., Guo, S.: Fault Propagate Pattern Based DFA on SPN Structure Block Ciphers using Bitwise Permutation, with Application to PRESENT and PRINTcipher, ePrint, http://eprint.iacr.org/2011/086.pdf
12. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 206–221. Springer, Heidelberg (2011)
13. William Stein, S., et al.: SAGE Mathematics Software, pp. 593–599. The Sage Development Team (2008), http://www.sagemath.org

A The List of Defining Sets of All Possible Invariant Projected Subsets of PRINTcipher–48

Table 6 presents defining sets of all possible invariant projected sets for PRINTcipher–48.

B Example of Computations for Weak Keys Number and Work Factors

Example 1. In this example we would like to provide a detailed workout of the computations discussed in Section 3.3. We work with PRINTcipher–48 and a specific defining set (no. 5 in Table 6):

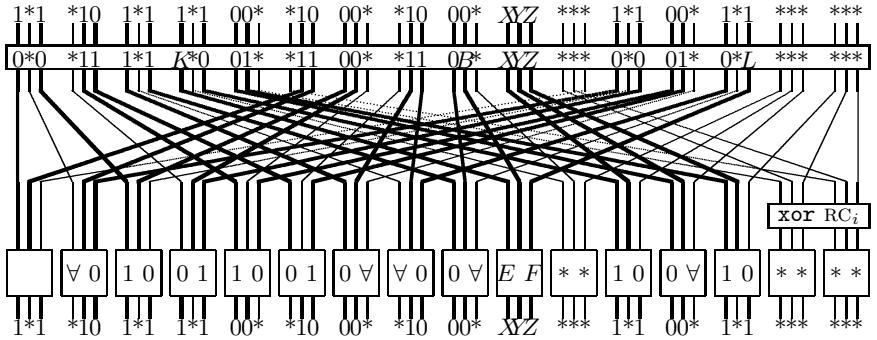
$$def_U = \{0, 2, 4, 5, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 33, 35, 36, 37, 39, 41\}.$$

In the figure below the positions of def_U and their transition through the round is denoted in bold.

Note that inputs and outputs of the S-Boxes in this example are uniquely determined by the set def_U and Table 3. In the figure the symbol $\forall$ means that any value of the corresponding sk_2-bit works for the invariant property. For ABC and XYZ it holds that $SBOX_{EF}(ABC) = XYZ$. Note that XYZ itself can attain arbitrary values and ABC is determined by these and EF bits of the key sk_2. The "passive" bits are denoted with "*" and can attain arbitrary value. We have $|U_2| = 24, |U_3| = 3, |U^*| = 5, |U^=| = 0, w = 0$. The following table makes the situation on the figure in a bit more formal way:

Table 6. Defining sets of invariant projected subsets of PRINTcipher–48
Bold font: families found in [12]
$\#k$: number of elements in the corresponding family
$\log WF$: logarithm of the time complexity (work factor) of key recovery for each class in the family
Gain: logarithm of the speed-up factor of the mixed CP/KP scenario over the simple brute force in KP
$\perp$: an upper bound, not an exact value

No.	def_U	$\#k$	$\log WF$	Gain
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]	$36^{\perp}$	25	11
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	26	10
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	27	7
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	25	10
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	27	7
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	27	6
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	27	7
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	27	7
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]	$34^{\perp}$	24	10
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	24	9
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	25	10
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]	$33^{\perp}$	24	9
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	24	8
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	25	9
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	26	7
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	24	8
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	26	6
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	26	6
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	26	5
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	26	5
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	27	5
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	27	4
23	[6, 8, 9, 11, 12, 13, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 39, 41]	50	38	6
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	26	8
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	26	7
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	26	7
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	26	6
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	27	6
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	27	4
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]	$30^{\perp}$	25	11
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	24	11
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	25	11
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]	$35^{\perp}$	24	11
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	24	10
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	24	10
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	26	8
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	26	8
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	26	7
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	27	7
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	26	8
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	26	11
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	26	10
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	27	9
44	**[0, 1, 4, 5, 12, 13, 16, 17, 24, 25, 28, 29, 36, 37, 40, 41]**	**51**	**48**	**3**
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	26	8
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	27	4
47	**[0, 1, 3, 4, 5, 9, 11, 12, 13, 16, 17, 24, 25, 27, 28, 29, 33, 35, 36, 37, 40, 41]**	**51**	**38**	**7**
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	27	6
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	27	5
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	27	5
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	27	5
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	26	6
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	27	3
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	27	5
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	27	4
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	27	5
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	27	3
58	[6, 7, 9, 11, 12, 13, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 34, 35, 36, 38, 40, 41]	48	38	4
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	27	5
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	25	9
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	27	7
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	26	7
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	27	8
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	26	6



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

Now, 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	∀0	10	01	10	01	0∀	∀0	0∀	EF	**	10	0∀	10	**	**
$out =$	1*1	*10	1*1	1*1	00*	*10	00*	*10	00*	XYZ	***	1*1	00*	1*1	***	***

In this 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 the S-Boxes (or, equivalently, to the outputs of the diffusion layer) 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 and EF.

For a specific example, let us take $XYZ = 000$ so that $v_3 = (0, 0, 0)$. So we are working now with

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

Note that independently of the values of EF we have $SBOX_{EF}^{-1}(000) = 000 = ABC$ and so $K = L = 1$. The weak keys from $WK(def_U, v_3)$ are of the form (sk_1, sk_2), where

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

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

Now let us compute the number of elements in $WK(def_U, 0)$. Using (4) we have

$$|WK(def_U, 0)| = 2^{80-27-24+5} = 2^{34}.$$

The upper bound provided by (3) is actually tight in this case and

$$\Big|\bigcup_{v_3 \in \mathbb{F}_2^3} WK(def_U, v_3)\Big| = 2^3 \cdot 2^{34} = 2^{37}.$$

Now as to the time complexity of the key recovery, from (8) we have

$$\log_2 WF = \max\left\{\frac{5}{3} \cdot 21 - \frac{1}{3} \cdot 24, \frac{5}{3} \cdot 3 + 5\right\} = \max\{27, 10\} = 27.$$

So for the key recovery it takes around 2^{27} encryptions having 3 chosen and 1 known plaintext. We have a *gain* of $34 - 27 = 7$ bits compared to the simple brute force attack.

It is not hard to see that the class of weak keys $WK(def_U, 0)$ is different from the ones presented in [12]. Indeed, for example the keys with

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

do not belong to the class defined by def_U no. 44, since there $sk_1[1] = 1$ and the keys with

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

do not belong to the class defined by def_U no. 47, since there $sk_1[40] = 1$, see [12].