Algebraic cryptanalysis of the round-reduced and side channel analysis of the full PRINTCipher-48

Stanislav Bulygin

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

Abstract. In this paper we analyze the recently proposed light-weight block cipher PRINTCipher. Applying algebraic methods and SAT-solving we are able to break 8 rounds of PRINTCipher-48 with only 2 known plaintexts and 9 rounds under some additional assumptions. We show that it is possible to break the full 48-round cipher by assuming a moderate leakage of internal state bits or even just Hamming weights. Such a simulation side-channel attack has practical complexity. We investigate applicability of our method to cryptanalysis of the full PRINTCipher-48.

Keywords: Algebraic cryptanalysis, SAT-solving, PRINTCipher, MiniSAT, CryptoMiniSAT

1 Introduction

The target of this paper is the light-weight block cipher PRINTCipher proposed in 2010 at CHES [1]. PRINTCipher proposes a security solution for low cost devices such as RFID tags. In particular, PRINTCipher aims to facilitate secure usage of integrated circuit (IC) printing for RFID tags. An interesting feature of such "printing" is that a tag can obtain key dependent circuitry already at the manufacturing phase. As a result, the authors of [1] propose to use the same key for all rounds, but implementing key-dependent S-Boxes that may be "printed" when a tag is manufactured. In cryptographic sense, PRINTCipher pushes even further the limits of lightweight block cipher design set by such ciphers as PRESENT [2] and KATAN/KTANTAN-family [3]. In fact the authors claim that e.g. PRINTCipher-48 that is claimed to provide 80-bit security may be implemented with almost three times as less gate equivalents (GEs) than the 80-bit version of PRESENT.

Naturally, the question of security for such an extremely lightweight design arises. The first cryptanalytic results known to us is the differential cryptanalysis of 24 rounds of the cipher using the entire code-book [4]. A much more powerful result is proposed in [5]. In this paper the authors show that there exist a non-negligible portion (2^{52} for PRINTCipher-48 with 80 bit keys and 2^{102} for PRINTCipher-96 with 160 bit keys) of weak keys. For these keys the cipher is easily broken with only several plaintext pairs, which makes the attack practical for these weak keys. Note, however, that the invariant coset attack of [5] may be easily overcome by adjusting the round counter appropriately, see Section 2.3 of [5]. Therefore, analyzing PRINTCipher still is of interest and serves further and deeper understanding of light weight design principles, which PRINTCipher is using extensively.

Note that although the IC printing technology itself is "in its infancy", side channel security is still of theoretical interest. The preprint [6] addresses, in particular, security of PRINTCipher against the fault injection. There the authors show that by introducing faults into just one nibble position at some last rounds and having 12–24 effective faulty samples it is possible to dramatically reduce the key space to be searched.

In this paper we propose algebraic cryptanalysis of round reduced PRINTCipher-48 using SAT-solving. Algebraic cryptanalysis combined with SAT-solving has become a popular tool in analyzing stream (e.g. [7]) and block ciphers (e.g. [9,8,10]). We show that we are able to break 8 rounds of the cipher having only 2 known plaintext pairs faster than the brute force. We are

able to break 9 rounds assuming knowledge of 50 key bits out of 80. The success of this attack (i.e. it is faster than the brute force) is around 3/4. Note that although the number of rounds we can break is rather low, the data complexity of these attacks is minimal. As indicated in [11], it is important to consider low data complexity attacks on block ciphers due to possible applications of block ciphers other than encryption.

Moreover, we show that assuming knowledge of internal state bits at round 4, we are able to break the full 48 rounds of PRINTCipher-48 in practical time. For example, assuming knowing 6 output bits of 2 S-Boxes (3 bits each) at round 4 for 12 known plaintext/ciphertext pairs, we estimate total key recovery in less than 2 hours on average. Relaxing the knowledge of internal state bits by knowing only Hamming weights of inputs and outputs to the 2 first S-Boxes at round 4, we estimate recovery of the 80-bit key in less than 3 days on average using 12 pairs of plaintext/ciphertext.

The paper is structured as follows. Section 2 gives a brief description of PRINTCipher-48. Then in Section 3 we elaborate on algebraic representations of the S-Boxes. Section 4 is a brief overview of SAT-solving technique and conversion techniques used in this paper. Section 5 provides detailed analysis of tools and conversion techniques that are further used in Section 6 to attack the cipher. Section 6 has several subsections. Section 6.1 gives results of algebraic cryptanalysis of the round-reduced PRINTCipher-48. Section 6.2 observes an interesting property of the cipher, which is then applied to side channel analysis in Section 6.3. Final Section 6.4 discusses potential applications of the idea in Section 6.2 to the attack on the full PRINTCipher-48 with 48 rounds. We conclude in Section 7 and outline there some open problems.

Main results of the paper are contained in Sections 6.1, 6.3, and 6.4.

2 PRINTCipher

PRINTCipher is a substitution-permutation network. The cipher is largely inspired by the lightweight block cipher PRESENT [2]. The main differences with PRESENT is absence of the key schedule (all round key 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 80 bit key and has 48 rounds, PRINTCipher-96 encrypts 96 bit blocks with 160 bit key and has 96 rounds. Here we present a short overview of the cipher, referring the reader to [1] for a more detailed description and analysis. In this paper we concentrate on the smaller version PRINTCipher-48. The encryption process of PRINTCipher-48 is organized as in Algorithm 1.

Algorithm 1 Encryption with PRINTCipher-48

Require:
- 48-bit plaintext p
- 80-bit key $k = (sk_1, sk_2)$, where sk_1 is 48 bits and sk_2 is 32 bits

Ensure: 48-bit ciphertext c

Begin
$state := p$
for $i = 1, \ldots, 48$ **do**
  $state := state \oplus sk_1$
  $state := Perm(state)$
  $state := state \oplus RC_i$
  $state := SBOX(state, sk_2)$
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. It allows full dependency on plaintext and key bits already at

round 4. RC_i for $i = 1, \ldots, 48$ is a 6-bit round counter that is placed in the last two 3-bit nibbles. The S-Box layer $SBOX$ is a layer of 16 3-bit S-Boxes numbered $0, \ldots, 15$, where each S-Box is chosen according to the value of 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 [1]. One may also consider this as a composition of a key dependent bit permutation that acts on nibbles of 3 bits and then followed by the layer of fixed S-Boxes, each one being a 3-bit S-Box with the truth table

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

This S-Box, called V_0 in [1], is preceded by a key-dependent permutation defined by the following table:

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)

Here the three input bits are permuted according to the two consecutive key bits from the key sk_2 called a_0 and a_1. Figure 1 provides an illustration for one encryption round of PRINTCipher-48.

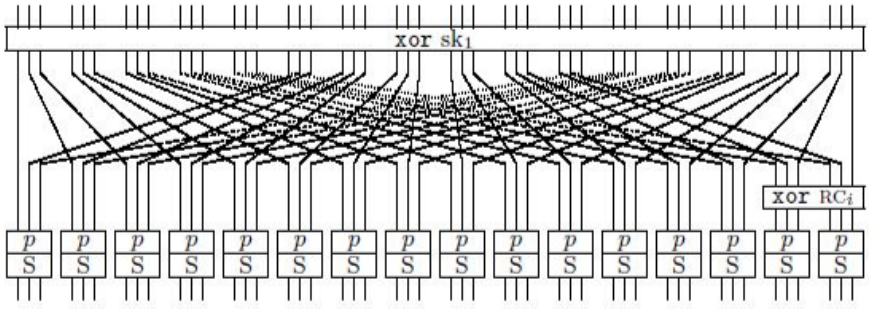

Fig. 1. Round function of PRINTCipher-48, cf. Figure 1 of [1]

As has been already mentioned in Section 1, in [5] the authors propose an attack on PRINTCipher that works for a large set of weak keys. In Section 2.3 of [5] it is indicated that it is possible

to overcome the attack by spreading the round counter to e.g. the last three nibbles. We apply this change and use the modified cipher when working with the full 48-round cipher in Sections 6.2–6.4.

In our attacks we extensively use another notion defined for bit-permutations in SP-networks. This is the notion of a *low diffusion trail*. The effect of a low diffusion trail was observed already for PRESENT and was used as a tool for statistical saturation attacks [12]. An observation on low diffusion trails is also made by the authors of [1] in Section 4.4. One fixes certain S-Box positions and computes how many bits stay within the fixed S-Boxes after the application of the bit permutation layer. Low diffusion trails correspond to those S-Box positions with the highest count. For example, looking at Figure 1 one sees that if we fix S-Boxes 0,1 and 5, then 5 out of 9 output bits at round i are input bits to these three S-Boxes at round $i+1$. In fact positions $\{0,1,5\}$ provide a low diffusion trail for 3 S-Box positions. We cite now Table 1 from [1] which provides examples of low diffusion trails and append it with some other trails that we used for the number of positions 2, 9 and 10, see Table 1. We used these trails in our experiments, see in particular Section 5.3 and 6.1.

Table 1. Low diffusion trails for PRINTCipher-48

# of S-Boxes in trail	Example trail	# bits in the trail	Ratio
2	$\{0,1\}$	2	2/6
3	$\{0,1,5\}$	5	5/9
4	$\{0,1,5,15\}$	7	7/12
5	$\{4,10,12,14,15\}$	9	9/15
6	$\{0,1,2,5,6,7\}$	12	12/18
7	$\{3,8,9,10,11,13,15\}$	14	14/21
8	$\{0,1,4,5,10,12,14,15\}$	18	18/24
9	$\{0,1,2,3,5,10,11,14,15\}$	20	20/27
10	$\{0,1,3,4,5,10,11,12,14,15\}$	24	24/30

3 Algebraic description

In this section we give an algebraic description of the PRINTCipher-48. As usual in algebraic cryptanalysis, the most interesting part is to describe non-linear transformations, the S-Boxes. In the case of PRINTCipher we have key dependent S-Boxes, so we have to treat this situation. We show which alternatives exist for algebraic descriptions of an S-Box.

First of all, let us describe the system of equations for r rounds of PRINTCipher-48. We are working in the Boolean ring with variables

- $X_i := (X_{i,j}), 0 \le i \le r, 0 \le j \le 47$ *input* variables
- $Y_i := (Y_{i,j}), 1 \le i \le r, 0 \le j \le 47$ *output* variables
- $K_1 := (K_{1,j}), 0 \le j \le 47$ *key-xor* variables
- $K_2 := (K_{2,j}), 0 \le j \le 31$ *key-permutation* variables

Now schematically the system of equations $Sys(p,c)$ is as follows

$$\begin{cases} X_0 = p, \\ X_i = Y_{i-1} \oplus K_1, 1 \le i \le r, \\ Y_i = SBOX(Perm(X_i) \oplus RC_i, K_2), 1 \le i \le r, \\ Y_r = c. \end{cases}$$

Variables X_0 and Y_r do not actually appear, since they are replaced by the values of the corresponding plaintext/ciphertext pair (p, c). Our task is to find solutions for the variables K_1 and K_2 which then give us the key k that encrypts p to c. The variables K_i represent the subkey sk_i for $i = 1, 2$. Some remarks on notation. It is clear that in the second line we simply do bitwise XOR of the variables. The second line assumes that the variables are grouped by three and two corresponding key-permutation variables. So $SBOX$ is the concatenation of 16 non-linear maps. The map $Perm$ is a bit permutation, see Algorithm 1 and Figure 1. For example for the first S-Box we have equations:

$$Y_{i,j} = SBOX_j(X_{i,0}, X_{i,16}, X_{i,32}, K_{2,0}, K_{2,1}), 0 \leq j \leq 2,$$

since RC_i is always zero at those positions. Another important observation is that since the key length of PRINTCipher-48 is larger than its block length, we actually need two plaintext/ciphertext pairs to uniquely determine the key. So actually we need two systems $Sys(p_1, c_1)$ and $Sys(p_2, c_2)$. This is not necessary if we assume that the key-permutation variables are known or guessed: then one pair suffices to determine sk_1, at least for r large enough.

Our next goal is to look closely at the $SBOX$ map. This map is a concatenation of 16 smaller maps $sbox(sk_2)$, which depend in the key sk_2. As we have seen from the previous section, each $sbox$ can be considered via two equivalent definitions:

1. A composition of the key-dependent permutation $SP : \mathbb{F}_2^3 \times \mathbb{F}_2^2 \to \mathbb{F}_2^3$ and the fixed S-Box V_0.

2. As a key-dependent S-Box V_i, where i is determined by the two bits of the key sk_2.

Let us first consider the definition (1). We need to find equations that describe the map SP. Introducing local notation, let $X = (X_0, X_1, X_2)$ be the input variables $A = (A_0, A_1)$ be the key-permutation variables and $W = (W_0, W_1, W_2)$ be the output variables. Then SP can be described as

$$W = SP(X, A) = (f_{ij}(A)) \cdot X^T,$$

where for each assigned value for $A = (A_0, A_1)$, the matrix $F := (f_{ij}(A))$ is a permutation matrix according to the table in Section 2. By using interpolation techniques, we find that f_{ij}'s are polynomials in A-variables of degree at most 2. The matrix is as follows:

$$F = \begin{pmatrix} A_0 + 1 & A_0A_1 + A_0 & A_0A_1 \\ A_0A_1 + A_0 & A_0 + A_1 + 1 & A_0A_1 + A_1 \\ A_0A_1 & A_0A_1 + A_1 & A_1 + 1 \end{pmatrix}.$$

Therefore, we obtain cubic equations for the map SP: quadratic in key-permutation variables and linear in the input variables. Appendix A.1 contains the equations describing this map.

Now the S-Box V_0 is a "classical" S-Box and standard techniques may be used to describe it. Let $Y = (Y_0, Y_1, Y_2)$ be the output variables of the S-Box, so that $Y = V_0(W)$. Since V_0 is a $\mathbb{F}_2^3 \to \mathbb{F}_2^3$ S-Box, we may obtain explicit quadratic equations for the output variables in terms of the input variables:

$$\begin{aligned} Y_0 &= W_1W_2 + W_0, \\ Y_1 &= W_0W_2 + W_0 + W_1, \\ Y_2 &= W_0W_1 + W_0 + W_1 + W_2. \end{aligned}$$

Likewise one may obtain explicit equations of W-variables in terms of Y-variables:

$$\begin{aligned} W_0 &= Y_1Y_2 + Y_0 + Y_1, \\ W_1 &= Y_0Y_2 + Y_1, \\ W_2 &= Y_0Y_1 + Y_0 + Y_1 + Y_2. \end{aligned}$$

In total there are 14 linearly independent quadratic equations that describe V_0.

Of course we may write equations for the $sbox$ in terms of X-, A-, and Y-variables only, eliminating the W-variables. Therewith, we obtain quartic equations for $sbox$: quadratic in the

key-permutation and the input variables. We list these equations in Appendix A.2. If we assume that values of the key-permutation variables are known (sk_2 is known) or guessed, the approach (2) is be useful. One can obtain equations for the S-Boxes $V_i, i = 1, 2, 3$ analogously to the V_0 S-Box.

Interestingly enough, it is possible to describe the map SP by equations of degree at most 2. One needs 6 quadratic equations and 1 linear equation for this. See Appendix A.3. This fact means that in principle it is possible to describe the key dependent S-Box by equations of degree at most 2 having three additional W-variables per S-Box.

We do not see immediate implications of the following fact, still we find it interesting to be noted. There exist 10 polynomials from the ideal describing the key dependent S-Box, each being a product of two linear polynomials. We list these in Appendix A.4.

4 SAT techniques for algebraic system solving

Using SAT-solvers in cryptanalysis is a quite recent trend. The idea is to translate an algebraic representation of a cryptographic primitive, such as block or stream cipher, into a satisfiability problem. Therewith, the problem of solving an algebraic system of equations in a Boolean ring is replaced with finding a satisfiability assignment of variables in a logical formula (or proving that such an assignment does not exist). Whereas equations from an algebraic representation are in the algebraic normal form (ANF), the resulting satisfiability problem is usually in the conjunctive normal form (CNF). In block cipher cryptanalysis there are examples of using SAT-solving in analyzing KeeLoq [8] and DES [9], whereas for stream ciphers an example target is Grain [7]. Although the problem of finding a satisfiability assignment of variables, as well as solving non-linear systems, is NP-hard in general, highly tuned SAT-solvers can solve particular instances occurring in practice amazingly fast. Two most commonly used SAT-solvers in the cryptographic context are MiniSAT2 [13] and recently proposed CryptoMiniSAT2 [14]. The latter uses ideas of MiniSAT adding a lot of new heuristics and the possibility to handle long XOR chains more efficiently.

As we said, applying SAT-solving techniques assumes translation of the algberaic representaion of a cipher into a logical formula. There exist several methods for doing so. In our experiments we rely on the two methods: the one due to Courtois, Bard, Jefferson [15] and another one that is based on the truth tables [16,7].

We would like to make a remark about estimating complexity of SAT-solving. Whereas memory consumption of SAT-solvers is very modest (at least compared to algebraic solvers that compute Gröbner bases), estimating time complexity is really an issue. The problem is that due to highly heuristic and randomized nature of SAT solvers (at least those based on the DPLL algorithm, such as MiniSAT and CryptoMiniSAT) the execution time varies very significantly for each trial run and also for runs of similar problems. Sometimes one observes a difference of factor 1000. Since variance for time measurements is so high, it is hard to make estimates even after many runs. Still, in this paper we apply the method of averaging running times of 100 trials for each experiment in question. This is similar to approaches of other authors in the field, e.g. [10,17]. Our experiments with some 1000-trial instances suggest that 100 is quite accurate in terms of the average time. A more advanced way of estimating time complexity would be statistical hypothesis testing. We set using such an estimation tool as a future work.

4.1 Conversion techniques

Conversion due to Bard, Courtois, Jefferson

ANF to CNF conversion of Bard, Courtois, Jefferson first represents separate monomials occurring in an algebraic system as a conjunction of clauses, which in turn are disjunctions of variables or their negations. Then one adds new variables corresponding to these monomials, so that all equations become linear in these new variables. The next task is to represent these linear polynomials as conjunctions of clauses. The problem here is that representing a linear relation (a

XOR-chain) needs exponentially many clauses in the number of variables. Therefore, one "cuts" a linear relation in several ones by introducing new variables. Representing one cut XOR-chain then needs relatively few clauses.

As an example, consider an equation

$$xyz + xy + yz + xz = 0.$$

We introduce new variables $a := xyz, b := xy, c := yz, d := xz$. Then a term $a = xyz$ is described by a logical formula

$$(x \vee \bar{a}) \wedge (y \vee \bar{a}) \wedge (z \vee \bar{a}) \wedge (a \vee \bar{x} \vee \bar{y} \vee \bar{z}).$$

Similarly one can write the terms b, c and d. Resulting linear relation $a + b + c + d = 0$ may be written as

$$(a \vee b \vee c \vee d) \wedge (a \vee b \vee \bar{c} \vee \bar{d}) \wedge (a \vee \bar{b} \vee c \vee \bar{d}) \wedge (a \vee \bar{b} \vee \bar{c} \vee d) \wedge$$
$$\wedge (\bar{a} \vee b \vee c \vee \bar{d}) \wedge (\bar{a} \vee b \vee \bar{c} \vee d) \wedge (\bar{a} \vee \bar{b} \vee c \vee d) \wedge (\bar{a} \vee \bar{b} \vee \bar{c} \vee \bar{d}).$$

As we see, the latter formula contains $1 + 6 + 1 = 8$ clauses. Similarly, describing a XOR-chain with n variables (n is even) needs

$$\binom{n}{0} + \binom{n}{2} + \cdots + \binom{n}{n} = 2^{n-1}$$

clauses. Therefore, the concept of cutting makes sense. We could cut the number of variables XORed in our formula above as $e+a+b = 0, f+c+e = 0, g+d+f = 0$ introducing new variables e, f, and g. So one needs 4 clauses for each of the three equations. The cutting number [15] defines, how many variables are in one XOR-chain. In the example given, it is 3. Usually cutting numbers 4–7 are used.

In our experiments we call this conversion technique `BCJ`. More details on this technique may be found in [15]. The convertor is implemented in SAGE computer algebra system [18] by Martin Albrecht and Mate Soos [19].

Conversion based on the truth table

Whereas in the `BCJ` method the number of variables in a logical formula is usually quite large compared to the initial number of variables in the ANF of a polynomial, the following method preserves this number. The method, which we call `TruthTable` in this paper, is based on writing a CNF for the given Boolean function by examining the evaluation (truth) table of the function. We work with a specific implementation of the method described in [16] and implemented by Michael Brickenstein [20]. The core of the method dates back to Karnaugh [21] and is also used in cryptographic context in [7]. Note that the problem of long XOR chains also appears here. So one might want to apply the cutting technique first and then the truth table method for resulting shorter polynomials.

As an example, the polynomial $f = xyz + xy + yz + xz$ from above with the method of [16] has the CNF

$$(x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z).$$

So contains only 3 clauses and 3 variables.

5 Optimal tools and strategies for the attacks

5.1 Agenda

In this section we prepare for algebraic attacks on PRINTCipher-48. We use SAT techniques as a solving tool. Since algebraic cryptanalysis is a lot about trying out certain heuristics, we would like to cover as many of them to find the best one. In order to understand what we are going to do, we formulate some questions, which we answer in order to get an optimal representation and solving strategy. The first block of questions concerns the algebraic representation and the ANF to CNF conversion. First, assume that the key sk_2 is known or guessed.

– Assuming we know sk_2, is there a difference whether $sk_2 = 0$ or $sk_2 \in_R \mathbb{F}_2^{32}$?
– Is there a difference in solving $Sys(p, c)$ for the correct value of sk_2 and a wrong one?
– Assuming sk_2 is given, is it better to use explicit equations for the S-Boxes ($Y = SBOX(X)$ or $X = SBOX^{-1}(Y)$) or all the quadratic equations that describe an S-Box?
– Which ANF-to-CNF conversion method yields better results: `TruthTable` or `BCJ`?

This questions are addressed in Section 5.2.

It is known that in order to be able to solve a large system of equations coming from a block cipher, it is a good practice to first guess some key bits and then try to solve easier systems. In particular, we solve one easier system if we assume that we are given the correct key bits. In the cryptanalytic context this means that we need to make sure that if we are given (or guessed) g out of k key bits, the resulting system solving is faster than the brute force of the remaining key space of size 2^{k-g}. See Section 6.1 for more details. Since PRINTCipher does not have a key schedule, i.e. the same key is used in all rounds of the cipher, it is particularly plausible to apply this approach. Moreover, by guessing enough key bits, we are able to bring our timings to some feasible figures. This in turn will give us an opportunity to make estimations of the attack complexity. So, guessing key bits is important. But now we have to answer certain questions in order to understand where and how to guess bits optimally.

– Since the bits in a round are grouped in groups of three bits before applying the S-Box transformations, does it make a difference to guess the key bits in such groups of three or random positions are as good?
– Does it make a difference which groups/positions do we choose for guessing?
– Is it better to first guess the key sk_2 and then some parts of the key sk_1 or one should guess parts of these keys?
– Related to the question above, latter part: should guessing in the keys sk_1 and sk_2 be synchronized (that is, for example, locations of chosen groups of three for sk_1 match guessing the corresponding key-permutation bits of sk_2) or the guesses may be done independently?

These questions are treated in Section 5.3.

We would like to investigate the effect of using more plaintext/ciphertext pairs, e.g. 4 or 8, see Section 5.4.

Finally, in Section 5.5 we address the question, which solver is better for our purposes: MiniSAT2 or CryptoMiniSAT2.

Computing environment: We do our computations on a server with 4 AMD processors each having 4 cores working at 2.3 GHz. The computations are run using one core. The server has 128 GB of RAM and is running under Debian Linux version 4.3.5-4 Linux kernel version 2.6.32-5-amd64. For our computations we use SAGE version 4.3.3, MiniSAT version 2.0 beta and CryptoMiniSAT version 2.9.0.

Experimental setting: For estimating time we compute the average time for 100 trial runs. The time in tables is given in seconds, unless is indicated otherwise. In Sections 5.2–5.4 we use MiniSAT.

5.2 Determining optimal representation and conversion

Now we want to answer the first block of questions from the previous subsection. Namely, which algebraic representation is better and which ANF-to-CNF conversion technique yields better results. Since finding an 80-bit key $K = (sk_1, sk_2)$ is out of reach for our methods, we have to assume that certain parts of the key are given or guessed. We prefer to first fix/guess the subkey sk_2, since then we turn out to be in the "classical" situation of algebraic cryptanalysis where the S-Boxes are fixed. We consider different guessing scenarios in the follow-up Section 5.3.

First we assume $sk_2 = 0$, i.e. all S-Boxes are fixed to V_0. We investigate which S-Box equations are better to take for solving and also how the two conversion techniques perform. In these experiments we attack 5 rounds of PRINTCipher-48 with $sk_2 = 0$. We set the cutting number of `BCJ` (see Section 4.1) to be equal to 4, since the describing equations are quite short anyway. The timings in

this and all the following experiments are in seconds. Table 2 summarizes our experiments. From this table we may conclude that at least taking all quadratic equations that describe the S-Box V_0 is not a good idea. Especially the `BCJ` method has a problem with that. The other two choices do not differ significantly. Also both conversion techniques behave pretty much the same for these choices.

Table 2. Attacking 5 rounds with $sk_2 = 0$.

Conversion	Eqs.	Time
`TruthTable`	$Y = SBOX(X)$	175
`TruthTable`	$X = SBOX^{-1}(Y)$	227
`TruthTable`	All	368
`BCJ`	$Y = SBOX(X)$	182
`BCJ`	$X = SBOX^{-1}(Y)$	174
`BCJ`	All	734

Next we try to see whether some specific choice of sk_2, e.g. $sk_2 = 0$ as above, is any different from a random choice. For a good cipher this should not be the case. Table 3 shows the results. In the experiments a random and correct values of sk_2 are used for each experiment. We work only with the explicit equations for the S-Boxes as suggested by the previous experiment. The table suggests no significant difference in attacking $sk = 0$ and sk_2 random. Moreover using equations $Y = SBOX(X)$ and $X = SBOX^{-1}(Y)$ is not significantly different.

Table 3. Attacking 5 rounds with sk_2 randomly chosen.

Conversion	Eqs.	Time, s.
`TruthTable`	$Y = SBOX(X)$	165
`TruthTable`	$X = SBOX^{-1}(Y)$	174
`BCJ`	$Y = SBOX(X)$	150
`BCJ`	$X = SBOX^{-1}(Y)$	166

In the previous experiments we assumed correct values for sk_2. For an efficient attack it is important to know what is the behavior for wrong guesses, since they constitute the majority of guesses (in fact we expect only one guess to be the correct one). Table 4 shows the experimental results. For each experiment trial we took a random incorrect guess for sk_2. This means that we take wrong S-Box equations. Again we consider both conversion techniques and explicit equations for the S-Boxes. Comparing these results with the results from Table 3 or Table 2 we see that a

Table 4. Attacking 5 rounds with sk_2 wrongly guessed.

Conversion	Eqs.	Time, s.
`TruthTable`	$Y = SBOX(X)$	354
`TruthTable`	$X = SBOX^{-1}(Y)$	379
`BCJ`	$Y = SBOX(X)$	370
`BCJ`	$X = SBOX^{-1}(Y)$	411

wrong guess results in a slow-down approximately by a factor of 2. We will see further that such a behavior is quite typical in out setting.

Remark 1. It is interesting to note that equations of the type $X = SBOX^{-1}(Y)$ do not yield better results. This is despite the fact that by having sk_2 fixed we essentially eliminate the last round and moreover the equations $X = SBOX^{-1}(Y)$ give immediately values of the variables of the last round.

As a result of this subsection we have only eliminated the usage of all the S-Box equations. All other experimental differences are not significant enough. We continue using the two conversion techniques in further subsections.

5.3 Determining optimal guessing strategy

Now we try to increase the number of rounds we can attack in a reasonable time. For this we need to guess more than 32 bits of the key. We first proceed with the approach of the previous section, where sk_2 is fixed and try to guess some bits from sk_1. As has been noted in Section 5.1 there are several ways how one can guess the bits of sk_1. First we consider guessing at random positions from $\{0, \ldots, 47\}$. Then we consider guessing at random *groups* of positions, where the groups are the groups of 3, which corresponds to the S-Box size. So we guess at S-Box positions $\{0, \ldots, 15\}$. Finally we guess at specific positions that correspond to the low diffusion trails, see Table 1. Table 5 summarizes the experimental results. Here we attack $r = 7$ rounds by guessing correctly $g = 12$ bits in the key sk_1. We take equations $Y = SBOX(X)$. The results indicate that it is a good choice to guess in groups of 3 bits, moreover, at those positions suggested by the low diffusion trail. Note that for $g = 12$ we use the trail $\{0, 1, 5, 15\}$, see Table 1.

Table 5. Attacking 7 rounds with 12 bits of sk_1 guessed.

Conversion	Guess	Time, s.
`TruthTable`	`RandomPos`	147
`TruthTable`	`RandomGroup`	86
`TruthTable`	`Trail`	46
`BCJ`	`RandomPos`	119
`BCJ`	`RandomGroup`	91
`BCJ`	`Trail`	56

To shed more light on using the trail strategy we provide another experiment that attacks $r = 8$ by guessing $g = 15$ variables at the trail $\{4, 10, 12, 14, 15\}$. This time we also take equations $X = SBOX^{-1}(Y)$. Results are in Table 6. The experiment suggests a slight advantage of using `TruthTable` with equations of the type $Y = SBOX(X)$.

Table 6. Attacking 8 rounds with 15 bits of sk_1 guessed.

Conversion	Eqs.	Time, s.
`TruthTable`	$Y = SBOX(X)$	114
`TruthTable`	$X = SBOX^{-1}(Y)$	141
`BCJ`	$Y = SBOX(X)$	151
`BCJ`	$X = SBOX^{-1}(Y)$	148

It is time now to look at the situation when we do not guess the entire sk_2, but rather use the full algebraic representation from Section 3 to guess parts of both sk_1 and sk_2. We use explicit equations with $W-$variables to describe S-Boxes. Other possibilities, e.g. eliminating $W-$variables (Appendix A.2) or using implicit equations of degree at most 2 (Appendix A.3) did not yield any

significant improvement. First, similarly to the above, we may try to guess independently at positions of sk_1 and sk_2 or at the grouped positions. The question is how many bits do we want to guess in each key part. In our experiments we guess $3x$ and $2x$ in sk_1 and sk_2 resp., where x is a guessing parameter. Therewith the guessed parts are proportional to the sizes of sk_1 and sk_2. We have then that $g = 5x$. Recall that for the full representation we need two plaintext/ciphertext pairs to uniquely determine the key. In the experiments we take two pairs coming from two random plaintexts. So we are in the known plaintext scenario. Table 7 gives the results. In the experiment we attack 5 rounds with $x = 6$, i.e. we guess overall $g = 30$ bits: 18 in sk_1 and 12 in sk_2. The guesses are correct guesses. The results do not suggest the difference in the two approaches (as opposed to the case when the entire sk_2 is fixed, see Table 5). Nevertheless now they suggest that using `TruthTable` is better than using `BCJ` with the cutting number 4. This makes sense, since equations in the full algebraic representation are longer than those with sk_2 fixed. We come back to this issue a bit later.

Table 7. Attacking 5 rounds with overall 30 bits guessed in sk_1 and sk_2.

Conversion	Grouping	Time, s.
`TruthTable`	`RandomPos`	8
`TruthTable`	`RandomGroup`	5
`BCJ`	`RandomPos`	27
`BCJ`	`RandomGroup`	28

In the previous experiment when guessing in groups, the groups were not "agreed". That is, the three bits from the sk_1-part do not necessarily match the guess of the two bits in sk_2 that are responsible for a corresponding S-Box. Our next experiment implements the agreeing strategy and compares it with the one that does not use agreeing. In the experiment we use $r = 6$ and $x = 7$. Table 8 presents the results. Now we clearly see that the agreeing strategy yields better results for both conversion methods. We again see that the `TruthTable` is superior.

Table 8. Attacking 6 rounds with 35 bits guessed in sk_1 and sk_2.

Conversion	Grouping	Time, s.
`TruthTable`	`RandomGroup`	74
`TruthTable`	`RandomAgree`	25
`BCJ`	`RandomGroup`	490
`BCJ`	`RandomAgree`	144

As we have mentioned above, the choice of the cutting number being equal to 4 may not be the optimal one. Taking the same setting of $r = 6$ and $x = 7$ and the strategy used being the one with agreeing, we investigate the effect of using different cutting numbers. The results of this experiment are in Table 9. As we see the experiment suggests a slight advantage for the choice of cutting number being equal to 6.

Next, again similarly to the experiments with sk_2 fixed, we try the strategy that not only uses agreed guessing, but also guessing in accordance with the low diffusion trails. We compare the result for `TruthTable` from the previous experiment with the one using the trail strategy. The used trail is $\{3, 8, 9, 10, 11, 13, 15\}$. Table 10 shows the results. The experiment clearly indicates the superiority of the trail strategy.

Now we would like to get back to the question of correct vs. incorrect guessing. We apply the trail strategy and investigate what is the effect of a guess being incorrect. Note that now we guess incorrectly both at the bits of sk_1 and also at the S-Boxes used, i.e. bits of sk_2. In the experiment

Table 9. Trying different cutting numbers for the BCJ method.

`cut_num`	Time, s.
4	144
5	143
6	118
7	146

Table 10. Guessing at grouped positions from the trail.

Grouping	Time, s.
`RandomAgree`	25
`Trail`	2

we guess incorrectly randomly at each trial. For this we consider a slightly harder instance of $r = 6$ and $x = 6$. We consider both conversion techniques with `BCJ` using the cutting number 6. Table 11 illustrates the experiment. The table suggests two things we have already seen before: a wrong guess takes approximately twice as much time to be detected and `TruthTable` is better than `BCJ` for the full algebraic representation.

Table 11. Wrong guesses of bits in sk_1 and sk_2.

Conversion	Guess	Time, s.
`TruthTable`	Correct	37
`TruthTable`	Wrong	71
`BCJ`	Correct	368
`BCJ`	Wrong	654

Table 12 illustrates parameters of the systems solved that are important for understanding performance of SAT-solving. In this table we list the number of variables and the number of clauses for one system using different conversion techniques from Section 4.1. For the `TruthTable` conversion we had to average the number of clauses, since it varies slightly, depending on the guess at key-permutation variables. A number after `BCJ` indicates the cutting number used. The table suggests that `TruthTable` conversion yields smaller SAT-problems both in terms of the number of variables and the number of clauses, which reflects accordingly on the performance, see Table 11.

As a result of this subsection we identified that it is a good strategy to guess key bits in groups, moreover the guessing should be done according to a low-diffusion trail. Then we identified that `TruthTable` is superior to `BCJ` for the full algebraic representation, whereas for the case of sk_2 fixed we did not observe a significant difference.

5.4 Effect of using more pairs

Until now we used one plaintext/ciphertext pair for the case of fixed sk_2 and two pairs otherwise (to uniquely determine the key $K = (sk_1, sk_2)$). In cryptanalysis having more pairs of plaintext/ciphertext usually helps to speed up an attack and/or reduce its complexity. In algebraic cryptanalysis having more pairs leads to more overdetermined systems that are usually easier to solve. On the other hand, one of the advantages of algebraic cryptanalysis over statistical cryptanalysis (e.g. differential, linear etc.) is the ability to work, at least theoretically, already with just a few pairs. Moreover, having too many pairs makes systems in question too large to be processed

Table 12. Sizes of SAT-problems with different conversion techniques.

Conversion	# vars	# clauses
`TruthTable`	1778	8627
`BCJ-4`	4303	18581
`BCJ-6`	3263	23341
`BCJ-7`	3195	26189

with modern algebraic solvers. Having this in mind, we want to see what is the effect of adding more plaintext/ciphertext pairs, but limiting the number of pairs to just a few.

In this subsection we use the `TruthTable` conversion, since as suggested by the previous subsections, it is always not worse that the `BCJ` and is often superior. For the first experiment we choose to attack 9 rounds and sk_2 is not fixed. Key bits from sk_1 and sk_2 are always guessed in agreed groups according to a low diffusion trail. In Table 13 we present results of the experiment. We guess $g = 50$ bits of the key: $x = 10$. We compare using 2 pairs against using 4 pairs. Moreover we investigate an effect of incorrect bit guessing. The table suggest that using more pairs does not really help in this situation.

Table 13. Working with more pairs: 2 vs. 4.

# Pairs	Guess	Time, s.
2	Correct	19
4	Correct	37
2	Wrong	54
4	Wrong	63

Next we consider the situation, when the key sk_2 is fixed/guessed. We attack 8 rounds, guess 15 bits of sk_1 in groups according to low diffusion trail, use 2,4, and 8 pairs, and consider wrong guesses as well. Table 14 presents the results. As the table suggests, we now have an advantage of using more pairs. The situation with wrong guesses is the same as before: we are approximately factor 2 slower.

Table 14. Working with more pairs: 2,4,8 and sk_2 is fixed.

# Pairs	Guess	Time, s.
2	Correct	105
4	Correct	84
8	Correct	67
2	Wrong	221
4	Wrong	182
8	Wrong	128

5.5 MiniSAT2 vs. CryptoMiniSAT2

Until now we have done our computations with the help of MiniSAT2 solver. Another solver, CryptoMiniSAT2, has been proposed [14] to tackle problems more specific to cryptography. In particular, this solver is supposed to handle long XOR chains better, which, as we know from

Section 4.1, pose a problem for SAT solvers. In our experiments we do not have too long XOR chains. Still it is worthwhile to investigate performance of this solver in our context, to see an effect of heuristics employed there. For the comparison we took partially our targets from the previous subsection. Namely we consider first the case, when sk_2 is not fixed. There we take 2 and 4 plaintext/ciphertext pairs: both known and chosen. We do both correct and incorrect guessing. Using notation as before we attack $r = 9, x = 10$. Table 15 contains the results. For convenience we also put there corresponding results for MiniSAT2. In the table "CMS" stands for CryptoMiniSAT2 and "MS" for MiniSAT2. From the table we observe that CryptoMiniSAT2 has the same tendency to solve instances with wrong guesses slower. Still CryptoMiniSAT2 seems to be somewhat better at this than MiniSAT2. Other than that, performance of both solvers appears to be comparable.

Table 15. CryptoMiniSAT2 vs. MiniSAT2.

# Pairs	Guess	Plaintexts	Time for CMS, s.	Time for MS, s.
2	Correct	Known	20	19
4	Correct	Known	21	37
2	Wrong	Known	46	54
4	Wrong	Known	39	63

Next we analyze the situation, when sk_2 is fixed. We take 8 rounds, 8 pairs, 15 bits are guessed, and both correct and wrong guesses are done. We use S-Box equations of the type $Y = SBOX(X)$. Table 16 summarizes our computations and compares them with those of MiniSAT2 from the previous subsection, see Table 14. From the table we see that in this case CryptoMiniSAT2 quite significantly outperforms MiniSAT2.

Table 16. CryptoMiniSAT2 vs. MiniSAT2. sk_2 is fixed.

Guess	Plaintexts	Time for CMS, s.	Time for MS, s.
Correct	Known	27	67
Wrong	Known	59	128

6 Algebraic analysis of PRINTCipher-48

In this section we apply strategies developed in Section 5 for specific attacks on PRINTCipher-48. We consider attacking round-reduced versions of the cipher, as well as simulation side channel attack on the full cipher. Now we specify our heuristic for the algebraic attacks.

- We always guess (or assume given) key bits at positions grouped in groups of three, according to S-Box positions. See Section 5.2.
- In the case that sk_2 is not fixed, guessing bits in sk_2 is agreed with guessing in sk_1. See Section 5.3.
- Grouped positions correspond to low diffusion trails. See Sections 5.2 and 5.3.
- In the case sk_2 is fixed, we use equations $Y = SBOX(X)$ for the S-Boxes.
- We use both MiniSAT2 and CryptoMiniSAT2 when sk_2 is not fixed and CryptoMiniSAT2 when it is.
- `TruthTable` conversion is used.
- We use 100 samples to measure the average time.

6.1 Attack on round-reduced PRINTCipher-48

In this subsection, basing on our experiences from Section 5, we attack round-reduced instances of PRINTCipher-48. First of all, we have to define what do we mean by an attack here and when such an attack succeeds. In all our attacks a certain amount of key bits will be fixed of guessed. We want to compare our method with the brute force attack. Similarly to previous works in the field of algebraic cryptanalysis of block ciphers, e.g. [10,17], we employ the following way of comparing the methods. First, let us assume that g bits out of 80 are given to an attacker, i.e. he/she knows correct values of those bits. Let $T_{correct}$ be an average time an attacker needs to solve a True-instance using methodologies from Section 5. Now let $T_{enc,r}$ be the time that one needs to check with a trial encryption one key from the remaining 2^{80-g} in a reduced cipher with r rounds. We assume that for such an encryption one needs 2 CPU cycles per round. To translate this figure to seconds, we observe that our experiments are done on a machine with the CPU frequency 2.3 GHz. So we may assume that 1 CPU cycle is done in $\frac{1}{2.3} \cdot 10^{-9} \approx 0.4 \cdot 10^{-9}$ seconds. We have then $T_{enc,r} = 2 \cdot 0.4 \cdot r$. Note that this is a rather optimistic estimate.[1] The time complexity for a brute force attacker is therefore

$$T_{bf,r} = 2^{80-g} T_{enc,r}/2 = 2^{80-g} \cdot 0.4r \cdot 10^{-9}$$

seconds. We divide over 2, since on average an attacker has to search only through a half of the remaining key space. [2] For a successful algebraic attack we require that

$$T_{correct} < 2^{80-g} \cdot 0.4r \cdot 10^{-9}. \tag{1}$$

Now assume that bits of the key are not given. So we have to guess some g key bits. As we have seen, the time needed to solve a False-instance, i.e. the one with an incorrect key guess, is higher than the one for a correct guess. Therefore, in this case our time complexity is determined by the average time needed to solve a False-instance: T_{wrong}. Assuming that all False-instances are approximately the same in difficulty, we have that the attack is successful if

$$T_{wrong} < 2T_{bf,r} = 2^{80-g} \cdot 0.8r \cdot 10^{-9}. \tag{2}$$

Since both methods run into a correct solution on average two times faster than the full search, we put back the factor 2, taken away in (1). For both algebraic and the brute force method, the memory complexity is quite low, although being higher for the former one. Still, memory is not a bottle neck for SAT-solving, so we ignore it in our analysis.

In the case when sk_2 is given or guessed the formulas above stay the same, except that we change 2^{80-g} to 2^{48-g}, since a brute force attacker has to search only through the key space of sk_1, which is 48 bits. For instance, (2) becomes

$$T^{sk2}_{wrong} < 2^{48-g} \cdot 0.8r \cdot 10^{-9}. \tag{3}$$

Our first result: **we are able to attack 8 rounds of PRINTCipher-48 with 2 known plaintext/ciphertext pairs faster than the brute force**. We achieve this by applying our algebraic attack for the case when sk_2 is not fixed. We guess at $g = 45$ bits of the key, i.e. we use $x = 9$. We take 2 random plaintexts for each of the 100 trials. Table 17 summarizes the results.

In order to see that we are really faster than the brute force, in Table 18 we provide the figures of LHS and RHS of (2) for $g = 45$. Here timings for (Crypto)MiniSAT2 correspond to False-instances. As we noted, it is necessary to assess time complexity of the attack. Both solvers are doing the job.

[1] We do not take into account other possibilities to speed up the brute force attacks, such as using FPGAs or GPUs.

[2] In fact for a portion of 2^{-32} keys one needs to do the second encryption, since 2^{32} out of 2^{80} keys are expected to encrypt a given plaintext to a given ciphertext. Due to negligible probability of this event, we do not take this into account.

Table 17. Attack on 8 rounds, 2 known plaintexts.

Guess	Time for CMS	Time for MS
Correct	24	30
Wrong	61	60

Table 18. Comparing to brute force.

	Time for CMS	Time for MS	Time for brute force
$s = 45$	61	60	239

Now we are interested to see whether the method of fixing sk_2 is better for attacking the same target. Below we show that this is not the case. We take the best heuristics from the previous subsections: CryptoMiniSAT2 as a solver, 8 known pairs, guessing 12,15, or 18 bits. As Table 19 suggests only when 15 bits are guessed we are slightly better than the brute force. Still taking 4 pairs results in the average time 118 sec. and that is worse than brute force. This means that we cannot outperform the previous method.

Table 19. Attack on 8 rounds, 8 known plaintexts, sk_2 is fixed.

# guessed bits	Time for CMS	Time for brute force
12	694	478
15	55	60
18	9.0	7.5

We could not entirely break the 9 rounds, but still we were very close and could achieve partial success. When doing experiments for 9 rounds we took a look at several low diffusion trails for $x = 10$. It turns out that some of them are better than the others. Table 20 summarizes experimental results for 9 rounds made with CryptoMiniSAT2. So we see that the trail $\{0, 1, 3, 4, 5, 10, 11, 12, 14, 15\}$ yields the best results. Still we are not able to get below the threshold for both False- and True- instances.

Nevertheless, for True-instances we may ask a question, what is the portion of measurements that fall below the brute force time. Table 21 shows these figures for all four trails. We see that with probability around 3/4 we are able to solve a True-instance faster than the brute force. Therewith, **having 2 known plaintext/ciphertext pairs and 50 key bits at nibble positions $\{0, 1, 3, 4, 5, 10, 11, 12, 14, 15\}$, we may find remaining 30 bits faster than brute force with probability around 75%.**

Remark 2. In fact when using low diffusion trails, as we have seen, only looking at how many bits stay within a trail is not enough. Actually, one should consider, how many bits one gets "for free" when having key bits corresponding to the trail nibble positions, plaintext and ciphertext bits. Here one can observe that, obviously, 3 known input/output bits to an S-Box yield 3 output/input bits to the previous or next rounds. Less obviously it is possible to show that knowing 2 input/output bits yields knowledge of on average 1 output/input bit.

Having these observation, one could improve the guessing strategy, e.g. as follows. Since knowing 2 bits yields 1 on average and knowing 3 yields 3, then if one knows e.g. 2 input bits to an S-Box it may be worthwhile to guess at the remaining bit. Therewith, a 2-fold slow-down due to guessing will be payed off by a 4-fold speed-up due to knowing 2 additional internal state bits. This should result in a speed up of a factor of 2.

Table 20. Attack on 9 rounds, 2 known plaintexts, different trails

Trail	Guess	Time	Time for brute force
$\{0,1,2,3,4,5,6,7,11,12\}$	Correct	19.5	4.2
$\{0,1,2,3,4,5,6,7,11,12\}$	Wrong	43.9	8.4
$\{0,1,3,4,5,10,11,12,14,15\}$	Correct	5.8	4.2
$\{0,1,3,4,5,10,11,12,14,15\}$	Wrong	16.3	8.4
$\{2,3,4,6,7,8,9,11,12,13\}$	Correct	14.3	4.2
$\{2,3,4,6,7,8,9,11,12,13\}$	Wrong	25.2	8.4
$\{3,4,8,9,10,11,12,13,14,15\}$	Correct	34.4	4.2
$\{3,4,8,9,10,11,12,13,14,15\}$	Wrong	56.3	8.4

Table 21. Attack on 9 rounds, 2 known plaintexts, percentage of runs faster than the brute force.

Trail	%
$\{0,1,2,3,4,5,6,7,11,12\}$	22
$\{0,1,3,4,5,10,11,12,14,15\}$	78
$\{2,3,4,6,7,8,9,11,12,13\}$	56
$\{3,4,8,9,10,11,12,13,14,15\}$	30

All in all, one may investigate how known internal state bits propagate to yield more known bits, maybe with some probability. We do not elaborate on this issue in this paper, although this is something to consider and that may have impact for other methods, like statistical saturation attacks.

6.2 Additional bits at round four

In the previous subsection we presented in detail our approach of attacking the PRINTCipher-48 based on SAT-solving. We could attack 8–9 rounds with it. In this subsection we show that it is possible to attack the full 48 rounds of the cipher by just having moderate amount of additional information. We are able to do so, after discovering quite an interesting property of the cipher. Namely, by just knowing values of output bits at round 4 of several known plaintext/ciphertext pairs and some key bits, it is possible to recover all remaining key bits fast. In this subsection we proceed as follows. We first provide results on attacking the full 48 rounds of the cipher with guessing bits at rounds 4 and 5. Then we try to explain, what makes round 4 so special by providing additional data and observations.

In this section we use CryptoMiniSAT2 for our experiments. Note that for the analysis of the full 48 rounds we adjusted the round counter of the cipher, so that the attack of [5] that works for many weak keys, does not work anymore, cf. Section 2.3 of [5]. In fact, this adjustment does not play any role for our attacks. It will become clear why, after we look deeply into the situation of using the 4th round for guessing.

Table 22 summarizes the results. As before, we guess key bits and internal state state bits at rounds 4 and 5 using low diffusion trails. Using the previous notation, we use parameter x, meaning how many nibbles are fixed in a trail, so that we guess at $5x$ key bits and $3xp$ intermediate state bits, where p is the number of known plaintext/ciphertext pairs.

One of the conclusions that may be drawn from the table is that assuming 6 output bits of round 4 known for 10/12 pairs and knowing only 10 key bits, the remaining 70 key bits can be recovered very fas[3]. The same is true for 21 bits at round 4 for only 2 pairs and knowing 35 key

[3] Here the following trick turned out to be particularly useful. We used "forward" explicit equations $W = SP(X, A), Y = V_0(W)$ for the first 24 rounds and "reverse" equations $X = SP^{-1}(W, A), W = V_0^{-1}(Y)$ for the second half of the rounds. This trick does not seem to have a significant impact for other instances we considered.

Table 22. Recovering key bits by having some output bits at round 4 and 5. Two S-Box positions are fixed.

r	p	x	Correct guess	Time
4	10	2	True	116
4	10	2	Wrong	62
4	12	2	True	10
4	12	2	Wrong	8
4	2	7	True	1.7
4	2	7	Wrong	0.4
4	2	6	True	295
4	2	6	Wrong	75
5	2	7	True	10
5	2	7	Wrong	11
5	2	6	True	934
5	2	6	Wrong	1426

bits: remaining 45 key bits are recovered in a matter of a second. We are still able to solve quite fast by having 18 bits at round 4 for 2 pairs. We could get some results also by inserting values at round 5, but the performance appears to be inferior to what we can get from round 4.

Knowing internal key bits is a strong assumption. Still, it may be feasible if one thinks about side channel security. See Section 6.3 for this. Some thoughts on using the idea of this section to a potential attack on the full cipher are given in Section 6.4. Note also that Table 22 suggests that False-instances are solved faster for $r = 4$, which is different from results of the previous sections. However, variance of these computations is also larger than for the previous ones.

Now we want to shed some light on why guessing/fixing at round 4 is so effective; why cannot we do the same for, e.g., rounds 3 or 5 with comparable efficiency. The main reason appears to be the following. Let us consider the variety $V(r, p, C)$ of keys that after r rounds encrypt p known plaintexts to some ciphertexts that have given values at positions from C ($C \subset \{0, \ldots, 47\}$). It turns out that for $r = 3$ this variety is rather large due to incomplete diffusion (recall that "full" diffusion for PRINTCipher-48 is attained at 4 rounds). On the other hand, from round 4 and on, this variety drastically looses in size. This means that having enough pairs p we cut off many keys starting already at round 4. The second issue is how easy is it to sample elements from $V(r, p, C)$. For $r = 3$ this is extremely easy (fast), for $r = 4$ is somewhat slower, but still quite fast. Starting from round 5, it gets harder, at least in some cases of interest. Therefore, roughly speaking, round 4 yields an optimal trade-off between the number of elements in the variety $V(r, p, C)$ and hardness of computing its elements.

In principle, one may run the following attack for the case one knows internal state bits at round r at positions C. Choose $r < 48$. Let $t(r, p, C)$ be the (average) time needed to compute one element from $V(r, p, C)$. Then one may simply run an exhaustive search on $V(r, p, C)$ and this will have time complexity approximately

$$(t(r, p, C) + T_{enc,48}) \cdot |V(r, p, C)|,$$

where $T_{enc,48} \ll t_(r, p, C)$ is the time needed for one trial encryption to check the correctness of the key guess for the full 48 rounds using p known plaintext/ciphertext pairs. Therefore, finding a trade-off as explained above is important. The elements of $V(r, p, C)$ may be sampled again using SAT-solving techniques in assumption that one gets independent random solutions from $V(r, p, C)$ each time. Considering highly heuristic and randomized nature of SAT-solvers this assumption does not appear unreasonable. Results from Table 22 are obtained by just assigning values at round 4 and running CryptoMiniSAT2 for the obtained systems having 48 rounds. Still we expect the solver to take advantage of the above trade-off: inserted values at round 4 give an opportunity to narrow down the search a SAT-solver takes.

We support the above reasoning with some experimental data. First we give some experimental values for $t(r, p, C)$ for different r, p and C. Table 23 summarizes the results. Abusing notation, we use nibble positions and not the bit positions. Positions, as usual, correspond to low diffusion trails. The table confirms the claims made before, concerning values of $t(r, p, C)$. We observe quite fast timings for $r = 5, x = 6 - 7$, though.

Table 23. Time $t(r, p, C)$.

r	p	C	$t(r, p, C)$
3	8	$\{0, 1\}$	0.06
3	10	$\{0, 1\}$	0.09
3	12	$\{0, 1\}$	0.12
3	16	$\{0, 1\}$	0.15
4	8	$\{0, 1\}$	0.31
4	10	$\{0, 1\}$	1.84
4	12	$\{0, 1\}$	3.38
4	16	$\{0, 1\}$	1.22
5	8	$\{0, 1\}$	4813
3	2	$\{0, 1, 2, 5, 6, 7\}$	< 0.02
4	2	$\{0, 1, 2, 5, 6, 7\}$	0.03
5	2	$\{0, 1, 2, 5, 6, 7\}$	0.25
3	2	$\{3, 8, 9, 10, 11, 13, 15\}$	< 0.02
4	2	$\{3, 8, 9, 10, 11, 13, 15\}$	0.04
5	2	$\{3, 8, 9, 10, 11, 13, 15\}$	1.42

A more complicated task is to estimate the number of elements in $V(r, p, C)$. One of the techniques ([22,17]) is estimating size of a variety by adding random linear relations on variables. The hope is that a random linear relation cuts approximately a half of the variety in question. Therefore, if adding s random linear relations yields a solution (a satisfying assignment of variables) and $s+1$ does not, then with certain probability we may claim that the variety size is 2^s. We applied this approach to estimate the values of $V(r, p, C)$. We added random XOR-chains that contain only key variables suggested by $\{0, \ldots, 48\} \setminus C$ (or $\{0, \ldots, 15\} \setminus C$ in the nibble notation), since these determine the rest of the variables. Also, since adding long XOR chains obstructs a SAT-solver from getting a solution in reasonable time, we have to work with chains of certain length. We chose length 7 for our experiments. The results are summarized in Table 24[4]. The numbers are obtained after averaging over 100 trials. As the table suggests, we get a serious reduction in the variety size when going from $r = 3$ to $r = 4$. We must note, however, that the method used above is not very accurate. By trying out sampling solutions from $V(r, p, C)$ as described earlier in the subsection, we noticed that figures in Table 24 are underestimations of the actual values. This is especially true for the case $r = 3$. This may be explained by observing that since we do not yet have full diffusion after 3 rounds, the systems we consider are less random than those for $r \geq 4$. Therefore, the variety $V(3, p, C)$ is not that "homogeneous" and XOR-relations may stop the count too early. Still, we believe that these experimental results provide rough intuition and the basis for our claims above.

We would like to make an interesting observation from the results of this section. It might not be a good idea to attain full diffusion in a cipher early. As we have seen, is such a case by fixing some internal state bits, we may cut off dramatically the searched key space. On the other hand, the fact that full diffusion happened early, suggests that algebraic complexity of the reduced key space is rather low, so that one may be able to sample elements of this space efficiently.

[4] "N/A" means we were not able to get any results in a reasonable time.

Table 24. Variety size $|V(r, p, C)|$.

r	p	C	$\log_2 \|V(r, p, C)\|$
3	10	$\{0, 1\}$	25
4	10	$\{0, 1\}$	11
5	10	$\{0, 1\}$	N/A
3	2	$\{3, 8, 9, 10, 11, 13, 15\}$	16
4	2	$\{3, 8, 9, 10, 11, 13, 15\}$	8
5	2	$\{3, 8, 9, 10, 11, 13, 15\}$	4

6.3 Side channel analysis of the full PRINTCipher-48

Using results of the previous subsection, we can show that it is possible to break the full PRINTCipher-48 with practical complexity by using side channel analysis. We first take a strong assumption of knowing internal state bits at round 4, namely some output bits of round 4. Then we consider more usual scenario in side channel analysis, when only Hamming weights of inputs and outputs of certain S-Boxes at round 4 are known. We show that due to properties of S-Boxes in PRINTCipher, we are still able to do our attack with a moderate increase in complexity.

Assuming knowledge of some information about internal states of a cipher we end up in the side channel scenario. That is, we assume that we can recover certain internal bits, which in principle can be done by examining an implementation of cipher's encryption function. Note that actual recovering of bits may be a very challenging task. In this section we only consider a simulation side channel attack, not an actual one, i.e. we simply assume some internal bits to be given.

From the point of view of side channel analysis, it seems reasonable to concentrate on the case, when as few nibbles as possible are spotted for input/output values. In this respect, using $x = 2$ with $C = \{0, 1\}$ seems appropriate. Note that in this case we are guessing at $5x = 10$ bits of the key: 6 at the key sk_1 and 4 at the key sk_2. In Table 22 we presented timings for recovering remaining 70 key bits. We also considered the situation, when our key guess was wrong and how much time does it take to realize this. As has been noted, solving False-instances, i.e. when key guess and guess of internal state bits is wrong is actually somewhat faster than solving a True-instance. Since wrong guesses constitute the majority in the search through 10 key bits, we take the time of solving False-instances, T_{wrong}, as a measurement. We multiply the values from Table 22 by 2^9 (i.e. the half of 2^{10} on average) to get an estimated average time for the full key recovery for the entire cipher with 48 rounds. The results are in Table 25.

Table 25. Side channel attack assuming knowledge of internal state bits at round 4: 6 bits for each of the p pairs are known

p	T_{wrong}, sec.	Total Time, hours
10	62	8.8
12	8	1.2

Therefore, **by having values of** 6 **output bits at round 4 that correspond to the nibble positions** $\{0, 1\}$**, we are able to recover the key on average in approximately 9 hours by using 10 known pairs of plaintext/ciphertext and in less than 2 hours with 12 pairs.** Note that we need knowledge of $6 \cdot 10 = 60$ and $6 \cdot 12 = 72$ internal bits respectively.

A weaker assumption we take next is the knowledge of Hamming weights instead of exact knowledge of values. For this, we assume knowledge of Hamming weights of inputs and outputs to the S-Boxes at positions 0 and 1. Small size of the S-Boxes actually does not make our task much more difficult than before. Examining the S-Boxes $V_0, \ldots, V_3$ of PRINTCipher it may be seen that input and output weights uniquely determine the values of inputs and outputs, except

for the case when both weights are equal to 2. In this case, for each S-Box, there are exactly two pairs of input/output that yield weight 2. Therefore, such an ambiguity happens with probability 1/4 for each of the S-Boxes $V_0, \ldots, V_3$. By guessing key bits at nibble positions $\{0, 1\}$, we fix the two S-Boxes at these positions. If for an S-Box the input and output weight is equal to 2, we have to choose one of the two potential values of inputs/outputs. In the case of other weights, we uniquely determine the values, since S-Boxes are fixed now. Therefore, on average, we expect 1/4 of the guesses for nibble positions $\{0, 1\}$ at round 4 to be wrong for a correct key guess. Therefore, for one guess of the 10 key bits, we will have to run our attack with p pairs on average around

$$2^{2 \cdot p/4 - 1} = 2^{p/2-1}$$

times, since $p/2$ out of $2p$ outputs will have wrong values on average and we expect to attain correct values after $2^{p/2-1}$ times. For estimating the total attack complexity, we use the formula

$$T_{sc-attack} = T_{wrong} \cdot 2^{8+p/2}.$$

This means multiplying the values in Table 25 by $2^{p/2-1}$. Table 26 shows the results.

Table 26. Side channel attack assuming knowledge of Hamming weights of input/output to S-Boxes 0 and 1 at round 4

p	T_{wrong}, sec.	$T_{sc-attack}$, days
10	62	12
12	8	3

We are able to find the key by only knowing Hamming weights for input/output to S-Boxes at positions 0 and 1 at round 4 on average in less than two weeks for $p = 10$ and in around 3 days for $p = 12$.

Note that although the number of bits or Hamming weights we require to be leaked is not so small, the place to get these leaks is localized to only two nibbles at round 4, rather than smeared all over the cipher's encryption function. Therewith, it may be possible to better exploit these leaks.

Remark 3. Unfortunately, we were not able to get any results for $x = 1$ in a reasonable time with CryptoMiniSAT2. We were able, though, to run our "direct" attack described in Section 6.2. The average time lies in 100–1000 region for $p = 32$. Note, however, that such an approach works only if the 5 key bits and the $32 \cdot 3 = 96$ internal state bits are correct. If they are wrong, there is no good way to show it with the "direct" attack, since we are not given the variety $V(4, 32, \{0\})$, but are sampling elements from it. So one may only make some probabilistic statements about absence of the solution (False-instance). This, however, would quite negatively reflect on the time complexity.

6.4 Towards cryptanalysis of the full PRINTCipher-48

In this subsection we investigate the potential of the method described in Section 6.2 to break the full cipher. Since the complexity of the method is not exponential in the number of rounds, but rather in the number of pairs used (i.e. number of internal state bits to be guessed), this approach appears to be worth investigating. Whereas in Section 6.2 and 6.3 we assumed information about intermediate values been given to us, for the cryptanalysis we have to guess this information. Note that if x nibbles are targeted for guessing and p known plaintext/ciphertext pairs are used, we need to guess $5x$ key bits and $3px$ intermediate bits; this sums up to $(3p + 5)x$ bits. We estimate the attack complexity by the formula

$$T_{attack} = T_{wrong} \cdot 2^{(3p+5)x},$$

similarly to Section 6.3. Table 27 contains comparisons of the attack for different x and p and the brute force attack. Namely, we compute the ratio of expected time for our attack and the brute force. The table suggests that in the best case we are around 2^{17} slower than the brute force.

Table 27. Comparing our attack with brute force

p	x	$(3p+5)x$	$3px$	$\log_2(T_{attack}/T_{bf,48})$
10	2	70	60	20
12	2	82	72	29
2	7	77	42	21
2	6	66	36	17

The following observation gives a hope that the method may still be applicable. Note that in the previous subsections we always assumed known plaintext/ciphertext pairs. Therewith intermediate state bits at round 4 may be assumed to be independent. So we really have to guess at $3px$ bits. It could be possible to apply chosen plaintext scenario with the goal to be able to compute certain intermediate bits via others that have been guessed at already. Ideally, we would like to have a possibility to compute intermediate bits at round 4 for $p-1$ pairs from those bits for just one plaintext/ciphertext. Clearly, plaintexts for the remaining $p-1$ pairs have to be chosen accordingly to allow such a relation. Other possible scenarios could be applied. It may be possible to compute intermediate bits for some $p-s$ pairs having those from s pairs. We may even be satisfied with computing values with some reasonably high probability. Alternatively, we could increase the number of known or chosen pairs to somehow spot those pairs that yield prescribed values at round 4.

Anyway, results of Table 27 suggest that if we were able e.g. to reduce the number of guessed internal bits from 60 to 39 in the case of $x = 2, p = 10$ or from 36 to 18 for $x = 6, p = 2$ we would have been better than the brute force. We set the problem of finding appropriate chosen plaintexts as an open problem.

7 Conclusion and future work

In this paper we considered algebraic attacks on PRINTCipher-48 with SAT-solving as a tool. We showed that it is possible to attack 8–9 rounds of the cipher using only 2 known plaintexts. Next, we observed an interesting fact on high diffusion / low algebraic complexity at round 4, which enabled us to give a simulation side channel attack on the full cipher with 48 rounds with practical complexity.

As future work we may identify the following.

- Obtaining symbolically for between internal states for several plaintext/ciphertext pairs in line with [23].
- Apply the above point to chosen plaintext scenario and also to ideas described in Section 6.4.
- Provide more accurate and statistically sound estimations for time complexity. This work is not specific to PRINTCipher and is of great importance for the entire field of applying SAT-solvers to cryptanalysis.
- More research should be put in studying the effect of using low diffusion trails as indicated in Remark 2. Therewith more advanced guessing techniques could emerge.

Acknowledgements

The author is supported by the German Science Foundation (DFG) grant BU 630/22-1. He would like to thank Johannes Buchmann for continuous support of his work. Special thanks go to Denise

Demirel for writing a considerable portion of the equation generator for PRINTCipher. The author is grateful to Martin Albrecht for numerous discussions and also for sharing his source codes for relevant matters. Thanks to Mohamed Saied Emam Mohamed for reading this manuscript. The author is also thankful to Mate Soos for assisting with using CryptoMiniSAT.

References

1. 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.
2. 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.
3. 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.
4. M.A. Abdelraheem, G. Leander, E. Zenner, "Differential cryptanalysis of round-reduced PRINTcipher: Computing roots of permutations". In Proceedings of FSE 2011, to appear.
5. G. Leander, M.A. Abdelraheem, H. AlKhzaimi, E.Zenner, "A Cryptanalysis of PRINTCipher: The Invariant Coset Attack", to appear at CRYPTO 2011.
6. X. Zhao, T. Wang, S. Guo, "Fault Propagate Pattern Based DFA on SPN Structure Block Ciphers using Biwise Permutation, with Application to PRESENT and PRINTCipher", ePrint, available at `http://eprint.iacr.org/2011/086.pdf`
7. M. Soos, "Grain of Salt - An Automated Way to Test Stream Ciphers through SAT Solvers", available at `http://www.msoos.org/grain-of-salt`
8. N. Courtois, G.V. Bard, D. Wagner, "Algebraic and Slide Attacks on Keeloq". In K. Nyberg (Ed.) FSE 2008, LNCS 5086, pp. 97–115.
9. N. Courtois, G.V. Bard, "Algebraic Cryptanalysis of the Data Encryption Standard". In S.D. Galbraith (Ed.) IMA int. Conf. 2007, LNCS 4887, pp. 345–359.
10. G.V. Bard, N.T. Courtois, J. Nakahara Jr., P. Sepehrdad, D. Zhang, "Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers". In G. Gong, K.C. Gupta (Eds.) IndoCrypt 2010, LNCS 6498, pp.176–196.
11. C. Bouillaguet, P. Derbez, O. Dunkelman, N. Keller, P.-A. Fouque, "Low Data Complexity Attacks on AES", available at `http://eprint.iacr.org/2010/633.pdf`
12. B. Collard, F.-X. Standaert, "A Statistical Saturation Attack against the Block Cipher PRESENT". In M. Fischlib (Ed.) CT-RSA 2009, LNCS 5473, pp.195–211.
13. N. Een, N. Sorensson, "Minisat – A SAT Solver with Conflict-Clause Minimization". In E. Giunchiglia, A. Tacchella (Eds.) SAT 2003, LNCS 2919, pp.502–518.
14. M. Soos, "CryptoMiniSat – a SAT solver for cryptographic problems", available at `http://planete.inrialpes.fr/s̃oos/CryptoMiniSat2/index.php`
15. G.V. Bard,"Algebraic Cryptanalysis", Springer, 2009.
16. M. Brickenstein,"Boolean Gröbner bases – Theory, Algorithms and Applications", Logos Berlin, 2010.
17. M. Albrecht,"Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis". Ph.D. thesis. Royal Holloway, University of London, available at `http://www.sagemath.org/files/thesis/albrecht-thesis-2010.pdf`
18. S. William Stein et al., "SAGE Mathematics Software". The Sage Development Team, 2008. Available at `http://www.sagemath.org`
19. M. Albrecht, M. Soos, "Boolean Polynomial SAT-Solver", avaiable at `http://bitbucket.org/malb/algebraic_attacks/src/tip/anf2cnf.py`
20. M.Brickenstein, "PolyBoRi's CNF converter", available at `https://bitbucket.org/malb/algebraic_attacks/src/013dd1b793e8/polybori-cnf-converter.py`
21. M. Karnaugh, "The map method for synthesis of combinational logic circuits". Transactions of American Institute of Electrical Engineers part I 72(9) (November 1953) 593–599.
22. C.P. Gomes, A. Sabharwal, B. Selman, "Handbook of Satisfiability", chapter Model Counting, pages 633-654. IOS Press, 2009.
23. M. Albrecht, C. Cid, T. Dullien, J.-C. Faugère, L. Perret, "Algebraic Precomputations in Differential and Integral Cryptanalysis". In X. Lai, M. Yung, D. Lin (Eds.) INSCRYPT 2010, LNCS 6584.

A Appendix: equations for different algebraic representation of a PRINTCipher S-Box

A.1 Explicit cubic equations describing the key dependent permutation SP

$$
\begin{aligned}
W_0 &= A_0A_1X_1 + A_0A_1X_2 + A_0X_0 + A_0X_1 + X_0,\\
W_1 &= A_0A_1X_0 + A_0A_1X_2 + A_0X_0 + A_1X_1 + A_0X_1 + A_1X_2 + X_1,\\
W_2 &= A_0A_1X_0 + A_0A_1X_1 + A_1X_1 + A_1X_2 + X_2.
\end{aligned}
$$

A.2 Explicit quartic equations describing the key dependent S-Box

$$
\begin{aligned}
Y_0 &= A_0A_1X_0X_1 + A_0A_1X_0X_2 + A_0A_1X_1 + A_0A_1X_2 + A_0X_0X_2 + A_0X_1X_2 +\\
&+A_0X_0 + A_0X_1 + X_1X_2 + X_0,\\
Y_1 &= A_0A_1X_0X_1 + A_0A_1X_1X_2 + A_0A_1X_0 + A_0A_1X_1 + A_0X_0X_2 + A_0X_1X_2 +\\
&+A_1X_0X_1 + A_1X_0X_2 + A_1X_1 + A_1X_2 + X_0X_2 + X_0 + X_1,\\
Y_2 &= A_0A_1X_0X_2 + A_0A_1X_1X_2 + A_1X_0X_1 + A_1X_0X_2 + X_0X_1 + X_0 + X_1 + X_2.
\end{aligned}
$$

A.3 Equations of degree at most 2 that describe the key dependent S-Box

$$
\begin{aligned}
&(X_0 + X_1 + X_2) + (W_0 + W_1 + W_2) = 0,\\
&(A_0 + 1) \cdot (X_2 + W_2) + A_1 \cdot (X_1 + W_1) = 0,\\
&(X_2 + W_2) \cdot (A_1 + 1) = 0,\\
&A_0 \cdot (X_0 + X_1) + A_1 \cdot (W_1 + W_2) + (X_1 + W_1) = 0,\\
&Y_0 = W_1W_2 + W_0,\\
&Y_1 = W_0W_2 + W_0 + W_1,\\
&Y_2 = W_0W_1 + W_0 + W_1 + W_2.
\end{aligned}
$$

A.4 Equations from the key dependent S-Box that are products of linear polynomials

$$
\begin{aligned}
&(Y_1 + X_0 + X_1 + X_2) \cdot (Y_0 + Y_1) = 0,\\
&(Y_1 + X_0 + X_1 + X_2) \cdot (Y_2 + X_0 + X_1 + X_2) = 0,\\
&(Y_1 + X_0 + X_1 + X_2) \cdot (Y_0 + Y_1 + Y_2 + X_0 + X_1 + X_2) = 0,\\
&(Y_2 + X_0 + X_1 + X_2) \cdot (Y_0 + Y_2) = 0,\\
&(Y_2 + X_0 + X_1 + X_2) \cdot (Y_0 + Y_1 + Y_2 + X_0 + X_1 + X_2) = 0,\\
&(Y_1 + Y_2) \cdot (Y_0 + Y_1 + Y_2 + X_0 + X_1 + X_2) = 0,\\
&(Y_0 + Y_1 + Y_2 + X_0 + X_1 + X_2) \cdot (A_0 + A_1 + X_0 + X_2) = 0,\\
&(Y_0 + Y_1 + Y_2 + X_0 + X_1 + X_2) \cdot (Y_1 + A_0 + A_1 + X_1) = 0,\\
&(Y_0 + Y_1 + Y_2 + X_0 + X_1 + X_2) \cdot (Y_2 + A_0 + A_1 + X_1) = 0,\\
&(Y_0 + Y_1 + Y_2 + X_0 + X_1 + X_2) \cdot (Y_1 + Y_2 + A_0 + A_1 + X_0 + X_2) = 0.
\end{aligned}
$$