Fault Analysis of the KATAN Family of Block Ciphers

Shekh Faisal Abdul-Latip[1,2], Mohammad Reza Reyhanitabar[1], Willy Susilo[1], and Jennifer Seberry[1]

[1] Center for Computer and Information Security Research,
School of Computer Science and Software Engineering,
University of Wollongong, Australia
{sfal620, rezar, wsusilo, jennie}@uow.edu.au

[2] Information Security and Digital Forensics Lab (INSFORLAB),
Faculty of Information and Communication Technology,
Universiti Teknikal Malaysia Melaka, Malaysia
shekhfaisal@utem.edu.my

Abstract. In this paper, we investigate security of the KATAN family of block ciphers against differential fault attacks. KATAN consists of three variants with 32, 48 and 64-bit block sizes, called KATAN32, KATAN48 and KATAN64, respectively. All three variants have the same key length of 80 bits. We assume a single-bit fault injection model where the adversary is supposed to be able to corrupt a single random bit of the internal state of the cipher and this fault induction process can be repeated (by resetting the cipher); i.e., the faults are transient rather than permanent. First, we show how to identify the exact position of faulty bits within the internal state by precomputing difference characteristics for each bit position at a given round and comparing these characteristics with ciphertext differences (XOR of faulty and non-faulty ciphertexts) during the online phase of the attack. Then, we determine suitable rounds for effective fault inductions by analyzing distributions of low-degree (mainly, linear and quadratic) polynomial equations obtainable using the cube and extended cube attack techniques. The complexity of our attack on KATAN32 is 2^{59} computations and about 115 fault injections. For KATAN48 and KATAN64, the attack requires 2^{55} computations (for both variants), while the required number of fault injections is 211 and 278, respectively.

Key words: Block ciphers, cube attack, differential fault analysis, KATAN

1 Introduction

Fault analysis as a type of side channel attack (or implementation attack) was originally introduced by Boneh et al. [6] by an attack against implementations of public key algorithms. The method was then adapted and extended by Biham and Shamir [5] to differential fault analysis, making it applicable to implementations of symmetric key algorithms as well [9, 10]. Several models for fault attacks have been introduced in the literature, among which we adopt a popular model, called transient single-bit fault model, as used for example in [10, 9]. In this model it is assumed that adversary can induce one bit of error into the internal state of a cipher during its execution (e.g. using a laser beam) without damaging the bit position permanently; that is, the cipher can be reset to resume its normal (unfaulty) operation and this fault induction can be repeated as many times as required. For some interesting practical settings for carrying out these attacks we refer to [14].

In this paper we present fault attacks on the KATAN family of block ciphers [7]. KATAN consists of three variants with 32, 48 and 64-bit block sizes, named KATAN32, KATAN48 and KATAN64, respectively. All three variants have the same key length of 80 bits. KATAN aims at meeting the needs of an extremely resource-limited environment such as RFID tags. Assuming the transient single-bit fault attack model, we present a differential fault attack empowered by the algebraic techniques of the cube attack [8] and its extended variants [1].

The cube attack, put forth by Dinur and Shamir at EUROCRYPT 2009 [8], is a generic type of algebraic attack that may be applied against any cryptosytem, provided that the attacker has access to

a bit of information that can be represented by a low-degree multivariate polynomial over GF(2) of the secret and public variables of the target cryptosytem. Dinur and Shamir in [8] compared the cube attack to some of the previously known similar techniques [13, 15]. Recently, Abdul-Latip et al. [1] showed an extended variant of the cube attack to extract low-degree (mainly quadratic) sparse system of equations in addition to the linear equations obtainable from the original cube attack. We use these techniques together with fault analysis to build a hybrid attack against KATAN.

First, we show how to apply the cube and extended cube methods to extract a system of low-degree polynomial multivariate equations in the key and plaintext variables, using differences between faulty and non-faulty ciphertext bits in a chosen plaintext attack scenario. Next, we show how to determine the faulty bit positions within the internal state using bit strings called difference characteristics, and how to construct such a difference characteristic for a particular faulty bit of a certain round. Using the cube method, we also determine the most effective rounds in which faults should be induced for all three variants of KATAN. Our fault attack on KATAN32 requires 2^{59} computations and turns out to need about 2^8 times more (off-line) operations compared with the previous side channel attack by Bard et al. [2] which requires 2^{51} computations; nevertheless, our attack model (namely, assuming that an adversary can induce a fault at a random bit position in the internal state and then observes the associated ciphertext) is essentially *different* from (and arguably sounds more practical than) the attack model used by Bard et al. [2]. Bard et al. [2] assume that adversary can obtain (read) the exact ("error free") value of a bit in the internal state, and as they stated in [2], "such data is supposed to have been independently captured by some side channels for instance, power or timing analysis or electromagnetic emanations"(but we note that none of these side channel methods provides error free measurements in practice). Note that the side channel attack model of [2] is not a fault attack. Furthermore, our attack is directly adapted to the cases of KATAN48 and KATAN64 (both requiring 2^{55} computations) and, so far, is the only attack against the latter variants of KATAN in the side channel attack model.

2 A Brief Description of KATAN

KATAN is a family of block ciphers [7] consisting of three variants, namely: KATAN32, KATAN48 and KATAN64. Each variant accepts an 80-bit secret key and performs 254 rounds to produce a ciphertext. All variants also share the same key schedule as well as the same nonlinear functions. KATAN ciphers aim at constrained environments such as hardware implementations with limited resources (power consumption, clock frequency and gate counts). KATAN32 with block size of 32 bits is the lightest variant in the family. A 32-bit plaintext block is loaded into two registers L_1 and L_2, respectively, of length 13 and 19 bits. The bits are indexed in the right-to-left order, from 0 to 12 for L_1 (i.e. $L_1 = (L_1[12], \cdots, L_1[0])$) and from 0 to 18 for L_2 (i.e. $L_2 = (L_2[18], \cdots L_2[0])$). The least significant bit (LSB) of the plaintext block is loaded to bit 0 of register L_2 followed by the other bits until the 18-th bit, and then remaining bits are loaded into register L_1 until the most significant bit (MSB) of the plaintext is loaded into bit 12 of register L_1. One round of KATAN32 consists of shifting the register L_1 and L_2 one bit to the left, and computing two new bit values using nonlinear functions f_a and f_b, respectively. These new bits are then loaded into the LSB bits of registers L_2 and L_1, respectively. The nonlinear functions f_a and f_b are defined as follows:

$$f_a(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_a \quad (1)$$

$$f_b(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_b \quad (2)$$

where IR specifies an irregular update rule (i.e. $L_1[x_5]$ is used only when $IR = 1$), and k_a and k_b are two subkey bits. We refer to [7] for the details on the irregular update rules (IRs) for each round.

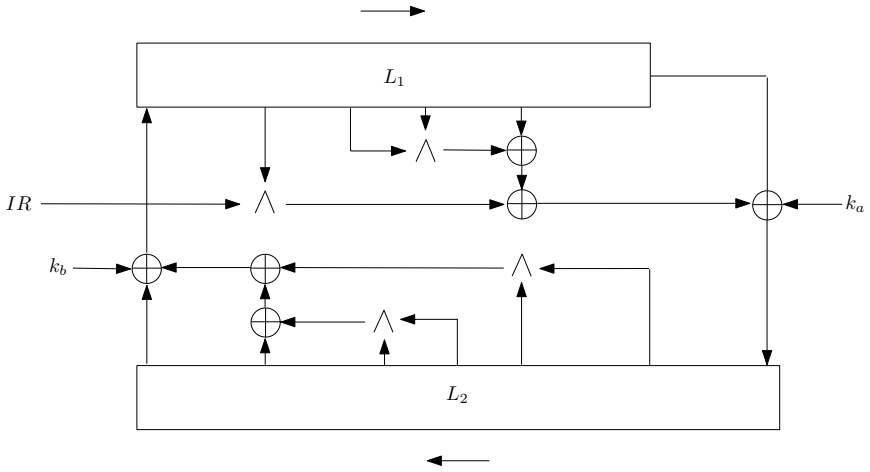

Fig. 1. The Outline of the KATAN Family of Block Ciphers

The key schedule for all variants of KATAN expands an 80-bit secret key K to 508 subkey bits using the following linear mapping

$$k_i = \begin{cases} K_i, & \text{for } 0 \le i \le 79, \quad (3) \\ k_{i-80} + k_{i-61} + k_{i-50} + k_{i-13}, & \text{otherwise} \quad (3') \end{cases}$$

Given the precomputed subkey values, the values of k_a and k_b for a particular round t are defined as k_{2t} and k_{2t+1}, respectively. Thus the subkey for round t is defined as $k_a||k_b = k_{2t}||k_{2t+1}$. The selection for tap positions, x_is ($1 \le i \le 5$) and y_js ($1 \le j \le 6$), and the length of registers L_1 and L_2 are defined independently for each variant as shown in Table 1. Besides the tap positions and the length of the

Table 1. Parameters for the KATAN Family of Block Ciphers

Cipher	$\|L_1\|$	$\|L_2\|$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32	13	19	12	7	8	5	3	18	7	12	10	8	3
KATAN48	19	29	18	12	15	7	6	28	19	21	13	15	6
KATAN64	25	39	24	15	20	11	9	38	25	33	21	14	9

registers, the difference between all the three variants is the number of times the nonlinear functions f_a and f_b are applied in each round using the same subkey. One round of KATAN48 is shifting the registers L_1 and L_2 two bits to the left (i.e. requires two clock cycles). In each shift within the same round, the function f_a and f_b are applied using the same subkey $k_a||k_b$. Hence, full round of KATAN48 requires 508 clock cycles (i.e. 254 rounds $\times$ 2 clocks per round) to produce the ciphertext.

In contrast, one round of KATAN64 requires the registers L_1 and L_2 to be shifted three bits to the left (i.e. requires three clock cycles). Similarly, in each shift within the same round, the function f_a and f_b are applied using the same subkey $k_a||k_b$. As a result, the full round KATAN64 requires 762 clock cycles to produce the ciphertext. Fig. 1 shows the generic structure of the KATAN family of block ciphers.

The initial state of KATAN-v (for v=32, 48, 64) is denoted by $IS = (s_{v-1}, \cdots, s_1, s_0) = L_1||L_2$ for the associated L_1 and L_2 registers.

3 An Overview of the Cube and Extended Cube Attacks

The main idea underlying the cube attack [8] is that the multivariate "master" polynomial $p(v_1, \cdots, v_m, k_1, \cdots, k_n)$, representing an output bit of a cryptosystem over GF(2) of secret variables k_i (key bits) and public variables v_i (i.e. plaintext or initial values), may induce algebraic equations of low degrees, in particular *linear* equations. The cube attack provides a method to derive such lower degree (especially linear) equations, given the master polynomial only as a black-box which can be evaluated on the secret and public variables.

Let's ignore the distinction between the secret and public variables' notations and denote all of them by $x_i, \cdots, x_\ell$, where $\ell = m + n$. Let $I \subseteq \{1, ..., \ell\}$ be a subset of the variable indexes, and t_I denote a monomial term containing multiplication of all the x_is with $i \in I$. By factoring the master polynomial p by the monomial t_I, we have:

$$p(x_1, \cdots, x_\ell) = t_I \cdot p_{S(I)} + q(x_1, \cdots, x_\ell) \tag{4}$$

where $p_{S(I)}$, which is called the *superpoly* of t_I in p, does not have any common variable with t_I, and each monomial term t_J in the residue polynomial q misses at least one variable from t_I. A term t_I is called a "*maxterm*" if its superpoly in p is linear polynomial which is not a constant, *i.e.* $deg(p_{S(I)}) = 1$.

The main observation of the cube attack is that, the summation of p over t_I, i.e. by assigning all the possible combinations of 0/1 values to the x_is with $i \in I$ and fixing the value of all the remaining x_is with $i \notin I$, the resultant polynomial equals $p_{S(I)} \pmod 2$. Given access to a cryptographic function with public and secret variables, this observation enables an adversary to recover the value of the secret variables (k_is) in two steps, namely the preprocessing and online phases.

During the preprocessing phase, the adversary first finds sufficiently many maxterms, i.e. t_Is, such that each t_I consists of a subset of public variables $v_1, \cdots, v_m$. To find the maxterms, the adversary performs a probabilistic linearity test (such as the BLR test of [4]) on $p_{S(I)}$ over the secret variables $k_i \in \{k_1, \cdots, k_n\}$ while the value of the public variables not in t_I are fixed (to 0 or 1) (cf. [8] for more details).

Then the next step is to derive linearly independent equations in the secret variables k_is from $p_{S(I)}$ that are closely related to the master polynomial p, such that, solving them enables the adversary to determine the values of the secret variables. Once sufficiently many linearly independent equations in the secret variables are found, the preprocessing phase is completed. In the online phase, the adversary's aim is to find the value of the right-hand side of each linear equation by summing the black box polynomial p over the same set of maxterms t_Is which are obtained during the preprocessing phase. Now, the adversary can easily solve the resultant system of the linear equations, e.g. by using the Gaussian elimination method, to determine the values of the secret (key) variables.

A generalized variant of the cube attack, called extended cube, has been shown in [1] for extracting "low-degree nonlinear" equations efficiently. It revises the notion of tweakable polynomials from the original cube attack as

$$p(x_1, ..., x_\ell) = t_I \cdot X_K \cdot p_{S(I \cup K)} + q(x_1, ..., x_\ell) \tag{5}$$

where t_I is a subterm of size s over x_is with $i \in I$; X_K is a subterm of size r over x_is with $i \in K$, and $p_{S(I \cup K)}$ is the superpoly of $t_I \cdot X_K$ in p. Note that since both subterms t_I and X_K are factored out from p, the superpoly $p_{S(I \cup K)}$ does not contain any common variable with t_I and X_K, and each term t_J in the residue polynomial q misses at least one variable from $t_I \cdot X_K$. Now using the main observation of the cube attack, the summation of p over '$t_I \cdot X_K$', by assigning all the possible combinations of 0/1 values to the x_is with $i \in I \cup K$ and fixing the value of all the remaining x_is with $i \notin I \cup K$, the resultant polynomial equals to $p_{S(I \cup K)} \pmod 2$.

The only difference between the original cube attack and the extended cube attack is in the preprocessing phase; the online phase for both of the methods are the same. During the preprocessing phase

of the extended cube attack, the adversary finds many monomials t_Is, such that each t_I consists of a subset of public variables $v_1, \cdots, v_m$, and the corresponding superpoly $p_{S(I)}$ is a polynomial of degree D. To find those t_Is, the adversary performs the generalized version of the BLR test as proposed by Dinur and Shamir in [8] on $p_{S(I)}$ over the secret variables $k_1, \cdots, k_n$.

To derive efficiently a nonlinear equation $p_{S(I)}$ of degree D over secret variables k_is, the adversary should identify the subset $S \subseteq \{1, \cdots, n\}$ that consists of the secret variable indexes within $p_{S(I)}$, in which each k_i with $i \in S$ is either a term or a subterm of $p_{S(I)}$. To do this, the subterm X_K (cf. equation (5)) is assigned with each secret variable $k_i \in \{k_1, \cdots, k_n\}$ one at a time while the subterm t_I is fixed to the monomial in which its superpoly $p_{S(I)}$ is of degree D, and all public variables v_is with $i \notin I$ are fixed to 0 or 1. For each assignment of X_K, the adversary chooses κ sets of vector $\mathbf{x} \in \{0,1\}^{n-1}$ representing samples of $n-1$ secret variables k_is with $i \notin K$ independently and uniformly at random, and verify that X_K (or similarly the secret variable k_i that is assigned to X_K) exists as a variable in the superpoly $p_{S(I)}$ if $p_{S(I \cup K)} = 1$ for at least an instance vector $\mathbf{x}$.

Having the set of secret variables k_is with $i \in S$ of the nonlinear superpoly $p_{S(I)}$ of degree D enables the adversary to derive the nonlinear equation over the secret variables by finding all terms of degrees $0, 1, \cdots, D$ within the superpoly equation. Suppose $N = |S|$ is the number of secret variables k_is with $i \in S$ of the superpoly $p_{S(I)}$ of degree D. To derive $p_{S(I)}$, firstly the adversary assigns the subterm X_K one at a time with a monomial indexed by a subset $K \in T$ where T is a set of cube indexes of monomials constructed from all combinations of k_is from degree 1 until degree D with $i \in S$. In each assignment, all $v_i, k_i \notin t_I \cdot X_K$ are set to zero. Then to verify the existence of the monomial $X_K \in T$ as a term in $p_{S(I)}$, the adversary sums p over the monomial $t_I \cdot X_K$. If the result is equal to 1, then with probability 1, X_K is a term in the superpoly $p_{S(I)}$. Finally, the existence of a constant term (i.e. a term of degree 0) in the superpoly $p_{S(I)}$ is also determined by setting all public variables, v_is, for $i \notin I$ and all secret variables $k_1, \cdots, k_n$ to zero, and sum the polynomial p over t_I. Similarly, if the result is equal to 1, then with probability 1, a constant term exists within the superpoly $p_{S(I)}$.

4 Fault Analysis of KATAN

We simulate a fault attack assuming that the adversary can cause one transient single-bit error at a time in the internal state during the encryption/decryption process. It is assumed that the adversary can choose the target round(s) in which faults should be induced, for example, based on the side channel information inferred from power consumption traces and/or the clocking sequence (e.g., this can be done by triggering a laser beam with the target number of clocks of the cryptographic module). However, it is assumed that adversary cannot influence the exact position of the faulty bit within the internal state; he can only induce the fault randomly with the hope that it will hit the target bit positions by chance.

Using this fault model, our aim is to recover the 80-bit secret key used in KATAN. Firstly, we demonstrate a method to determine the position of the faulty bit within the internal state using *difference characteristics*. Secondly, we show how to recover a *low-degree* system of multivariate polynomial equations which are obtainable within certain rounds of the enciphering process using the difference between faulty and non-faulty ciphertexts. More precisely, we only concentrate on extracting linear and quadratic equations from the internal state and subkey bits that are easily solvable. Having a sufficient number of independent equations that are solvable, we exploit the key schedule algorithm to recover the 80-bit secret key. Finally, we identify the faulty rounds of the enciphering process that should be considered in order to efficiently implement a successful fault attack on KATAN.

Our attack on the KATAN ciphers exploits the observation that after recovering n neighboring "subkey bits", we can recover the 80-bit "secret key" with time complexity of 2^{80-n} computations. This is because the 80-bit secret key is directly loaded into an 80-bit LFSR (the key register) and the subkey bits for round $t > 79$ are computed using a linear update function and shift operations (cf. Equation 3 and

Equation 3′). Therefore, at any round $t > 79$, if we can recover the value of some of the LFSR bits (or similarly the value of the subkey bits), we can guess the remaining $80 - n$ values of the LFSR internal state bits and iteratively clock the LFSR backward until round $t = 0$ to recover the secret key. Suppose the highest and the lowest index values of the subkey bits to be recovered are H and L respectively. Hence, our aim is to recover the subkey bits such that $H - L \leq 79$, as all subkey bits between these index range will be the content of the 80-bit LFSR at a particular round t (Section 4.1 provides more details).

Besides the key schedule algorithm, the issue also lies with the low degree boolean functions of the update functions as shown in Equation 1 and Equation 2. Having low degree update functions (such as quadratic ones as in KATAN) will cause a slow increase in the degree of the polynomials describing the cipher during the enciphering and deciphering processes. This enables the adversary to exploit low degree polynomial equations from many rounds of the cipher through (for example) side-channel attacks.

4.1 Extracting Low Degree Polynomial Equations

The main idea behind algebraic attacks is to recover the secret key of a cipher by solving a system of multivariate polynomial equations (over the plaintext and secret key variables) that describe the cipher. Since the system of equations representing a cipher is usually of very high degree, directly solving such equations, in general, to recover the secret key, is a well-known hard problem. A well-known method to try to solve such a system is linearization, i.e. replacing a high degree monomial or equation with a new variable. In our work we take a similar approach in which, to induce faults in a targeted round on KATAN, we redefine each bit of registers L_1 and L_2 as a new variable instead of defining each one of them as a boolean function over the plaintext and secret key variables. Similarly, for the key schedule algorithm, we also redefine each subkey bit that is generated by the key register update function as a new variable instead of considering each one of them as a boolean function over the 80 secret key variables. Consequently, the subkey bits are indexed from 0 to 507. Thus, the system of equations arising from the faulty and non-faulty ciphertext differential is in the linearized parameters.

Although the linearization method is used, considering fault induction at an early round of the enciphering process will cause the polynomials representing the ciphertext bits in the linearized parameters to become too complex to be analyzed explicitly. Hence, to avoid dealing with such a complex problem, we only represent them as black-box polynomials and extract them using the recently proposed cube [8] and extended cube methods [1]. This removes the need to know the explicit (enormous) symbolic representations of the related polynomials. Application of these cube based methods in our fault attack is inspired by the observation that computing ciphertext differentials (obtained by XORing non-faulty and faulty ciphertexts) in the single-bit fault model is similar to summing over the black-box master polynomial over cubes of size 1.

From the cube and extended cube methods, we found that the subkey variables only begin to appear in quadratic polynomial equations. We utilize both linear and quadratic equations to recover n subkey bits which enable us to recover the secret key by guessing the remaining $80 - n$ bits of the key register and clocking backward until round $t = 0$, where the secret key can be found.

4.2 Fault Position Determination

Since faults are randomly introduced into the internal state of registers L_1 and L_2 after a certain "known" number of rounds, say t, identifying the exact faulty bit position is necessary to ensure the success of the attack. To find this exact position, we construct a *difference characteristic* for each internal state bit generalizing the idea of [10] to locate the faulty bit position in the internal state of the Trivium stream cipher. A difference characteristic corresponding to any bit position of the internal state is a string obtained by XORing the non-faulty ciphertext and the faulty ciphertext (resulting from fault induction

at the bit position). We use right-to-left ordering of bit numbers, denoting index 0 of the difference characteristic bit as the LSB of the characteristic and the highest index as the MSB. Values '0' and '1' represent difference values 0 and 1 respectively for the corresponding characteristic bits with probability 1, while the '-' sign represents unknown values (i.e. can be either '0' or '1'). The difference characteristic for each faulty bit position of a certain round can be represented by a lookup table. For lack of space, we have only shown one example table representing the difference characteristic for the case of KATAN32, for an internal state bit s_{j+t} (after t=231 rounds for an unknown position $0 \leq j \leq 31$) in Table 6 in Appendix.

Given a faulty ciphertext resulting from a random fault induction into an "unknown" internal state bit s_{j+t} after t-th round, to determine the position j, first we compute the ciphertext differential, Δc, by XORing (summing modulo 2) the non-faulty ciphertext c with the faulty ciphertext c' such that $\Delta c_j = c_j \oplus c'_j$, for $0 \leq j < |L_1| + |L_2|$. Then, guided by the lookup table, we refer to positions with values '0' and '1' (and ignore those with a '-' sign) within each characteristic and compare them with bits in the same positions in Δc. If all the corresponding bits in Δc match the bits in the characteristic of the faulty bit s_{j+t} then we can ensure that a fault has been induced into the bit at position j. However, there might be a case where there are no characteristics uniquely distinguishing two or more faulty bit positions. This may occur as the result of inducing faults in the early rounds of the enciphering (or deciphering) process. If there were only very few faulty bit positions sharing the same characteristic, we can just guess them in order to find the correct one. Having too many faulty bit positions sharing the same characteristic will cause extra overhead in the attack complexity. Thus, one should try to consider later rounds to avoid such an overhead.

Constructing Difference Characteristics. To construct a differential characteristic, we study the error propagation in the ciphertext bits due to one faulty bit at a certain position in the internal state. Based on this we can determine which bits within the ciphertext are

- affected by the faulty bit with *probability 1*,
- not affected by the faulty bit with *probability 1*, or
- affected by the faulty bit with some *probability less than 1*.

The ciphertext bits that are certainly affected (affected with probability 1) will always give value '1' in the differential, while those that are certainly not affected will always be '0' in the differential. However, the bits that are affected with some non-zero probability less than 1 can hold either value '0' or '1' in the differential (i.e. an unknown value) and are denoted by '-' in the difference characteristic.

To know how a faulty bit after round t affects bits in the ciphertext, we consider each ciphertext bit as a boolean function in the linearized variables after round t. Then we analyze how the faulty bit can appear as a parameter in the boolean function. There are three ways that a faulty bit can appear as a parameter within the polynomial describing the the boolean function, namely it either

- exists as a term but not as a subterm of some monomials in the polynomial (i.e. appears as a linear variable),
- does not exist within the polynomial, or
- exists as a subterm of some monomials (and probably also as a term) within the polynomial.

To identify which of the above three cases occurs, we utilize the method used in the cube attack as described in [8]. We select the faulty bit as the monomial, t_I, and apply the linearity test on the corresponding superpoly, $p_{S(I)}$, to determine whether the test will result constant 0, constant 1, linear or higher degree superpoly. Constant 0 and constant 1 superpolys indicate values '0' and '1' in the difference characteristic bits, respectively. However linear and higher degree superpolys indicate unknown values in the characteristic bits, i.e. the '-' sign.

4.3 Finding Effective Rounds for Fault Induction

To recover most of the subkey bits, the rounds which contain a high number of quadratic equations (resulting from non-faulty and faulty ciphertext differential) need to be determined as the subkey bits only begin to appear within these equations. We analyze the distribution of the linear and quadratic equations after each round, obtainable from non-faulty and faulty ciphertext differentials, by considering every bit (one bit at a time) of the internal state (i.e. from register L_1 and L_2) being induced by a fault value. We apply the cube and extended cube methods considering cubes of size 1 to simulate the non-faulty and faulty ciphertext differentials due to one faulty bit for each of the internal state bits and after each round. For each linear and quadratic equations found, we accumulate the total number of such equations for each round. Fig. 2 shows the result of our analysis. In the figure, "Faulty Round" denotes

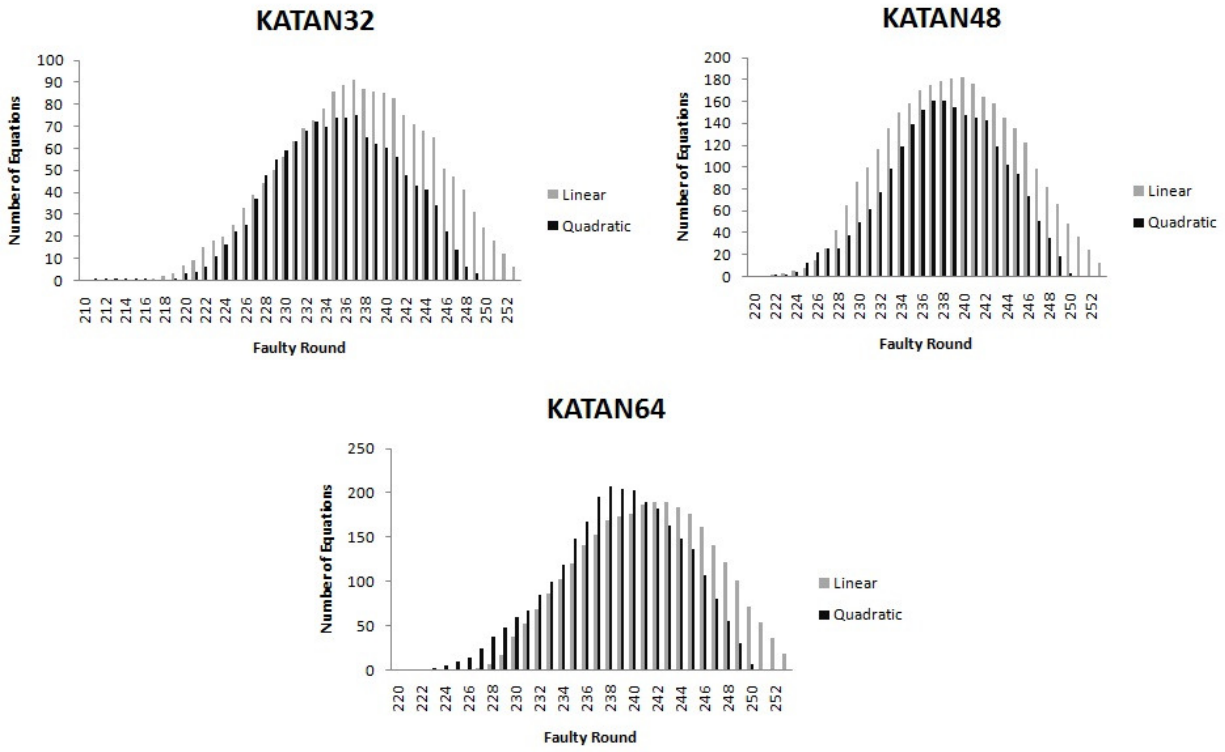

Fig. 2. Distribution of Linear and Quadratic Equations in KATAN

the number of rounds that the cipher has run before inducing a fault into the internal state. It is obvious that the ranges of faulty rounds which can provide a high number of quadratic equations for all variants of KATAN are about the same. We call such Faulty Round numbers *effective rounds* for a particular variant. Thus, the fault attack on KATAN will be possible if faults are induced into the internal state within these specific effective rounds.

4.4 Attack on KATAN32

The fault attack can be efficiently applied if the rounds that have high number of quadratic equations (resulting from non-faulty and faulty ciphertext differentials) are considered. As for KATAN32, this refers to the fault inductions after $t = 237$ rounds as shown in Fig. 2. Considering this faulty round, we provide a sample set of linear and quadratic equations that can help in recovering the target subkey bits as shown in Table 3 in Appendix.

In the table, $L_2 = (s_{18+t}, \cdots, s_{0+t})$ and $L_1 = (s_{31+t}, \cdots, s_{19+t})$. Δc_j denotes a ciphertext bit difference where the difference is obtained by XORing the non-faulty ciphertext bit c_j with the faulty ciphertext bit c'_j, i.e. $\Delta c_j = c_j \oplus c'_j$, for $0 \leq j \leq 31$. For subkey bits we use a slightly different notation to facilitate our analysis, in which we denote the k_is as subkey bits whose indexes range from $0 \leq i \leq 507$ (in which bits indexed 0 until 79 are from the original secret key bits). We do not consider each subkey bit indexed $i > 79$ as a boolean function over the 80 secret key bits. Instead, to facilitate our analysis we only consider each one of them as an independent new variable.

Considering fault induction after $t = 237$ rounds, 10 subkey bits can be found within the quadratic equations, i.e. $k_{474}, \ldots, k_{482}$ and k_{484} (cf. Table 3 for the polynomial equations and Table 6 for the difference characteristics in Appendix). Recovering these subkey bits, requires solving the corresponding quadratic equations in which some of the linear equations listed in the table should also be involved, as they can provide the solution for the internal state bits of registers L_1 and L_2 within the quadratic equations. For example, to find the solution for k_{474}, we consider s_{1+t} as the faulty bit after $t = 237$ rounds. Considering the difference between non-faulty and faulty ciphertext bit c_{24}, i.e. Δc_{24}, the symbolic representation of the differential is

$$s_{22+t} + s_{26+t} + s_{31+t} + k_{474} + s_{24+t}s_{27+t} = \Delta c_{24}. \tag{6}$$

The value of the right hand side (RHS) of this equation (either 0 or 1) can be determined by numerically computing Δc_{24}, such that $\Delta c_{24} = c_{24} \oplus c'_{24}$. Then recovering k_{474}, requires the bits s_{22+t}, s_{26+t}, s_{31+t}, s_{24+t} and s_{27+t} to be known. With the exception of bit s_{31+t}, all these bits can be recovered directly by utilizing the linear equations available within the faulty round. However, observation of Table 3 shows that no solution can be found for s_{31+t} which prevents us from finding the solution for k_{474}. Applying the same approach to recover $k_{475}, k_{476}, k_{477}, k_{478}, k_{479}, k_{480}, k_{481}, k_{482}$ and k_{484}, we encounter the same problem, in which we are unable to recover those subkey bits, except for subkey bit k_{484}. This is because no solutions can be found for $s_{18+t}, s_{30+t}, s_{17+t}, s_{29+t}, s_{16+t}, s_{28+t}$ and s_{15+t} to make the system of equations in the subkey and internal state bits solvable. However, knowing k_{484} enables us to reduce the key space by 50 percent (more explanation about this in Section 4.7).

Next, we further reduce the complexity of our attack by recovering the unknown bits s_{31+t}, s_{18+t}, s_{30+t}, s_{17+t}, s_{29+t}, s_{16+t}, s_{28+t} and s_{15+t} to find the solutions for the remaining 9 subkey bits. We exploit the properties of registers L_1 and L_2 to recover those bits. It is obvious that all bits in L_1 and L_2 except s_{0+t} and s_{19+t}, are updated each round only by a shift operation (i.e. bit indexed j is moved to bit indexed $j + 1$). Thus we can search for equivalent bits in any other rounds (i.e. other than $t = 237$) to recover the unknown bits from the original faulty round. Now, we consider round $t = 231$ as another faulty round (cf. Table 2 for the polynomial equations). Since $t = 231$ is 6 rounds earlier than $t = 237$, bit s_{j+t} at faulty round $t = 237$ can be considered similarly to bit s_{j-6+t} at faulty round $t = 231$. This implies bits s_{31+t}, s_{18+t}, s_{30+t}, s_{17+t}, s_{29+t}, s_{16+t}, s_{28+t} and s_{15+t} at round $t = 237$ are similar to bits s_{25+t}, s_{12+t}, s_{24+t}, s_{11+t}, s_{23+t}, s_{10+t}, s_{22+t} and s_{9+t} at round $t = 231$ respectively being shifted 6 clocks towards the MSB. Hence, considering faulty round $t = 231$, the bits s_{25+t}, s_{12+t}, s_{24+t}, s_{11+t}, s_{23+t}, s_{10+t}, s_{22+t} and s_{9+t} can be recovered directly which enables the remaining 9 subkey bits at faulty round $t = 237$ to be recovered too.

Note that choosing the faulty round $t = 231$ enables another 10 subkey bits which do not exist in faulty round $t = 237$ to be found, i.e. bits $k_{462}, \ldots, k_{470}$ and bit k_{472}. Again, we cannot recover all

the 10 subkey bits directly as the internal state bits s_{26+t}, s_{31+t}, s_{27+t}, s_{18+t}, s_{30+t}, s_{17+t}, s_{29+t}, s_{16+t}, s_{28+t} and s_{15+t} that can help in finding the solutions for the subkey bits are unknown. Thus we repeat the process as described before to recover all the 10 subkey bits by further considering earlier rounds. However considering another earlier round will result in having difficulty in determining the faulty bit position within registers L_1 and L_2. This is because the difference characteristics that can uniquely define the faulty bit position are reducing as we consider more earlier rounds. Consequently, there exist many possible bit positions that have the same difference characteristic. Hence, to locate the exact faulty bit position, guessing the bit positions among those with similar difference characteristic is required. However, this will introduce an overhead in the attack, and hence we don't consider recovering the 10 subkey bits within this faulty round, $t = 231$ (i.e. bits $k_{462}, \ldots, k_{470}$ and bit k_{472}).

To avoid too much guessing, one should try inducing faults at round $t > 237$ as this can provide a more defined difference characteristic. We consider faulty round $t = 243$ as our next target round (cf. Table 4 for the polynomial equations in Appendix). Considering this round, we find another 10 new subkey bits within the quadratic equations in which 2 of them can be directly recovered, i.e. k_{494} and k_{496}, while the other 8 subkey bits, i.e. $k_{486}, \ldots, k_{493}$, cannot be recovered unless the internal state bits s_{31+t}, s_{18+t}, s_{30+t}, s_{17+t}, s_{29+t}, s_{16+t}, s_{28+t} and s_{15+t} are known. However, these bits can be recovered by referring directly to bits s_{25+t}, s_{12+t}, s_{24+t}, s_{11+t}, s_{23+t}, s_{10+t}, s_{22+t} and s_{9+t} respectively which can be directly solved at faulty round $t = 237$. Thus, all the 10 subkey bits can also be recovered in this round.

Finally, we consider faulty round $t = 249$ as our last target round, as no other rounds can provide subkey bits within the differential equations. In this final round, there is only one subkey bit which can be obtained from the equations, i.e. k_{498}. The equations are shown in Table 5 in Appendix. It is obvious s_{31+t} is unknown. However it can be recovered by referring to s_{25+t} at $t = 243$.

4.5 Attack on KATAN48

Following the method used on KATAN32, we consider the KATAN48 block cipher as our next target. Recall that the main differences between KATAN32 and KATAN48 are the block size, the tap positions in the internal state L_1 and L_2, and the number of clocks for each round. For KATAN48 we have $L_2 = (s_{28+t}, \cdots, s_{0+t})$ and $L_1 = (s_{47+t}, \cdots, s_{29+t})$. Since KATAN48 requires two clocks for each round, if a certain internal state bit s_{j+t} cannot be solved directly in certain faulty round t, then its solution may be found by referring to bit s_{j-2n+t} in an earlier faulty round $t - n$, for $j - 2n \geq 29$ and $31 \leq j \leq 47$, or $j - 2n \geq 0$ and $2 \leq j \leq 28$.

Our attack on KATAN48 considers faulty rounds $t = 234, 238, 242, 246, 250$ as the target rounds. Similarly to KATAN32, the selection of these rounds is based on the number of quadratic equations that can be found within the effective rounds. Fig. 2 shows that the highest number of quadratic equations for KATAN48 can be found at faulty rounds $t = 237$ and $t = 238$. Since the difference characteristics are more clearly defined when we consider later rounds: thus we choose $t = 238$ rather than $t = 237$ as our first target round.

Considering faulty round $t = 238$, we have been able to find 8 subkey bits within the quadratic equations, i.e. $k_{476}, \ldots, k_{483}$ (cf. Table 8 for the polynomial equations in Appendix). The same problem as in KATAN32 occurs where some of the internal state bits cannot be solved directly from this round which prevents the 8 subkey bits being solved. These bits are s_{22+t}, s_{23+t}, s_{25+t}, s_{27+t}, s_{45+t} and s_{46+t}. Finding the solution for each of these bits requires us to consider 4 earlier rounds, i.e. faulty round $t = 234$. At $t = 234$ (cf. Table 7 for the polynomial equations in Appendix), these bits are equal to bits s_{14+t}, s_{15+t}, s_{17+t}, s_{19+t}, s_{37+t} and s_{38+t} respectively.

Note that, we don't consider recovering the subkey bits at faulty round $t = 234$, to avoid the possibility of needing to perform too much guessing for the exact faulty bit positions in the earlier faulty round $t = 230$ (for recovering the unknown internal state bits) resulting from lack of clearly defined difference

characteristics. Hence, we consider faulty round $t = 242$ instead as our next target round. Considering this faulty round gives us another 8 subkey bits within the quadratic equations, i.e. $k_{484}, \ldots, k_{491}$ (cf. Table 9 for the polynomial equations in Appendix). There are only 5 internal state bits which cannot be solved when considering the system of equations within this faulty round. These are bits s_{22+t}, s_{23+t}, s_{25+t}, s_{27+t} and s_{46+t}. However, each of these bits holds the same value as bits s_{14+t}, s_{15+t}, s_{17+t}, s_{19+t} and s_{38+t} respectively at faulty round $t = 238$. Since the relevant bits can be solved, thus their values obtained enables the 8 subkey bits to be solved.

Another 8 subkey bits can also be found within quadratic equations in faulty round $t = 246$, i.e. $k_{492}, \ldots, k_{499}$ (cf. Table 10 for the polynomial equations in Appendix). To recover the value of these subkey bits requires 5 internal state bits, i.e. s_{22+t}, s_{23+t}, s_{25+t}, s_{28+t} and s_{47+t}, to be solved by referring to the equivalent bits at faulty round $t = 242$ which are s_{14+t}, s_{15+t}, s_{17+t}, s_{19+t} and s_{38+t} respectively. This in turn can help to provide the solution for the 8 subkey bits.

Finally we consider faulty round $t = 250$ as the last target round as no more later rounds can supply subkey bits within quadratic equations. Considering this round, only one subkey bit can be found within the quadratic equations, i.e. k_{500} (cf. Table 11 for the polynomial equations in Appendix). There is only one internal state bit, i.e. bit s_{47+t}, that is required to find the solution for the subkey bit which cannot be solved directly in this round. However this bit is equal to bit s_{39+t} in faulty round $t = 246$. Knowing the value of this bit enables the subkey bit to be recovered successfully.

4.6 Attack on KATAN64

Now we consider a fault attack on the third variant of the KATAN block cipher, namely, KATAN64. In KATAN64 we have $L_2 = (s_{38+t}, \cdots, s_{0+t})$ and $L_1 = (s_{63+t}, \cdots, s_{39+t})$. Since each round in KATAN64 requires 3 clocks, if certain internal state bits s_{j+t} cannot be recovered at faulty round t, we can try to recover their values from bit s_{j-3n+t} of faulty round $t-n$, for $j-3n \geq 39$ and $42 \leq j \leq 63$, or $j-3n \geq 0$ and $3 \leq j \leq 38$.

As in the attack on the previous two variants, we concentrate on the round which can provide the highest number of quadratic equations from the differential. According to Fig. 2, this refers to faulty round $t = 238$. There are 8 subkey bits can be found within the quadratic equations at this round, i.e. $k_{476}, \ldots, k_{483}$ (cf. Table 13 for the polynomial equations in Appendix). However 7 internal state bits are unknown to help recover those subkey bits, i.e. s_{61+t}, s_{51+t}, s_{60+t}, s_{25+t}, s_{38+t}, s_{34+t} and s_{24+t}, but these bits are equal to bits s_{55+t}, s_{45+t}, s_{54}, s_{19+t}, s_{42+t}, s_{28+t} and s_{18+t} at faulty round $t = 236$ (cf. Table 12 for the polynomial equations in Appendix). However s_{55+t} can be recovered from bit s_{3+t} and equation $s_{3+t} + s_{55+t}$ within the same faulty round $t = 236$; the bit s_{45+t} can be recovered from bit s_{2+t} and equation $s_{2+t} + s_{45+t}$; the bit s_{54+t} from s_{2+t} and $s_{2+t} + s_{54+t}$; and bit s_{32+t} can be recovered from bit s_{42+t} and equation $s_{32+t} + s_{42+t}$. However bits s_{25+t}, s_{34+t} and s_{24+t} (in faulty round $t = 238$) can be recovered by directly referring to the corresponding bits s_{19+t}, s_{28+t} and s_{18+t} in faulty round $t = 236$ respectively.

Considering faulty round $t = 242$, another 8 subkey bits can be found (cf. Table 14 for the polynomial equations in Appendix), i.e. bits $k_{484}, \ldots, k_{491}$ with 6 unknown internal state bits, i.e. s_{62+t}, s_{23+t}, s_{29+t}, s_{32+t}, s_{27+t} and s_{24+t}, which need to be solved by referring to the equivalent bits in faulty round $t = 238$, i.e. bits are s_{50+t}, s_{11+t}, s_{17+t}, s_{20+t}, s_{15+t} and s_{12+t} respectively; while 2 unknown internal state bits, i.e. s_{36+t} and s_{34+t} need to be solved by referring to the equivalent bits in faulty round $t = 236$, i.e. s_{18+t} and s_{16+t} respectively.

Next, we consider faulty round $t = 246$ to find 8 more subkey bits, namely, $k_{492}, \ldots, k_{499}$ (cf. Table 15 for the polynomial equations in Appendix). However bits s_{63+t} and s_{36+t} are unknown, preventing the recovery of k_{492} and k_{493} respectively, if only $t = 246$ is considered. This unknown bit can be recovered by referring to the equivalent bit, i.e. s_{39+t} and s_{12+t} respectively in faulty round $t = 238$.

Finally, we consider faulty round $t = 250$ as the last target round as no more subkeys can be recovered from later rounds. In this round, only one subkey bit (i.e. $t = 250$) can be found (cf. Table 16 for the polynomial equations in Appendix). However the internal state bit, s_{62} needs to be recovered by referring to bit s_{50+t} in faulty round $t = 246$ before k_{500} can be solved.

4.7 Attack Complexity

Since in our attack we are targeting more than one faulty round for fault inductions, some of the target faulty bits are used in more than one target faulty round. This observation can be used to reduce the hassle for the adversary to induce fault at different target bit positions many times and hence helps increase the success probability. More precisely, if the adversary were able to hit a target faulty bit of a certain round, without changing the position of fault induction point, the adversary could simply reset and then clock the cipher to reach the other target faulty round that require the same bit to be faulty before inducing fault at the same position. This enables the fault induction in the same bit position for different target faulty rounds to be performed with probability 1. Thus, each time a fault is induced in the internal state randomly, it should aim to hit (by chance) any one of the target faulty bits within all target faulty rounds (but not considering a particular target faulty round only) to increase the success probability. We have used this observation in making our experimental simulation of the attack against different variants as summarized in the following.

Result on KATAN32. Our experimental simulation of the attack on KATAN32 shows that 21 subkey bits from faulty rounds $t = 231, 237, 243, 249$ can be recovered, requiring collectively 20 specific internal state bit positions (regardless of the round number) to be considered as target faulty bits, as shown in Tables 2-5 in Appendix. The average number of fault injections needed to successfully hit these 20 target faulty bits is 115 (where the average is taken over 10,000 trials).

Since the highest index of the subkey bits is $H = 498$ and the lowest index is $L = 474$ (hence $H - L = 24 < 80$) the target subkey bits can be found in the 80-bit key register within rounds $209 \leq t \leq 236$. Therefore, to recover the secret key, we need to guess the remaining 59 bits of the key register and then to clock the key register backward until round $t = 0$. This reduces the complexity of the attack to 2^{59} computations compared to 2^{80} by exhaustive key search.

Result on KATAN48. The attack on KATAN48 results in recovering 25 subkey bits considering faulty rounds $t = 234, 238, 242, 246, 250$ which requires collectively 27 specific internal state bits positions to be considered as target faulty bits, as shown in Tables 7 - 11 in Appendix. The average number of required fault injections to successfully hit these 27 target faulty bits is 211 (where the average is taken over 10,000 trials).

The highest and the lowest subkey bit indexes are $H = 500$ and $L = 476$, respectively (hence $H - L = 24 < 80$), so all the subkey bits can be found within the content of the 80-bit key register at rounds $210 \leq t \leq 237$. Therefore, to recover the secret key we need to guess the remaining 55 bits of the key register and then to clock backward until round $t = 0$ to recover the secret key. Thus, finding the correct key requires 2^{55} computations in this attack.

Result on KATAN64. In attacking KATAN64 we consider faulty rounds $t = 236, 238, 242, 246, 250$ to recover (at least) the same 25 subkey bits as in the attack on KATAN48 which requires collectively 44 specific internal state bit positions to be faulty as shown in Tables 12 - 16 in Appendix. The average number of required fault injections to successfully hit these 44 target faulty bits is 278 (where the average is taken over 10,000 trials). This results in an attack with complexity 2^{55} (Noticing that the highest index of the subkey bits is $H = 491$ and the lowest index is $L = 476$ (i.e. $H - L = 15 < 80$); hence, these 25 target subkey bits can be found in the 80-bit secret key register and we only need to guess the remaining 55 bits of the key register).

References

[1] Abdul-Latip, S.F., Reyhanitabar, M.R., Susilo, W., Seberry, J.: Extended Cubes: Enhancing the Cube Attack by Extracting Low-Degree Non-Linear Equations. In: Cheung, B. et al. (Eds.) ASIACCS 2011. ACM, pp. 296–305 (2011)

[2] Bard, G.V., Courtois, N.T., Jr, J.N., Sepehrdad, P., Zhang, B.: Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers. In: Gong, G., Gupta, K.C. (Eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 176–196. Springer, Heidelberg (2010)

[3] Biham, E., Biryukov, A.: An Improvement of Davies' Attack on DES. In: Santis, A. D. (Ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 461–467. Springer, Heidelberg (1994)

[4] Blum, M., Luby, M., Rubinfield, R.: Self-Testing/Correcting with Application to Numerical Problems. In: STOC, pp. 73–83. ACM, New York (1990)

[5] Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski, B.S (Ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)

[6] Boneh, D., DeMillo, R., Lipton, R.: On the Importance of Checking Cryptographic Protocols for Faults. In: Fumy, W. (Ed.) EUROCRYPT 1997. LNCS, vol. 1223, pp. 37–51. Springer, Heidelberg (1997)

[7] de Cannière, C., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (Eds.) CHES 2009. LNCS, vol. 5754, pp. 272–288. Springer, Heidelberg (2009)

[8] Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (Ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)

[9] Hoch, J.J., Shamir, A.: Fault Analysis of Stream Ciphers. In: Joye, M., Quisquater, J.-J. (Eds.) CHES 2004. LNCS, vol. 3156, pp. 240–253. Springer, Heidelberg (2004)

[10] Hojík, M., Rudolf, B.: Differential Fault Analysis of Trivium. In: Nyberg, K. (Ed.) FSE 2008. LNCS, vol. 5086, pp. 158–172. Springer, Heidelberg (2008)

[11] Hojík, M., Rudolf, B.: Floating Fault Analysis of Trivium. In: Chowdhury, D.R., Rijmen, V., Das, A. (Eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 239–250. Springer, Heidelberg (2008)

[12] Hu, Y., Zhang, F., Zhang, Y.: Hard Fault Analysis of Trivium. Cryptology ePrint Archive, Report 2009/333 (2009)

[13] Lai, X.: Higher Order Derivatives and Differential Cryptanalysis. In: Communication and Cryptology, pp. 227–233. Kluwer Academic Publisher (1994)

[14] Skorobogatov, S.P., Anderson, R.J.: Optical Fault Induction Attacks. In: Kaliski Jr, B.S., Koç, C.K., Paar, C. (Eds.) CHES 2002. LNCS, vol. 2523, pp. 31–48. Springer, Heidelberg (2002)

[15] Vielhaber, M.: Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. IACR ePrint Archive, Report 2007/413 (2007), http://eprint.iacr.org/2007/413

[16] Vielhaber, M.: AIDA Breaks BIVIUM (A&B) in 1 Minute Dual Core CPU Time. Cryptology ePrint Archive, Report 2009/402, IACR (2009)

A Appendix

Tables 2, 3, 4 and 5 show the polynomial equations resulting from proper and faulty ciphertext differential for faulty rounds $t = 231, 237, 243, 249$ for KATAN32.

Table 2. Fault induction after $t = 231$ rounds of KATAN32

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{8+t}	Δc_7	s_{10+t}
s_{9+t}	Δc_8	s_{11+t}
s_{10+t}	Δc_9	s_{12+t}
s_{11+t}	Δc_{17}	s_{9+t}
s_{19+t}	Δc_{28}	s_{22+t}
s_{20+t}	Δc_{29}	s_{23+t}
s_{21+t}	Δc_{30}	s_{24+t}
s_{22+t}	Δc_{31}	s_{25+t}

Table 3. Fault induction after $t = 237$ rounds of KATAN32

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{1+t}	Δc_{28}	$s_{19+t} + s_{23+t} + s_{28+t} + k_{480} + s_{21+t}s_{24+t}$
	Δc_{24}	$s_{22+t} + s_{26+t} + s_{31+t} + k_{474} + s_{24+t}s_{27+t}$
	Δc_6	s_{6+t}
	Δc_4	$s_{4+t} + s_{15+t} + k_{481} + s_{0+t}s_{5+t} + s_{7+t}s_{9+t}$
s_{2+t}	Δc_{29}	$s_{20+t} + s_{24+t} + s_{29+t} + k_{478} + s_{22+t}s_{25+t}$
	Δc_{27}	s_{4+t}
	Δc_{25}	s_{0+t}
	Δc_5	$s_{5+t} + s_{16+t} + k_{479} + s_{1+t}s_{6+t} + s_{8+t}s_{10+t}$
s_{3+t}	Δc_{30}	$s_{25+t} + s_{30+t} + k_{476} + s_{23+t}s_{26+t}$
	Δc_{28}	s_{5+t}
	Δc_{26}	s_{1+t}
	Δc_{12}	s_{8+t}
	Δc_6	$s_{6+t} + s_{17+t} + k_{477} + s_{2+t}s_{7+t} + s_{9+t}s_{11+t}$
s_{4+t}	Δc_{27}	s_{2+t}
	Δc_7	$s_{7+t} + s_{18+t} + k_{475} + s_{3+t}s_{8+t} + s_{10+t}s_{12+t}$
s_{5+t}	Δc_{30}	s_{7+t}
	Δc_{28}	s_{3+t}
	Δc_{21}	$s_{21+t} + s_{26+t} + k_{484} + s_{19+t}s_{22+t}$
	Δc_8	s_{19+t}
s_{9+t}	Δc_2	s_{11+t}
s_{10+t}	Δc_{12}	s_{12+t}
s_{11+t}	Δc_7	s_{9+t}
s_{12+t}	Δc_{12}	s_{10+t}
s_{19+t}	Δc_{22}	s_{22+t}
s_{20+t}	Δc_{23}	s_{23+t}
s_{21+t}	Δc_{24}	s_{24+t}
s_{22+t}	Δc_{25}	s_{25+t}
s_{23+t}	Δc_{26}	s_{26+t}
	Δc_{12}	s_{20+t}
s_{24+t}	Δc_{27}	s_{27+t}
	Δc_{20}	$s_{7+t} + s_{18+t} + s_{22+t} + s_{27+t} + k_{475} + k_{482} + s_{3+t}s_{8+t} +$ $s_{10+t}s_{12+t} + s_{20+t}s_{23+t} + 1$
	Δc_{13}	s_{21+t}

Table 4. Fault induction after $t = 243$ rounds of KATAN32

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{0+t}	Δc_{21}	$s_{22+t} + s_{27+t} + k_{494} + s_{20+t}s_{23+t}$
s_{1+t}	Δc_{27}	s_{6+t}
	Δc_{22}	$s_{23+t} + s_{28+t} + k_{492} + s_{21+t}s_{24+t}$
	Δc_{20}	s_{3+t}
s_{2+t}	Δc_{28}	s_{7+t}
	Δc_{23}	$s_{24+t} + s_{29+t} + k_{490} + s_{22+t}s_{25+t}$
	Δc_{21}	s_{4+t}
	Δc_{19}	s_{0+t}
s_{3+t}	Δc_{24}	$s_{25+t} + s_{30+t} + k_{488} + s_{23+t}s_{26+t}$
	Δc_{22}	s_{5+t}
	Δc_{20}	s_{1+t}
	Δc_0	$s_{6+t} + s_{17+t} + k_{489} + s_{2+t}s_{7+t} + s_{9+t}s_{11+t}$
s_{4+t}	Δc_{25}	$s_{26+t} + s_{31+t} + k_{486} + s_{24+t}s_{27+t}$
	Δc_{21}	s_{2+t}
	Δc_1	$s_{7+t} + s_{18+t} + k_{487} + s_{3+t}s_{8+t} + s_{10+t}s_{12+t}$
s_{18+t}	Δc_4	s_{21+t}
	Δc_1	$s_{4+t} + s_{15+t} + k_{493} + s_{0+t}s_{5+t} + s_{7+t}s_{9+t}$
s_{19+t}	Δc_5	s_{22+t}
	Δc_2	$s_{5+t} + s_{16+t} + k_{491} + s_{1+t}s_{6+t} + s_{8+t}s_{10+t}$
s_{21+t}	Δc_7	s_{24+t}
s_{22+t}	Δc_8	s_{25+t}
	Δc_5	s_{19+t}
s_{23+t}	Δc_9	s_{26+t}
	Δc_6	s_{20+t}
s_{24+t}	Δc_{10}	s_{27+t}
	Δc_7	s_{21+t}
s_{26+t}	Δc_{20}	$s_{21+t} + s_{23+t} + s_{31+t} + k_{486} + k_{496} + s_{19+t}s_{22+t} + s_{24+t}s_{27+t} + 1$
	Δc_9	s_{23+t}

Table 5. Fault induction after $t = 249$ rounds of KATAN32

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{4+t}	Δc_{19}	$s_{22+t} + s_{26+t} + s_{31+t} + k_{498} + s_{24+t}s_{27+t}$
s_{5+t}	Δc_{20}	s_{0+t}
s_{21+t}	Δc_1	s_{24+t}
s_{23+t}	Δc_3	s_{26+t}
	Δc_0	s_{20+t}
s_{25+t}	Δc_2	s_{22+t}

Table 6 below shows the difference characteristics for faulty rounds $t = 237$ for KATAN32. Denote '0' and '1' as differential values 0 and 1 respectively of the corresponding ciphertext bit differential Δc_j, and '-' as unknown differential value.

Table 6. Difference characteristics for KATAN32 (faulty round t=237)

	Δc_j																													
s_{j+t}	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	-	0	-	0	-	-	0	-	-	0	1	0	0	0	0	0	0	0	0	0	-	0	-	-	-	-	1	-
1	0	0	1	-	0	-	0	-	-	0	-	-	-	1	0	0	0	0	0	0	0	0	0	-	0	-	-	-	-	1	-	-
2	0	1	-	0	-	0	-	-	0	-	-	-	-	0	0	0	0	0	0	0	0	0	-	0	-	-	-	-	1	-	-	-
3	1	-	0	-	-	-	-	0	-	-	-	-	-	0	0	0	0	0	0	-	0	-	0	-	-	-	-	-	-	-	-	-
4	-	0	-	0	-	0	0	0	0	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	-	-	1	-	-	-	-	-
5	0	-	0	-	0	0	0	0	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	-	-	1	-	-	-	-	-	-
6	-	0	-	0	0	0	0	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	-	-	1	-	-	-	-	-	-	0
7	0	-	0	0	-	0	-	-	-	-	-	-	-	0	0	0	0	0	0	1	0	-	-	1	-	-	-	-	-	-	-	-
8	-	0	0	0	-	0	-	0	-	-	-	-	-	0	0	0	0	0	0	-	0	-	0	-	-	-	-	-	-	-	-	-
9	0	0	0	0	0	1	0	-	0	-	-	-	-	0	0	0	0	0	0	0	0	0	-	0	-	-	-	-	0	-	-	-
10	0	0	0	0	-	0	-	0	-	-	-	-	-	0	0	0	0	0	0	-	0	-	0	-	-	-	-	-	-	-	-	-
11	0	0	0	1	0	0	0	-	0	-	-	-	-	0	0	0	0	0	0	0	0	0	-	0	-	-	0	0	-	-	-	-
12	0	0	1	0	-	0	-	0	-	-	-	-	-	0	0	0	0	0	0	-	0	-	0	-	-	0	0	-	-	-	-	-
13	0	1	0	0	0	0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	0	1	-	0	0
14	1	0	0	0	0	0	0	0	0	0	-	0	-	0	0	0	0	0	0	0	0	0	0	0	0	-	0	1	-	0	0	0
15	0	0	0	0	0	0	0	0	0	-	0	-	-	0	0	0	0	0	0	0	0	0	0	0	-	0	1	-	0	0	0	1
16	0	0	0	0	0	0	0	0	-	0	-	-	-	0	0	0	0	0	0	0	0	0	0	-	0	1	-	0	0	0	1	0
17	0	0	0	0	0	0	0	-	0	-	-	-	-	0	0	0	0	0	0	0	0	0	-	0	1	-	0	0	0	1	-	0
18	0	0	0	0	-	0	-	0	-	-	-	-	-	0	0	0	0	0	0	1	0	-	0	1	-	0	0	-	1	-	0	-
19	0	0	0	-	0	-	0	-	-	-	-	1	-	0	0	0	0	0	1	0	-	0	1	-	0	0	0	1	-	0	-	-
20	0	0	-	0	-	0	-	-	-	-	1	-	-	0	0	0	0	1	0	-	0	1	-	0	0	0	1	-	0	-	-	-
21	0	0	0	-	0	-	-	-	-	1	-	-	-	0	0	0	0	0	-	0	1	-	0	0	0	1	0	0	-	0	-	-
22	-	0	-	0	-	-	-	-	1	-	-	-	-	0	0	1	0	-	0	1	-	0	0	0	1	-	0	-	-	-	-	-
23	0	-	0	-	-	-	-	1	-	-	-	-	-	0	0	0	-	0	1	-	0	0	0	1	0	0	-	-	-	-	-	-
24	-	0	-	-	-	-	1	-	-	-	-	-	-	0	0	-	0	1	-	0	0	0	1	0	0	-	0	-	-	-	-	-
25	0	-	-	0	0	1	-	-	-	-	-	-	-	0	0	0	1	-	0	0	0	1	0	0	0	-	-	-	-	-	-	-
26	-	-	0	0	1	-	-	-	-	-	-	-	-	0	0	1	-	0	0	0	1	0	0	0	0	-	-	-	-	-	-	-
27	-	0	0	0	-	-	0	-	1	-	-	-	-	0	0	-	0	0	0	1	0	0	0	0	0	-	0	-	-	-	-	-
28	0	0	0	-	0	0	0	1	-	0	-	0	-	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-	0	-	-
29	0	0	-	0	0	0	1	-	0	-	0	-	-	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-	0	-	-	-
30	0	-	0	0	0	1	-	0	-	0	-	-	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-	0	-	-	-	-
31	-	0	0	0	1	-	0	-	0	-	-	0	-	0	0	1	0	0	0	0	0	0	0	0	0	-	0	-	-	-	-	1

Tables 7, 8, 9, 10 and 11 show the polynomial equations resulting from proper and faulty ciphertext differential for faulty rounds $t = 234, 238, 242, 246, 250$ for KATAN48.

Table 7. Fault induction after t=234 rounds of KATAN48

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{6+t}	Δc_{26}	s_{15+t}
	Δc_{25}	s_{14+t}
s_{9+t}	Δc_{28}	s_{17+t}
s_{11+t}	Δc_{18}	$s_{19+t} + 1$
s_{29+t}	Δc_{41}	s_{37+t}
s_{30+t}	Δc_{42}	s_{38+t}

Table 8. Fault induction after t=238 rounds of KATAN48

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{4+t}	Δc_9	$s_{12+t} + 1$
	Δc_{16}	s_{13+t}
s_{5+t}	Δc_{17}	s_{14+t}
s_{6+t}	Δc_{24}	s_{15+t}
s_{8+t}	Δc_{47}	s_{0+t}
	Δc_{13}	s_{16+t}
s_{9+t}	Δc_{14}	s_{17+t}
s_{10+t}	Δc_{15}	s_{18+t}
s_{11+t}	Δc_{16}	s_{19+t}
s_{12+t}	Δc_{17}	s_{20+t}
	Δc_{16}	s_{29+t}
	Δc_{15}	s_{3+t}
s_{13+t}	Δc_{17}	s_{30+t}
s_{14+t}	Δc_{18}	s_{31+t}
	Δc_{17}	s_{5+t}
s_{15+t}	Δc_{19}	s_{32+t}
	Δc_6	s_{7+t}
s_{16+t}	Δc_{20}	s_{33+t}
	Δc_{13}	s_{8+t}
s_{17+t}	Δc_{38}	$s_{29+t} + s_{35+t} + s_{41+t} + k_{482} + s_{30+t}s_{38+t}$
	Δc_{21}	s_{34+t}
	Δc_{14}	s_{9+t}
	Δc_{12}	$s_{16+t} + s_{25+t} + k_{479} + s_{3+t}s_{12+t} + s_{10+t}s_{18+t}$
s_{18+t}	Δc_{22}	s_{35+t}
	Δc_{15}	s_{10+t}
s_{19+t}	Δc_{46}	s_{1+t}
	Δc_{40}	$s_{31+t} + s_{37+t} + s_{43+t} + k_{480} + s_{32+t}s_{40+t}$
	Δc_{14}	$s_{18+t} + s_{27+t} + k_{477} + s_{5+t}s_{14+t} + s_{12+t}s_{20+t}$
s_{29+t}	Δc_{33}	s_{37+t}
s_{30+t}	Δc_{25}	s_{38+t}
	Δc_{17}	$s_{13+t} + s_{22+t} + k_{483} + s_{0+t}s_{9+t} + s_{7+t}s_{15+t}$
s_{31+t}	Δc_{36}	$s_{33+t} + s_{39+t} + s_{45+t} + k_{478} + s_{34+t}s_{42+t} + 1$
	Δc_{35}	s_{39+t}
	Δc_{18}	$s_{14+t} + s_{23+t} + k_{481} + s_{1+t}s_{10+t} + s_{8+t}s_{16+t}$
	Δc_7	s_{4+t}
	Δc_1	$s_{4+t} + s_{40+t} + s_{46+t} + k_{476} + s_{35+t}s_{43+t}$
s_{32+t}	Δc_{36}	s_{40+t}
s_{33+t}	Δc_{37}	s_{41+t}
s_{34+t}	Δc_{38}	s_{42+t}
s_{35+t}	Δc_{39}	s_{43+t}

Table 9. Fault induction after t=242 rounds of KATAN48

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{6+t}	Δc_{45}	s_{14+t}
	Δc_{10}	s_{15+t}
s_{12+t}	Δc_9	$s_{20+t} + 1$
	Δc_8	s_{29+t}
	Δc_7	s_{3+t}
s_{15+t}	Δc_{46}	s_{7+t}
s_{16+t}	Δc_{47}	s_{8+t}
	Δc_{12}	s_{33+t}
s_{17+t}	Δc_{13}	s_{34+t}
	Δc_6	s_{9+t}
s_{18+t}	Δc_{14}	s_{35+t}
	Δc_7	s_{10+t}
s_{19+t}	Δc_{15}	s_{36+t}
s_{30+t}	Δc_{17}	s_{38+t}
	Δc_9	$s_{13+t} + s_{22+t} + k_{491} + s_{0+t}s_{9+t} + s_{7+t}s_{15+t}$
s_{31+t}	Δc_{35}	$s_{40+t} + s_{46+t} + k_{484} + s_{35+t}s_{43+t}$
	Δc_{29}	$s_{34+t} + s_{40+t} + k_{490} + s_{29+t}s_{37+t}$
	Δc_{18}	s_{39+t}
	Δc_{10}	$s_{14+t} + s_{23+t} + k_{489} + s_{1+t}s_{10+t} + s_{8+t}s_{16+t}$
s_{33+t}	Δc_{31}	$s_{36+t} + s_{42+t} + k_{488} + s_{31+t}s_{39+t}$
	Δc_{29}	s_{41+t}
	Δc_{12}	$s_{16+t} + s_{25+t} + k_{487} + s_{3+t}s_{12+t} + s_{10+t}s_{18+t}$
s_{34+t}	Δc_{38}	s_{1+t}
	Δc_{30}	s_{42+t}
s_{35+t}	Δc_{33}	$s_{38+t} + s_{44+t} + k_{486} + s_{33+t}s_{41+t}$
	Δc_{31}	s_{43+t}
	Δc_{14}	$s_{18+t} + s_{27+t} + k_{485} + s_{5+t}s_{14+t} + s_{12+t}s_{20+t}$
s_{36+t}	Δc_{32}	s_{44+t}
s_{39+t}	Δc_{37}	s_{0+t}
	Δc_{36}	s_{5+t}
	Δc_{18}	s_{31+t}

Table 10. Fault induction after t=246 rounds of KATAN48

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{2+t}	Δc_{33}	s_{10+t}
	Δc_{31}	$s_{29+t} + s_{35+t} + s_{41+t} + k_{498} + s_{30+t}s_{38+t}$
s_{3+t}	Δc_{41}	s_{12+t}
	Δc_{32}	$s_{36+t} + s_{42+t} + k_{496} + s_{31+t}s_{39+t}$
s_{5+t}	Δc_{43}	s_{14+t}
	Δc_{36}	s_{13+t}
	Δc_{34}	$s_{38+t} + s_{44+t} + k_{494} + s_{33+t}s_{41+t}$
s_{8+t}	Δc_{39}	s_{16+t}
	Δc_{37}	$s_{41+t} + s_{47+t} + k_{492} + s_{36+t}s_{44+t}$
	Δc_{31}	s_{0+t}
s_{10+t}	Δc_{41}	s_{18+t}
	Δc_{39}	s_{1+t}
s_{12+t}	Δc_0	s_{29+t}
s_{13+t}	Δc_{44}	s_{21+t}
	Δc_1	s_{30+t}
s_{14+t}	Δc_2	s_{31+t}
s_{15+t}	Δc_{44}	s_{6+t}
	Δc_{38}	s_{7+t}
s_{16+t}	Δc_4	s_{33+t}
s_{18+t}	Δc_6	s_{35+t}
s_{19+t}	Δc_7	s_{36+t}
s_{30+t}	Δc_9	s_{38+t}
	Δc_1	$s_{13+t} + s_{22+t} + k_{499} + s_{0+t}s_{9+t} + s_{7+t}s_{15+t}$
s_{31+t}	Δc_{10}	s_{39+t}
	Δc_2	$s_{14+t} + s_{23+t} + k_{497} + s_{1+t}s_{10+t} + s_{8+t}s_{16+t}$
s_{33+t}	Δc_{12}	s_{41+t}
	Δc_4	$s_{16+t} + s_{25+t} + k_{495} + s_{3+t}s_{12+t} + s_{10+t}s_{18+t}$
s_{36+t}	Δc_{32}	s_{3+t}
	Δc_{15}	s_{44+t}
	Δc_7	$s_{19+t} + s_{28+t} + k_{493} + s_{6+t}s_{15+t} + s_{13+t}s_{21+t}$

Table 11. Fault induction after t=250 rounds of KATAN48

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{8+t}	Δc_{29}	$s_{41+t} + s_{47+t} + k_{500} + s_{36+t}s_{44+t}$
s_{44+t}	Δc_7	s_{36+t}

Tables 12, 13, 14, 15 and 16 show the polynomial equations resulting from proper and faulty ciphertext differential for faulty rounds $t = 236, 238, 242, 246, 250$ for KATAN64.

Table 12. Fault induction after t=236 rounds of KATAN64

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{8+t}	Δc_{37}	s_{3+t}
s_{16+t}	Δc_{38}	s_{28+t}
s_{20+t}	Δc_{36}	$s_{32+t} + s_{42+t}$
s_{28+t}	Δc_{38}	s_{16+t}
s_{30+t}	Δc_{34}	s_{18+t}
s_{31+t}	Δc_{35}	s_{19+t}
s_{33+t}	Δc_{36}	s_{42+t}
s_{45+t}	Δc_{61}	$s_{2+t} + s_{54+t}$
s_{46+t}	Δc_{62}	$s_{3+t} + s_{55+t}$
	Δc_{25}	s_{2+t}
s_{54+t}	Δc_{61}	$s_{2+t} + s_{45+t}$

Table 13. Fault induction after t=238 rounds of KATAN64

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{2+t}	Δc_{55}	$s_{39+t} + s_{45+t} + s_{54+t} + k_{482} + s_{41+t}s_{50+t}$
	Δc_{10}	$s_{52+t} + s_{61+t} + k_{476} + s_{48+t}s_{57+t}$
s_{5+t}	Δc_{27}	s_{10+t}
	Δc_{13}	s_{0+t}
s_{6+t}	Δc_{28}	s_{11+t}
s_{8+t}	Δc_{16}	s_{3+t}
s_{10+t}	Δc_{27}	s_{5+t}
s_{17+t}	Δc_{27}	$s_{29+t} + s_{39+t}$
s_{18+t}	Δc_{28}	$s_{30+t} + s_{40+t}$
s_{21+t}	Δc_{37}	s_{33+t}
	Δc_{19}	s_{9+t}
	Δc_{18}	$s_{17+t} + s_{30+t} + k_{481} + s_{1+t}s_{6+t} + s_{13+t}s_{25+t} + 1$
s_{24+t}	Δc_{53}	s_{0+t}
	Δc_{34}	s_{46+t}
	Δc_{22}	s_{12+t}
s_{25+t}	Δc_{54}	s_{1+t}
	Δc_{35}	s_{47+t}
	Δc_{26}	$s_{16+t} + s_{29+t} + k_{483} + s_{0+t}s_{5+t} + s_{12+t}s_{24+t}$
	Δc_{23}	s_{13+t}
	Δc_{22}	$s_{21+t} + s_{34+t} + k_{479} + s_{5+t}s_{10+t} + s_{17+t}s_{29+t} + 1$
s_{26+t}	Δc_{24}	s_{14+t}
s_{27+t}	Δc_{25}	s_{15+t}
s_{28+t}	Δc_{39}	s_{16+t}
s_{29+t}	Δc_{40}	s_{17+t}
s_{31+t}	Δc_{28}	s_{40+t}
s_{32+t}	Δc_{36}	s_{20+t}
s_{33+t}	Δc_{37}	s_{21+t}
	Δc_{30}	s_{42+t}
s_{36+t}	Δc_{33}	s_{45+t}

Table 13 (Continued)

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{39+t}	Δc_{55}	s_{2+t}
	Δc_{36}	s_{48+t}
s_{41+t}	Δc_{38}	s_{50+t}
s_{43+t}	Δc_{59}	s_{6+t}
	Δc_{53}	$s_{0+t} + s_{52+t}$
s_{45+t}	Δc_{55}	$s_{2+t} + s_{54+t}$
s_{46+t}	Δc_{56}	$s_{3+t} + s_{55+t}$
s_{47+t}	Δc_{44}	$s_{56+t} + 1$
	Δc_{35}	$s_{25+t} + s_{38+t} + k_{477} + s_{9+t}s_{14+t} + s_{21+t}s_{33+t}$
s_{48+t}	Δc_{49}	$s_{39+t} + s_{45+t} + s_{51+t} + s_{60+t} + k_{478} + s_{47+t}s_{56+t}$
	Δc_{45}	$s_{57+t} + 1$
	Δc_{36}	s_{39+t}
s_{50+t}	Δc_{63}	$s_{40+t} + s_{46+t} + s_{55+t} + k_{480} + s_{42+t}s_{51+t}$
	Δc_{38}	s_{41+t}

Table 14. Fault induction after t=242 rounds of KATAN64

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{3+t}	Δc_{56}	s_{15+t}
	Δc_{13}	s_{8+t}
s_{5+t}	Δc_{58}	s_{17+t}
	Δc_{15}	s_{10+t}
s_{7+t}	Δc_{60}	s_{19+t}
	Δc_{17}	s_{12+t}
s_{8+t}	Δc_{4}	s_{3+t}
s_{17+t}	Δc_{58}	s_{5+t}
s_{19+t}	Δc_{60}	s_{7+t}
	Δc_{17}	$s_{31+t} + s_{41+t}$
s_{24+t}	Δc_{22}	s_{46+t}
s_{25+t}	Δc_{48}	$s_{43+t} + s_{52+t} + k_{490} + s_{39+t}s_{48+t}$
	Δc_{23}	s_{47+t}
	Δc_{14}	$s_{16+t} + s_{29+t} + k_{491} + s_{0+t}s_{5+t} + s_{12+t}s_{24+t}$
	Δc_{10}	$s_{21+t} + s_{34+t} + k_{487} + s_{5+t}s_{10+t} + s_{17+t}s_{29+t} + 1$
s_{28+t}	Δc_{14}	s_{16+t}
s_{34+t}	Δc_{19}	s_{43+t}
s_{39+t}	Δc_{24}	s_{48+t}
s_{40+t}	Δc_{25}	s_{49+t}
s_{41+t}	Δc_{51}	$s_{46+t} + s_{55+t} + k_{488} + s_{42+t}s_{51+t}$
	Δc_{39}	$s_{50+t} + s_{53+t} + s_{62+t} + k_{484} + s_{49+t}s_{58+t}$
	Δc_{26}	s_{50+t}
	Δc_{17}	$s_{19+t} + s_{32+t} + k_{489} + s_{3+t}s_{8+t} + s_{15+t}s_{27+t}$
s_{45+t}	Δc_{55}	$s_{50+t} + s_{59+t} + k_{486} + s_{46+t}s_{55+t}$
	Δc_{21}	$s_{23+t} + s_{36+t} + k_{485} + s_{7+t}s_{12+t} + s_{19+t}s_{31+t}$
s_{46+t}	Δc_{31}	s_{55+t}
s_{47+t}	Δc_{32}	s_{56+t}
s_{48+t}	Δc_{24}	s_{39+t}
s_{49+t}	Δc_{34}	s_{58+t}
s_{50+t}	Δc_{60}	s_{0+t}
	Δc_{35}	s_{59+t}
	Δc_{26}	s_{41+t}

Table 15. Fault induction after t=246 rounds of KATAN64

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{3+t}	Δc_{56}	s_{8+t}
	Δc_{44}	s_{15+t}
s_{4+t}	Δc_{57}	s_{9+t}
	Δc_{45}	s_{16+t}
s_{10+t}	Δc_{58}	s_{5+t}
s_{11+t}	Δc_{52}	s_{23+t}
	Δc_{40}	$s_{54+t} + s_{63+t} + k_{492} + s_{50+t}s_{59+t}$
s_{15+t}	Δc_{56}	s_{27+t}
	Δc_{44}	s_{3+t}
s_{16+t}	Δc_{45}	s_{4+t}
s_{19+t}	Δc_{60}	s_{31+t}
	Δc_{48}	s_{7+t}
s_{20+t}	Δc_{61}	s_{32+t}
s_{23+t}	Δc_{9}	s_{45+t}
s_{30+t}	Δc_{3}	s_{39+t}
s_{33+t}	Δc_{6}	s_{42+t}
s_{38+t}	Δc_{2}	$s_{16+t} + s_{29+t} + k_{499} + s_{0+t}s_{5+t} + s_{12+t}s_{24+t}$
s_{41+t}	Δc_{5}	$s_{19+t} + s_{32+t} + k_{497} + s_{3+t}s_{8+t} + s_{15+t}s_{27+t}$
s_{42+t}	Δc_{15}	s_{51+t}
	Δc_{6}	$s_{20+t} + s_{33+t} + k_{495} + s_{4+t}s_{9+t} + s_{16+t}s_{28+t}$
s_{45+t}	Δc_{43}	$s_{50+t} + s_{59+t} + k_{494} + s_{46+t}s_{55+t}$
	Δc_{18}	s_{54+t}
	Δc_{9}	$s_{23+t} + s_{36+t} + k_{493} + s_{7+t}s_{12+t} + s_{19+t}s_{31+t}$
s_{50+t}	Δc_{48}	s_{0+t}
	Δc_{39}	$s_{46+t} + s_{55+t} + k_{496} + s_{42+t}s_{51+t}$
	Δc_{23}	s_{59+t}
	Δc_{14}	s_{41+t}
s_{55+t}	Δc_{19}	s_{46+t}
s_{59+t}	Δc_{43}	$s_{39+t} + s_{45+t} + s_{54+t} + k_{498} + s_{41+t}s_{50+t}$
	Δc_{23}	s_{50+t}

Table 16. Fault induction after t=250 rounds of KATAN64

Faulty Bit	Ciphertext Bit Differential	Polynomial Equations
s_{3+t}	Δc_{39}	$s_{53+t} + s_{62+t} + k_{500} + s_{49+t}s_{58+t}$
s_{40+t}	Δc_{1}	s_{49+t}