Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers

Gregory V. Bard[1]*, Nicolas T. Courtois[2], Jorge Nakahara Jr[3]**,
Pouyan Sepehrdad[3], and Bingsheng Zhang[4]***

[1] Fordham University, NY, USA
[2] University College London, Gower Street, London, WC1E 6BT, UK
[3] EPFL, Lausanne, Switzerland
[4] Cybernetica AS, Estonia and University of Tartu, Estonia
bard@fordham.edu, n.courtois@ucl.ac.uk,
{jorge.nakahara, pouyan.sepehrdad}@epfl.ch, zhang@ut.ee

Abstract. This paper presents the first results on AIDA/cube, algebraic and side-channel attacks on variable number of rounds of all members of the KATAN family of block ciphers. Our cube attacks reach 60, 40 and 30 rounds of KATAN32, KATAN48 and KATAN64, respectively. In our algebraic attacks, we use SAT solvers as a tool to solve the quadratic equations representation of all KATAN ciphers. We introduced a novel pre-processing stage on the equations system before feeding it to the SAT solver. This way, we could break 79, 64 and 60 rounds of KATAN32, KATAN48, KATAN64, respectively. We show how to perform side channel attacks on the **full 254-round** KATAN32 with one-bit information leakage from the internal state by cube attacks. Finally, we show how to reduce the attack complexity by combining the cube attack with the algebraic attack to recover the full 80-bit key. Further contributions include new phenomena observed in cube, algebraic and side-channel attacks on the KATAN ciphers. For the cube attacks, we observed that the same maxterms suggested more than one cube equation, thus reducing the overall data and time complexities. For the algebraic attacks, a novel pre-processing step led to a speed up of the SAT solver program. For the side-channel attacks, 29 linearly independent cube equations were recovered after 40-round KATAN32. Finally, the combined algebraic and cube attack, a leakage of key bits after 71 rounds led to a speed up of the algebraic attack.

Keywords: algebraic, cube, side-channel attacks, cryptanalysis, lightweight block ciphers for RFID tags

* This work was supported by the US National Science Foundation Grant No. DMS-0821725 to Prof. William A. Stein and the SAGE community. The computing power provided yielded the results of Section 3.

** This work was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the SNF under grant number 5005-67322.

*** This work is supported by Estonian Science Foundation, grant #8058 and #8124, the European Regional Development Fund through the Estonian Center of Excellence in Computer Science (EXCS) and ICT doctoral school.

1 Introduction

This paper describes our findings of cube attacks [16], also known as AIDA [30], algebraic [11] and side-channel (cube) attacks [17] applied to a variable number of round of all members of the KATAN family of block ciphers [13]. As far as we are aware of, this is the first paper detailing these attacks on the KATAN ciphers.

The cube attack is a kind of algebraic technique that exploits the existence of low-degree polynomial equations in the output of cryptographic algorithms. An attractive feature of the cube attack is that it requires only black-box access to the cryptographic function, that is, the knowledge of internal details of the target function is not required. Informally, if the decomposition of algebraic multivariate polynomial equations at the output of a target cipher has degree at most $d+1$, then linear equations on unknown key bits can potentially be extracted, provided that at most 2^d computations (encryptions) are feasible. Thus, the basic setting is key-recovery, but distinguish-from-random variants have been demonstrated in [2]. In [16] the cube attack has been applied to reduced-round variants of the Trivium [14] stream cipher.

Algebraic cryptanalysis exploit the multivariate polynomial system of equations in ANF format representing a given cipher [11]. The aim is to solve such systems for the unknown key (usually in a known-plaintext setting, but often we use chosen-plaintext attack to reduce the running time). Typically, quadratic equations are the main target representation. We convert the ANF equations to CNF format and feed it to a SAT solver, in our case MiniSat [18] and CryptoMiniSat [26]. Furthermore, we introduce a novel pre-processing step on the system of equations before giving it to a SAT solver. This allows us to break a larger number of rounds. Ultimately, we combine the cube and algebraic attacks on reduced-round KATAN ciphers.

We also combined the cube and side-channel techniques to attack the full-round KATAN32, following the model in [17]. Table 4 summarizes all the attack complexities in this paper. Up to the moment, we know of no other independent attacks on the KATAN ciphers, even for reduced-round versions.

This paper is organized as follows: Sect. 2 briefly describes the KATAN family of block ciphers; Sect. 3 provides some theoretical background on algebraic attacks; Sect. 4 gives theoretical framework and our experimental findings on AIDA/cube attacks; Sect. 5 combines both attacks; Sect. 6 describes side-channel cube attack; Sect. 7 concludes the paper.

2 The KATAN Family of Block Ciphers

KATAN is a family of lightweight, hardware-oriented block ciphers consisting of three variants with 32, 48 and 64-bit blocks. For all KATAN ciphers, key size is of 80 bits ($n = 80$), and they all iterate 254 rounds [13]. The block size is used as suffix to designate each cipher member, as KATAN32, KATAN48 and KATAN64. The design of these ciphers was inspired by the stream cipher Trivium [14]. The structure of KATAN32 cipher consists of two LFSR's, called L_1 and L_2, loaded with the plaintext and then transformed by two nonlinear Boolean functions, f_a and f_b as follows (Table 1 lists the bit sizes and the indices x_i and y_j of L_1 and L_2).

$$f_a(L_1) = L_1[x_1] + L_1[x_2] + (L_1[x_3] \cdot L_1[x_4]) + L_1[x_5] \cdot \mathrm{IR}) + k_a$$
$$f_b(L_2) = L_2[y_1] + L_2[y_2] + (L_2[y_3] \cdot L_2[y_4]) + L_2[y_5] \cdot L_2[y_6]) + k_b$$

where IR is the output of an LFSR i.e. $L_1[x_5]$ is used whenever $\mathrm{IR} = 1$. The values of IR for each round is specified in [13]. For the i-th round, $k_a = k_{2i}$ and $k_b = k_{2i+1}$ that is, only two key bits are used per round. The output of each of these functions is loaded to the least significant bits (LSB) of the other LFSR, after they are left-shifted. This operation is performed in an invertible manner.

For KATAN48, f_a and f_b are each applied twice per round, so that the LFSR's are clocked twice (but the same pair of key bits are reused). For KATAN64, each Boolean function is applied three times per round, again with the same pair of key bits reused three times.
The selection of bits x_i and y_i in f_a and f_b are listed in Table 1. In this report, plaintext and ciphertext bits are numbered in **right-to-left order** starting from 0. Thus, for instance, a plaintext block for KATAN32 will be numbered as $p = (p_{31}, \ldots, p_0)$. The key schedule algorithm of all KATAN ciphers is a linear mapping that expands an 80-bit key K to 508 subkey bits according to

Table 1. Parameters for the f_a and f_b functions.

Cipher	$\lvert L_1\rvert$	$\lvert L_2\rvert$	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

$$k_i = \begin{cases} K_i, & \text{for } 0 \leq i \leq 79 \\ k_{i-80} + k_{i-61} + k_{i-50} + k_{i-13}, & \text{otherwise} \end{cases}$$

Thus, the subkey of the i-th round is $k_a || k_b = k_{2i} || k_{2i+1}$.

After r rounds, at most $2 * r$ key bits are mixed with the internal state since two key bits are xored per round. Thus, at least 40 rounds are needed before complete key diffusion for any KATAN cipher is achieved. Further details about these ciphers can be found in [13].

For analyses purposes, the numbering of the key bits in the user key in our attacks is $K = (K_{79}, \ldots, K_0)$.

3 Algebraic Attacks Using SAT Solvers

Algebraic cryptanalysis is a type of cryptographic attack that relies on solving a multivariate polynomial representation of a given cipher or hash function. It was initially formulated as early as 1949 by Shannon [29]. Algebraic attacks, since the controversial paper of [11], have been applied to several stream ciphers [1,9,10] and is able to break some of them but it has not been successful in breaking real-life block ciphers, except Keeloq [7,21]. Compared to statistical analysis, such as linear and differential cryptanalysis, algebraic attacks require a comparatively small number of text pairs. The adversary formulates the cipher as a polynomial multivariate system of equations. This representation is usually over small finite fields like $\mathbf{GF}(2)$. This system of equations is often sparse, since efficient implementations of real-world systems require a low gate-count. In the subsequent stage, the adversary solves the system. The problem of solving such system is NP hard in general and is recognized as the MQ problem. An instance of an MQ problem is a set of functions

$$f_1(x_1,\ldots,x_n)=y_1, \quad f_2(x_1,\ldots,x_n)=y_2 \quad ,\ldots, \quad f_m(x_1,\ldots,x_n)=y_m$$

where f_i can always be converted to a quadratic polynomial by introducing new variables. Notice that n is the number of variables and m is the number of equations. Let $c=\frac{m}{n}$ denote the degree of “overdefinition” of a system [4]. Hence, $c=1$ denotes exactly a defined system, $c>1$ an overdefined system and $c<1$ an underdefined system. The polynomial representation of most block ciphers are overdefined or if not then $c\approx 1$. It turned out that the more the system is overdefined and sparse the easier it is to be solved [11]. For one instance of KATAN cipher, $c<1$. This because in all versions, the key size is larger than the block size. Thus, more than one pair is required to find the correct key.

There are multiple methods for solving such systems. The traditional method uses the Gröbner basis approach such as Buchberger [6] or F4 [20] and F5 [19] algorithms. The drawback of such methods is memory, implying that after a while the algorithm outputs the result or it crashes due to running out of memory. This is true particularly for large systems, but they are usually faster than other methods for small systems and when the characteristic of field q is not 2. When $q=2$, more efficient methods were proposed, such as converting these equations to Boolean expressions in Conjunctive Normal Form (CNF) [4] and deploying various SAT-solver programs. Other strategies include the XL family [12,11], the recent MutantXL [15,25], ElimLin [8] and the Raddum-Semaev [27] algorithms. We focus on SAT-solver based methods in this paper. From now on we will work only with the field $\mathbf{GF}(2)$. To solve such polynomial system by SAT solvers, the attacker initially converts the system from Algebraic Normal Form (ANF) to CNF. There is an efficient conversion method due to Bard-Courtois-Jefferson [4]. We also use a direct method which we call “local interpolation”. Let assume that the total degree of the equations is at most 6. We proceed as follows:

- If there are equations which contain more than 6 variables, split these long XORs into several shorter XORs with at most 6 variables, by adding extra variables, for example $abc+def+gh=1$ becomes $abc+def=x$ and $x+gh=1$.
- for each equation, convert the Boolean function to CNF and write it explicitly.

The concatenation of these CNFs gives a file with extension .cnf on which we can apply any SAT solver. The magic number 6 originates from [4], and is called the cutting number—sometimes 4, 5, or rarely 7 is optimal instead.

The area of SAT Solving has seen tremendous progress over the last years. Many problems (e.g. in hardware and software verification) and in our application in cryptanalysis that seemed to be completely out of reach a decade ago can now be handled routinely. Besides, new algorithms and better heuristics, refined implementation techniques turned out to be vital for this success. New SAT solvers can now solve large systems in reasonable time. Since 2002, almost each year a SAT Race competition [28] was established. In 2007 and 2010 respectively, MiniSat [18] and CryptoMiniSat [26] won the Gold prizes. We used these two SAT solvers in our analysis but since the timings of MiniSat were faster, we do not report CryptoMinisat results in this paper.

3.1 Straightforward Algebraic Attack on KATAN Using SAT Solvers

One instance of KATAN32 can be represented as 8620 very sparse quadratic equations with 8668 variables, KATAN48 as $24,908$ equations and $24,940$ variables and KATAN64 as $49,324$ equations and $49,340$ variables. As can be observed, the system is underdefined. That is because the key size is larger than block size for all versions. To have a defined or an overdefined system, we need multiple samples.

A summary of our results is in Table 4. We used the “guess and determine” algebraic attack initially proposed in [4]. This implies that we fix t bits of the key and then we show that recovering the other $80-t$ bits is faster than exhaustive search. This is represented in the column titled “Fixed” in Table 4. In fact, we fix t LSB of the key, since heuristically we obtained better results than fixing the t MSB of the key. We used the graph partitioning method by Wong and Bard [31] to derive the best state variables to fix, but it did not bring about anything better than using the heuristic of fixing the least t significant bits of the key. Note, if we fix t bits, the algebraic attack is solving a system of equations to recover the $80-t$ remaining bits.

We represent the time complexity of the SAT Solver (MiniSat) in seconds using a 3 Ghz CPU. Note that our algebraic attacks are in the chosen-plaintext scenario, except in some rare cases as noted. We noticed that chosen-plaintext attack is much stronger against KATAN family than known-plaintext (KP) attack. In our attacks, we followed the following structure for the chosen plaintexts for KATAN32: $p_{i+1} = ((p_i \gg 19)+1) \ll 19$ and $p_{i+1} = ((p_i \gg 29)+1) \ll 29$ for KATAN48 and $p_{i+1} = ((p_i \gg 39)+1) \ll 39$ for KATAN64 for $i \geq 1$, where p_i is the i-th plaintext we pick and p_1 can be arbitrary. Note that bits $19,29,39$ are exactly bit 0 of L_1 register for KATAN32, KATAN48 and KATAN64 respectively. This choice of the bits makes the SAT solver run faster. Moreover, we believe it is fair to assume each round encryption of KATAN takes at least 3 CPU cycles. This yields a comparison between the complexity of our attacks and exhaustive key search.

Deploying the straightforward method of converting ANF to CNF and then feeding it to a SAT solver, we could break up to 79 rounds of KATAN32 and 64 rounds of KATAN48 and 60 rounds of KATAN64. But, we can do better by performing a pre-processing on the system of equations before applying it to a SAT solver. Using this pre-processing (see next section), we could break 79 rounds of KATAN32. We only tried this method on KATAN32 equations. Further research would apply this technique to other members of the family.

3.2 The Pre-processing SAT-Solver Attack

In this attack, we use the equations generated as described earlier and solve them with the SAT solver MiniSat [18]. It is simpler to formulate KATAN as a sparse system. But, this may not be the best representation for a SAT solver. One characteristic of these equations is that there are many of them with the form $x = y$, as well as $x = 0$, $y = 1$ and more rarely $x + y = 1$. Also, in a typical example (78 rounds, 45 key bits fixed and 20 CPs of KATAN32) there are 51,321 total equations. Naturally one wants to take advantage of these special equations, during pre-processing, to create a smaller system which has fewer variables and equations.

More precisely, the four heuristics of a CNF problem are (1) the number of variables, (2) the number of clauses, (3) the average number of symbols per clause, and (4) the total number of symbols in the system. The pre-processing algorithm that we describe in the next section is designed on the principle of primarily reducing (1) and (2) while causing the minimum possible increase in (3) and (4). To be specific, at each iteration, a substitution will be made and this substitution reduces (1) and (2) by one, and the substitution is selected in the style of the "greedy algorithm" using (4) as the criterion.

The following pre-processing algorithm, due to Bard, is a refinement of the "massaging" algorithm of [4] and so we call it "turbo-massage". Starting with the equations that were generated, we ran the pre-processing algorithm; after that, we converted the polynomials into a CNF problem, according to [4] and ran MiniSat on that CNF problem to get a solution. We will explain the pre-processors here and refer the reader to [4] or [3] for the process of converting a polynomial system into a CNF problem.

3.3 The Turbo-Massage Pre-processing Algorithm

As described before, the equations can be thought of as a series of polynomials $f_1(x) = 0$, $f_2(x) = 0$, ... We define the operation "fully-substitute" as follows: Let $f(x)$ be a polynomial with some monomial μ. To fully-substitute $f(x)$ into $g(x)$ on μ means to

- Write $g(x)$ in the form $g(x) = \mu h_1(x) + h_2(x)$.
- Write $f(x)$ in the form $f(x) = \mu + h_3(x)$.
- Replace $g(x)$ with $h_1(x)h_3(x) + h_2(x)$, which is mathematically equivalent, because in any satisfying solution x, we would have $\mu = h_3(x)$.
- By clever use of data structures, this can be made highly efficient.

Observe that for the four common forms: $x = y$, as well as $x = 0$, $y = 1$, and more rarely $x + y = 1$, the "fully-substitute" definition does what one would do if solving a system of equations with a pencil and paper. For more higher weight $f(x)$, understanding what it does to $g(x)$ is more complex.

We must also use a non-standard definition of the weight of a polynomial $f(x)$. We define it to be the number of monomials in $f(x)$, but excluding the constant $+1$ from the tabulation. The reason for this is that if $f(x)$ has weight w according to this modified definition, then 2^{w-1} conjunctive normal form clauses of length w will be required to represent the polynomial, assuming all the monomials are already defined. The total number of symbols is then $w2^{w-1}$ and so minimizing w is crucial in keeping the CNF problem small and thus solvable. We now perform the following algorithm:

- **INPUT**: A system of polynomial equations over $\mathbf{GF}(2)$, and a weight-limit w_{max}.
- Mark all polynomials "unused."
- While the set of unused polynomials is not empty do:
  - Locate the lowest weight unused polynomial $f(x)$.
  - If $f(x)$ exceeds w_{max}, terminate.
  - Mark $f(x)$ as "used."
  - If $f(x)$ has weight 1, then select μ to be the only monomial in $f(x)$.
  - If $f(x)$ has weight 3 or higher, select μ to be the monomial which appears least frequently in the entire system of equations.
  - If $f(x)$ has weight exactly 2, select μ to be the monomial which appears most frequently in the entire system of equations.
  - For any polynomial $g(x)$ containing the monomial μ, simply "fully-substitute" $f(x)$ into $g(x)$ on μ.

The "turbo-massaging" algorithm will always terminate, because eventually every polynomial has been marked used. In practice, it will terminate early, where all unused polynomials are of weight w_{max} or higher. If there are n "used" polynomials, then there will be n monomials which appear nowhere in the entire system except in exactly one polynomial. This is, of course, the monomial μ which was chosen when that polynomial was getting used. In our system of equations, it was almost always the case (by an overwhelming margin) that μ was degree one. And so, each used polynomial effectively amputates one variable from the polynomial system of equations.

The special case of weight 2 deserves explanation. When f has weight 1, there is no decision to be made, but it is noteworthy that the weight of g will decrease. When f has weight 2, then the weight of g will not change during the "fully-substitute" operation, except in some odd cases like substituting $x = y$ into $zx + zy + w + x + y = 0$, where the weight goes from 5 to 1 instantly. Since the weight is not likely to change and we are eliminating a monomial, it makes sense to eliminate a common monomial. When the weight of f is 3 or more, then the weight of g will increase. If we choose μ to be very popular, appearing k times, then the total weight of the system will increase by $k(w-2)$. Thus, it makes sense to keep the weight growth bounded and choose μ to be rare. This heuristic was found after an enormous number of iterations of "trial-and-error."

For example, in the 78-round, 20 CP, 45-bit-key case of KATAN32, the weight went from $110,726$ to $101,516$ after $47,032$ polynomials got used. Furthermore, the "fully substitute" function was called $330,587$ times. There were $50,033$ equations at this point, down from $51,321$, representing $1,288$ equations that became $0 = 0$. In other words, the original system was not full rank. The average weight of a polynomial, using the modified definition of weight, was roughly 2.02898. The system had $53,993$ distinct monomials, plus $2,005$ variables which appeared only in degree 1 monomials, and ended with a CNF problem of $55,398$ variables, and $156,010$ clauses. The conversion process, which must be run only once and not 2^{45} times, takes between 20 and 29 minutes in all the cases explored here.

It should be noted that in other polynomial systems it might be the case that μ is often quadratic or higher degree. It remains open if one should force μ to be linear when possible. This is a question that the authors hope to investigate shortly. As it comes to pass, $w_{max} = 2$ turned out to be slightly better than $w_{max} = 3$ for this problem, but in other cases up to $w_{max} = 5$ has been used.

A minor note for algebraic geometers familiar with the concept of a Macaulay matrix [23] in the Lazard [22] family of algorithms (including F4 [20], XL and their variants [12,11]) is that this algorithm is like a Gaussian Elimination on that matrix, but stopping early. The pivoting strategy used is reducible to the Markowitz pivoting algorithm [24]. However, the "fully-substitute" is not the same in this case, as adding $x = y$ to $zx + zy + w + x + y = 0$ would result in $zx + zy + w = 0$. On the other hand, fully-substituting $x = y$ into $zx + zy + w + x + y = 0$ would result in $zx + zx + w + x + x = 0$ which turns into $w = 0$. As you can see, full-substitution is distinct from adding, and is very similar to what a mathematician would do if solving a system of polynomial equations with a pencil and paper.

3.4 Results

The first result was 76 rounds, 20 CP and 45 (fixed) key bits of KATAN32, broken faster than by brute force. To extend this result, we explored using fewer key bits, and more rounds. First we conducted the above process for 20 CPs, and for 76, 77, 78, 79 and 80 rounds. Every case was run 50 times.

Because we fixed 45 bits of the key, and so assuming one nano-second per round for a brute force attacker, our attack against r rounds is faster than brute force if and only if it runs in t seconds with $2^{45}t < r2^{80}10^{-9}$ or more plainly $t < r2^{35}10^{-9} \approx r(34.3597\cdots)$. We also ran trials with 43 bits of the key fixed for 76 rounds and there the threshold would be 4 times greater or $137.439r$ seconds and for 41 bits of the key $549.755r$ seconds.

The running times are given in Table 2. Observe the enormous variance in each trial. In some cases, the fastest run is 1000× faster than the slowest. This is very typical in SAT-solver-based cryptanalysis. We excluded the three fastest and slowest trials and took the mean and standard deviation of the remaining 44 trials.

The running time of 2^{45} executions, all added together, is the sum of 2^{45} samples from independent random variables. Therefore, the central limit theorem applies and regardless of the actual distribution of running times, if the mean is m_1 and stdev is σ_1, the sum of 2^{45} of them will be normally distributed and have a mean of $2^{45}m_1$ and a standard deviation of $2^{22.5}\sigma_1$. Since σ/m is an important instrument in gauging the reliability of a normal sample, it is interesting to note here that σ/m (for the sum of 2^{45} execution times) would be $2^{-22.5}(\sigma_1/m_1)$ which is phenomenally tiny. Thus, the running time of the real-world attacker would be essentially constant.

Notice, that we claim that the 2^{45} running times are independent, but we do not claim that they are identically distributed. On the other hand, one could conceive of a cipher where one key bit was ignored by the cipher, in which case the running times for two keys which differ only in that bit would be highly dependent. These cases are of pedagogical interest only, because no cipher designer would ever do that.

As can be seen in Table 2, we are between 80.75 and 2.39 times faster than brute force search for up to and including 79 rounds. In the case of 80 rounds, out of 50 trials, 29 of them timed-out after 1 hour. Since this is majority, it is not possible that the mean is less than the required 2748.77 seconds, and so we are not faster than brute-force for 80 rounds. For 43 key bits and 41 key bits, the attack becomes vastly more efficient.

But, we cannot test 39 key bits, as the time-out value would have to be set to 167,125 seconds or roughly 46 hours, for each of 50 processes.

In addition to MiniSat, we ran all 50 instances with CryptoMiniSat [26], a SAT-Solver constructed specifically for cryptography by Mate Soos. However, it was consistently slower than MiniSat. We suspect that this is the case because CryptoMiniSat was intended to minimize the impact of long-XORs, which are normally very damaging to the running time of SAT-solver methods; however, we have no long-XORs in our equations, in fact, no sum was longer than 5 symbols after pre-processing, excluding the constant monomial.

3.5 The Gibrat Hypothesis

In [4], [8] as well as [3], Bard hypothesized that the true distribution of the running times of a CNF-problem in a polynomial-system-based SAT problem follows the Gibrat distribution. That is to say, that the logarithm of the running time is normal. The running times here were such that their standard deviations exceeded the mean. If the distribution of the running time were normal, having $\sigma > \mu$ would imply a very significant fraction of the running times would be negative. Therefore, it is not possible that the running time is normally distributed. On the other hand, we also tabulated the mean and standard deviation of the logarithm.

The ratio of the mean and standard deviation of the logarithm of running times is much more reasonable. The kurtosis is the typical measurement of the "normalness" of a distribution and the kurtosis of the logarithms of the running times are far closer to 1 (and are in fact within ± 1) than the kurtosis of the running times themselves (which had kurtoses over 9). So the hypothesis that the running times are Gibrat, from [4], seems well-justified for these examples.

3.6 A Strange Phenomena

We were perplexed to discover that solving 77 rounds was far easier than solving 76 rounds or 78 rounds. Therefore, we ran the experiments again, with both sets of results listed in the Table 2 as first batch and second batch. As you can see, in both cases, 77 rounds is much easier than 76 or 78—and with a very large margin. Moreover, this remained true as well in our experiments with CryptoMiniSat. As random variables, the *i*th iteration of the 76 round attack and the *i*th iteration of the 77 round attack had absolute correlation of $0.060419\cdots$ and likewise between 77 and 78 it was $-0.09699\cdots$. These extremely low correlations make it safe to hypothesize that the running times are independent and this removes the possibility that the effect is an artifact of some methodology error. Note, the formula for correlation that we used is

$$\mathrm{Cor}(X,Y) = \frac{\mathrm{E}(X-\mu_x)\mathrm{E}(Y-\mu_y)}{\sigma_x \sigma_y}$$

as is standard. Moreover, we observed the same behaviour when dealing with the size of the vertex separator in the variable-sharing graph representation of polynomial system of equations of KATAN32 using the strategy described in [31]. For KATAN32, the size of vertex separator is not increasing with the number of rounds and as a matter of fact it fluctuates. We offer no explanation as to the cause of the weakness of the 77-round version of KATAN32.

4 AIDA/Cube attacks

AIDA/cube attacks [16] are generic key-recovery attacks that can be applied to cryptosystems in a black-box setting, that is, the internal structure of the target cipher is unknown. An important requirement is that the output from the cryptosystem can be represented as a low-degree decomposition multivariate polynomial in Algebraic Normal Form (ANF), called master polynomial, in the key and the plaintext. This attack does not depend on the knowledge of the master polynomial, which may be dense, or whose representation is so large that it cannot even be stored.

Let $p(x_1,\ldots,x_n,v_1,\ldots,v_m)$ denote a master polynomial over $\mathbf{GF}(2)$ in ANF, with x_i, $1 \le i \le n$, the public variables (plaintext, IV bits) and v_j the secret key variables. We assume the adversary is allowed to query the master polynomial at values x_i (that is, a chosen-plaintext, chosen-IV setting) of its choice (these are also called tweakable parameters) and obtain the resulting bit from the master polynomial. This way, the adversary obtains a system of polynomial equations in terms of secret variables only. The ultimate goal of the attack is to solve this system of equations, which reveals the key variables v_j. For this attack, the master polynomial is decomposed as follows:

$$p(x_1,\ldots,x_n,v_1,\ldots,v_m) = t_I \cdot p_{S(I)} + q(x_1,\ldots,x_n,v_1,\ldots,v_m)$$

where t_I is a monomial containing only public variables from an index set $I \subset \{1,2,\ldots,n\}$ called cube or hypercube; '+' stands for bitwise xor; $p_{S(i)}$ is called the superpoly of I in p. The superpoly of I in p does not contain any common variable with t_I and each monomial in q does not contain at least one variable from I, since they have all been factored out in $p_{S(I)}$. The $p_{S(I)}$ of interest are linear mappings in terms of v_j's. Any t_I that leads to a linear $p_{S(I)}$ in key bits is called maxterm. The output of the offline phase of the attack consists of linear equations in the user key bits directly. Further, Gaussian elimination allows one to reconstruct the user key (independent of the key schedule algorithm). For instance, let

$$p(x_1,x_2,x_3,v_1,v_2,v_3,v_4) = x_2.x_3.v_3 + x_1.x_2.v_1 + x_2.v_4 + x_1.x_3.v_2.v_3 + x_1.x_2.v_2 + 1$$

Let $I = \{1,2\}$, so that $t_I = x_1.x_2$ and we have the following decomposition

$$p(x_1,x_2,x_3,v_1,v_2,v_3,v_4) = x_1.x_2.p_{S(i)} + q$$

where $p_{S(I)} = v_1 + v_2$ and $q = x_2.x_3.v_3 + x_2.v_4 + x_1.x_3.v_2.v_3 + 1$.

The main motivation for this decomposition of the master polynomial is that the symbolic sum over $\mathbf{GF(2)}$ of all evaluations of p by assigning all possible binary values to the variables in I (and a fixed value, usually 0, to all the public variables not in I) is exactly $p_{S(I)}$, the superpoly of t_I in p. This is the fundamental theorem in [16]. In the example,

$$\begin{aligned}\bigoplus_{x_i, i \in I} p(x_1,x_2,x_3,v_1,v_2,v_3,v_4) = {} & p(0,0,x_3,v_1,v_2,v_3,v_4)+\\ & p(0,1,x_3,v_1,v_2,v_3,v_4)+\\ & p(1,0,x_3,v_1,v_2,v_3,v_4)+\\ & p(1,1,x_3,v_1,v_2,v_3,v_4) = v_1+v_2 = p_{S(I)}\end{aligned}$$

since $t_I = 0$ whenever either of x_1, x_2 is zero. In q, since each monomial does not contain at least one of the variables in t_I, each monomial will appear an even number of times in the summation of p and the xor sum will be zero.

The cube attack has a pre-processing (offline) and an online phase. In the former, the aim is to find monomials t_I's that lead to linear superpolys. The maxterms are not key dependent, so they need to be computed only once per master polynomial, for a fixed number of rounds. For each maxterm, the adversary computes the coefficients of the v_j's, effectively reconstructing the ANF of the superpoly of each t_I. This step is performed by linearity tests [5]. The main issue in the pre-processing is to find the correct combination of $|I|$ public variables (out of n) x_i that result in **linear superpolys**. Since the exact form of the master polynomial is unknown, this step is heuristic and consists in randomly choosing the cube variables and using linearity tests to check the superpolys. This phase is performed only once for a given cipher and a fixed number of rounds.

Besides the linearity tests there are also 'constant' tests that are used to determine the constant terms 0 or 1 in the superpoly's. The public variables not in the maxterms should be set to the same fixed value in both phases. After a sufficient[5] number of linearly independent (LI) superpolys have been found, the online phase starts by evaluating the superpolys, that is, summing up p over all the values of the corresponding maxterm, $\bigoplus_{x_i, i \in I} p$ and deriving the value of the linear combination of secret v_j bits. If the degree of t_I is d, each xor sum requires 2^d evaluations of p (which implies a chosen-plaintext setting). Thus, the time and data complexities are proportional to the maximum degree d among all maxterms.

The online complexity is proportional to 2^{d_i} encryptions, for a superpoly whose maxterm has d_i variables, since the ciphertexts have to be collected (and xored) for this same amount of chosen plaintexts. If t LI superpolys are available, then $\sum_{i=1}^{t} 2^{d_i}$ encryptions will be needed to recover each superpoly. On the other hand, if the key size is k bits, then 2^{k-t} encryptions shall be enough to recover the remaining unknown part of the key. In total, the time complexity becomes $2^{k-t} + \sum_{i=1}^{t} 2^{d_i}$.

[5] An ideal quantity is a trade-off between the number of linearly independent superpolys and the effort to recover the remaining key bits.

4.1 Cube Attack on KATAN32

Table 5 shows cubes and maxterms for 40-round KATAN32. The maxterm is shown in hexadecimal (the bits set to '1' are the selected bits) for a compact description in the tables in the appendix. We used Gaussian elimination to select LI equations. Experimentally, not all ciphertext bits leak information on the key bits (cube equations). Some ciphertext positions provide larger leakage than others. There are three cubes of degree 15, 36 cubes of degree 16, one cube of degree 19 and four cubes of degree 20. We found that the same maxterm can be used for different key equations, for distinct cipher bits. This means that we can save data and computational complexity during the online phase. For instance, the maxterm $\texttt{41D3D98E}_\texttt{x}$ gives two LI equations: $k_{12}+k_{25}+k_{36}$ and $k_{24}+1$. Thus, the data complexity is $3\cdot 2^{15}+35\cdot 2^{16}+2^{19}+4\cdot 2^{20}=2^{22.76}$ CP. The memory cost is negligible. The computational complexity is $2^{22.76}+2^{80-44}\approx 2^{36}$ 40-round KATAN32 computations, which is dominated by the exhaustive search for the remaining 36 key bits.

Table 6 shows cubes and maxterms for 50-round KATAN32. Out of the 46 maxterms obtained in total, we observed that the maxterm $\texttt{1B8EE77B}_\texttt{x}$ gives equations k_2+k_{12} and $k_8+k_{26}+1$ and a similar phenomenon happened for the maxterm $\texttt{EB3AEAE6}_\texttt{x}$ and $\texttt{9CF75766}_\texttt{x}$. Thus, the data complexity becomes $43\cdot 2^{20}=2^{25.42}$ and the time complexity is $2^{25.42}+2^{80-46}\approx 2^{34}$ 50-round KATAN32 computations.

Table 7 shows cubes and maxterms for 60-round KATAN32. Out of the 41 maxterms obtained in total, we observed that the maxterm $\texttt{EF2FF9EF}_\texttt{x}$ gives equations $k_{26}+1$ and $k_{22}+k_{32}+1$ and a similar phenomenon happened for the maxterm $\texttt{B7F2DFDF}_\texttt{x}$. Thus, the data complexity becomes $39\cdot 2^{25}\approx 2^{30.28}$ CP and time complexity is $2^{30.28}+2^{80-41}=2^{39}$ 60-round KATAN32 encryptions.

In all our cube attacks, we ran $10,000$ linearity tests and then we tested the equations for 50 distinct random keys to be sure they are correct.

4.2 Cube Attack on KATAN48

Table 9 shows cubes and maxterms for 30-round KATAN48. Out of 33 maxterms, six of the have degree 14, nine have degree 13 and eighteen have degree 12. The data complexity is $6\cdot 2^{14}+9\cdot 2^{13}+18\cdot 2^{12}=2^{17.90}$ CP. The memory cost is negligible. Since two subkey bits are used per round, at most 60 key bits are used across 30 rounds. The computational complexity is $2^{17.90}+2^{60-33}\approx 2^{27}$ 30-round KATAN48 computations, which is dominated by the exhaustive search for the remaining 27 key bits.

Table 10 shows cubes and maxterms for 40-round KATAN48. All 31 obtained maxterms have degree 20. The data complexity is $31\cdot 2^{20}\approx= 2^{24.95}$ CP. The memory cost is negligible. The computational complexity is $2^{24.95}+2^{49}\approx 2^{49}$ 40-round KATAN48 computations, which is dominated by the exhaustive search for the remaining 49 key bits.

4.3 Cube Attack on KATAN64

Table 11 shows cubes and maxterms for 30-round KATAN64. All 25 maxterms found have degree 16. The data complexity of the attack is $25\cdot 2^{16}\approx 2^{20.64}$ CP. The memory cost is negligible. Since only two subkey bits are used per round, there are at most 60 key bits involved in 30 rounds. The time complexity is $2^{20.64} + 2^{60-25} \approx 2^{35}$ 30-round KATAN64 computations.

5 Combining Cube and Algebraic Attacks

The bottleneck in cube attacks is that after some rounds, the degree of maxterms becomes large. Therefore, it takes a long time to find a linear superpolynomial. But still, if we even get a few linear superpolynomials, it would help to reduce the complexity of the classical algebraic attack. In fact, the overall complexity would be the sum of those two complexities. For a small number of rounds, algebraic attacks are successful, but for larger number of rounds it becomes slower. In such cases, the result of cube attacks and classical algebraic attacks can be combined. For instance, observing Table 4, we have obtained a 3-bit condition on the key bits for 71-round KATAN32 using cube attacks with time complexity $2^{29.58}$. The complexity of algebraic attack alone is $2^{66.60}$. Binding these two attacks reduces the complexity of algebraic attack by $1/8$ because it reduces the number of keys to be guessed from 35 to 32. In fact, we need to guess 3 bits less in order to get the same complexity. So, in Table 4, this reduces our complexity to $2^{63.60}$.

6 Side-Channel Attack for Full-Round KATAN32

In this section we consider side-channel attack models such as [17] in which internal cipher data leaks after r rounds, where $r < 254$, of some full-round KATAN cipher. On one hand, such data is supposed to have been independently captured by some side channels for instace, power or timing analysis or electromagnetic emanations (which is a strong assumption). On the other hand, for our attack setting, only one bit of the cipher state is needed.

The position of the internal cipher data that leaks is selected by the adversary such that its polynomial representation has low degree d and it can be regarded as ciphertext bit c_j after r rounds. Unlike [17], though, we consider c_j to be error free, that is, noise-free. Cube attacks are further employed to derive information on the key from c_j. In this setting, the same bit c_j is supposed to be accessible after each encryption of 2^d CP by the adversary. The adversary chooses different cubes in order to obtain new equations from c_j, all of which are mutually linearly independent.

In this model, only very few internal cipher bits are allowed to leak. In our case, only a single internal bit will be used. Assume one can get the value of internal bit c_{19} after 40 rounds (c.f. Table 3). We can recover 29 key bits via cube attack with data complexity $10 \cdot 2^{12} + 2^{15} + 2 \cdot 2^{16} + 2^{18} + 3 \cdot 2^{19} + 12 \cdot 2^{20} = 2^{23.80}$ CP. The remaining key bits are recovered by brute force. This brings about the time complexity of 2^{51} encryptions to attack the full 254-round KATAN32.

7 Conclusions

This paper described algebraic, AIDA/cube and side-channel attacks on the KATAN family of block ciphers [13]. A new feature observed in cube attacks is that the same maxterms suggests more than one linear independent equation on the key bits. This phenomenon leads to a reduction in the data complexity of our attacks.

For algebraic attacks, deploying pre-processing step on the system of equations before feeding it to the SAT solvers decreases the complexity of the attack for KATAN32. As topic for further research, this method can be tried on other family members.

In the side-channel attack for KATAN32, we observed significant leakage from bit 19 after 40 rounds. More specifically, we could recover 29 linear independent equations on the key bits. Surprisingly enough, this bit position is exactly the LSB of register L_1. This finding is similar to the structure of chosen plaintexts picked in attacking various versions using SAT solvers (Sect. 3.1). We leave similar side-channel analysis of KATAN48 and KATAN64 as future work.

Table 4 summarizes the attack complexities on the KATAN family of block ciphers. In this table, we keep two different time complexities: Time_1 and Time_2, since there is no straightforward and unique way to convert one into the other. Recall that Time_1 measures the effort in number of encryptions, while Time_2 measures the effort in clock time. The former is used for attacks that explicitly perform partial encryption or decryption, while the latter is used for attacks related to internal operations in SAT solvers.

References

1. G. Ars and J-C. Faugère. An Algebraic Cryptanalysis of Nonlinear Filter Generators using Gröbner Bases. Technical report, INRIA research report, https://hal.ccsd.cnrs.fr/, 2003.
2. J.P. Aumasson, I. Dinur, W. Meier, and A. Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In *FSE*, volume 5665, pages 1–22, 2009.
3. G. Bard. *Algebraic Cryptanalysis*. Springer, 2009.
4. G. Bard, N. Courtois, and C. Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. In *presented at ECRYPT workshop Tools for Cryptanalysis*, eprint/2007/024, 2007.
5. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *ACM STOC*, pages 73–83, 1990.
6. B. Buchberger. *An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal*. PhD thesis, Johannes Kepler University of Linz (JKU), 1965.
7. N. Courtois, G. Bard, and D. Wagner. Algebraic and Slide Attacks on Keeloq. In *FSE*, volume 5086, pages 97–115, 2009.
8. N. Courtois and G.V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In *IMA Int. Conf.*, volume 4887, pages 152–169. Springer, 2007.
9. N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *EUROCRYPT*, volume 2656, pages 345–359. Springer, 2003.
10. N. Courtois, S. O'Neil, and J. Quisquater. Practical Algebraic Attacks on the Hitag2 Stream Cipher. In *ISC*, volume 5735, pages 167–176. Springer, 2009.
11. N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *Asiacrypt*, volume 2501, pages 267–287. Springer, 2002.

12. N. Courtois, A. Shamir, J. Patarin, and A. Klimov. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology, Eurocrypt*, volume 1807, pages 392–407. Springer, 2000.
13. C. De Canniére, O. Dunkelman, and M. Knezević. Katan and ktantan - a family of small and efficient hardware-oriented block ciphers. In *CHES*, volume 5747, pages 272–288, 2009.
14. C. De Canniére and B. Preneel. *Trivium. in New Stream Cipher Designs*, volume 4986. Springer, 2008.
15. J. Ding, J. Buchmann, Mohamed M.S.E, N. Mohamed, W.S.A, and R-P. Weinmann. MutantXL algorithm. In *Proceedings of the 1st International Conference in Symbolic Computation and Cryptography*, pages 16–22, 2008.
16. I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. In *EUROCRYPT*, volume 5479, pages 278–299. Springer, 2009.
17. I. Dinur and A. Shamir. Side Channel Cube Attacks on Block Ciphers. IACR ePrint Archive, ePrint 127, 2009.
18. N. Een and N. Sorensson. Minisat - A SAT Solver with Conflict-Clause Minimization. In *Theory and Applications of Satisfiability Testing*, 2005.
19. J. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Symbolic and Algebraic Computation - ISSAC*, pages 75–83, 2002.
20. J.C. Faugère. A new effcient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999.
21. S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on Keeloq. In *EUROCRYPT*, volume 4965, pages 1–18, 2008.
22. D. Lazard. Gröbner-bases, Gaussian elimination and resolution of systems of algebraic equations. In *European Computer Algebra Conference on Computer Algebra*, volume 162. Springer, 1983.
23. F.S. Macaulay. *The algebraic theory of modular systems*. Cambridge Mathematical Library, 1916.
24. H.M. Markovitz. The Elimination Form of the Inverse and Its Application to Linear Programming. *Management Science*, pages 225–269, 1957.
25. M. S. E. Mohamed, W.S.A.E Mohamed, J. Ding, and J. Buchmann. MXL2: Solving Polynomial Equations over GF(2) using an Improved Mutant Strategy. In *Proceedings of The Second international Workshop on Post-Quantum Cryptography*, volume 5299, pages 203–215, 2008.
26. K. Nohl and M. Soos. Solving Low-Complexity Ciphers with Optimized SAT Solvers. In *EUROCRYPT*, 2009.
27. H. Raddum and I. Semaev. New technique for solving sparse equation systems. In *Cryptology ePrint Archive*, http://eprint.iacr.org/2006/475, 2006.
28. SAT. Sat Race Competition. http://www.satcompetition.org/.
29. C.E. Shannon. *Claude Elwood Shannon Collected Papers*. Piscataway: Wiley-IEEE Press, 1993.
30. M. Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. In *Cryptology ePrint Archive, report 413*, 2007.
31. K.K.H Wong and G. Bard. Improved Algebraic Cryptanalysis of QUAD, Bivium and Trivium via Graph Partitioning on Equation Systems. In *ACISP*, 2010.

A Maxterms and Cube indexes

The maxterms listed bellow represent the ones with the smallest maxterms. All cube equations listed are linearly independent (LI). We use hex format to describe maxterms such that those one bits are selected bits. e.g. $0000000f_x$ means the maxterm cube indices are 0, 1, 2 and 3. For KATAN32, plaintext/ciphertext bits are numbered as $p = (p_{31}, \ldots, p_0)$. For KATAN48, the bit numbering is $p = (p_{47}, \ldots, p_0)$. For KATAN64, the bit numbering is $p = (p_{63}, \ldots, p_0)$.

Table 2. Running time and some statistical results for different number of rounds of the preprocessed equations for KATAN32. The running times are in second.

# of rounds	76	77	77	78	79	80	76	76
fixed	45	45	45	45	45	45	43	41
		first batch	second batch					
1	2.89	1.00	2.43	11.04	17.05	59.62	1.50	1.75
2	3.15	2.16	3.69	11.54	24.97	64.61	5.48	1.91
3	3.39	2.25	4.01	14.51	26.86	100.28	15.75	3.36
4	3.39	3.39	4.12	15.83	28.82	135.34	25.88	3.77
5	4.61	3.93	4.40	19.17	54.27	157.10	34.81	5.17
6	6.73	4.16	4.44	24.99	57.02	166.41	39.92	5.65
7	8.29	4.22	4.65	51.46	60.72	230.60	39.97	8.64
8	8.46	4.58	4.72	63.04	64.08	277.04	45.06	11.35
9	11.54	4.81	5.07	86.06	70.34	353.45	50.19	21.71
10	13.15	4.84	6.41	89.89	89.17	354.07	50.79	35.31
11	17.19	4.96	6.81	109.21	109.86	402.56	52.09	41.71
12	17.62	5.44	10.08	115.86	130.28	423.76	60.94	53.7
13	23.64	5.62	14.54	141.19	137.77	433.73	75.35	55.77
14	26.60	5.74	15.03	148.91	145.05	463.78	102.91	61.6
15	27.69	5.83	18.16	161.49	210.29	516.65	116.01	78.29
16	37.32	6.80	18.51	163.23	217.28	687.88	121.89	84.18
17	38.04	7.64	19.51	206.66	269.08	1163.48	123.25	87.51
18	39.67	8.38	21.31	218.43	326.69	1591.56	123.36	104.76
19	48.68	9.54	21.35	230.86	402.61	2180.93	124.39	108.29
20	50.63	10.08	21.57	236.17	408.39	3261.20	131.54	128.62
21	56.51	11.32	22.06	241.45	537.16	3274.25	132.67	138.37
22	62.53	13.81	22.41	248.64	547.32	29 timeouts	134.03	166.93
23	66.03	15.72	22.63	256.66	718.58		207.34	170.14
24	81.25	16.69	27.15	293.66	780.44		208.48	182.83
25	88.88	17.47	28.45	319.31	873.25		233.40	183.9
26	101.43	17.86	32.39	377.06	893.29		258.52	185.41
27	115.13	19.19	45.27	455.50	949.06		300.38	200.08
28	127.09	19.63	49.92	504.97	1007.55		326.94	223.6
29	176.33	22.76	54.80	593.65	1223.91		374.62	246
30	200.26	24.29	54.82	822.36	1244.11		387.17	248.05
31	224.75	29.68	73.71	854.80	1388.40		444.42	254.58
32	243.36	30.09	82.72	880.31	1436.00		449.31	256.05
33	258.53	33.27	85.42	1111.59	1632.59		542.73	263.13
34	278.53	34.02	85.56	1118.54	1838.31		829.13	275.75
35	294.99	35.62	97.22	1197.05	1864.98		905.35	304.75
36	353.49	35.94	97.76	1388.38	1875.87		954.94	305.1
37	407.02	43.33	103.34	1449.29	2031.08		1217.79	305.18
38	423.38	43.65	111.18	1514.89	2038.93		1367.94	328.86
39	475.98	48.18	118.48	1517.73	2167.55		1390.52	352.89
40	506.67	48.22	119.15	1533.10	2262.50		1618.79	356.23
41	687.95	49.96	184.91	1538.97	2369.57		2234.32	403.7
42	842.95	73.62	222.26	1689.96	2413.38		2455.77	407.63
43	942.88	106.69	226.48	1894.40	2495.42		2668.97	418.7
44	2387.95	133.21	335.07	2031.93	2641.90		3246.26	427.04
45	2400.12	186.39	456.45	2375.14	2960.11		3326.73	429.21
46	3722.62	201.89	662.92	2682.71	3460.90		3530.63	555.35
47	4471.28	302.66	815.38	2837.97	4023.81		7157.16	577.3
48	> 6000	344.63	976.94	3731.61	4129.64		9378.05	6248.59
49	> 6000	433.70	2378.61	> 6000	4212.65		$> 10,000$	6763.91
50	> 6000	524.56	> 6000	> 6000	> 6000		$> 10,000$	9655.8
Threshold-time	2611.34	2645.70	2645.70	2680.06	2714.42	2748.78	10445.35	41781.40
# faster	47	50	49	48	49	21	48	50
Median	95.16	17.67	30.42	348.19	883.27	n/a	245.96	184.66
Mean of all but 6	463.21	38.98	100.88	768.47	1146.77	n/a	868.70	205.97
Stdev of all but 6	957.09	60.29	168.93	786.56	1054.09	n/a	1381.91	154.29
Kurtosis of all but 6	9.51	9.19	9.30	0.17	-0.12	n/a	9.33	-0.46
Times faster than brute force	5.64	67.87	26.23	3.49	2.37	n/a	12.02	202.85
Mean of log	4.648	2.914	3.650	5.938	6.369	n/a	5.706	4.810
Stdev of log	1.820	1.185	1.421	1.380	1.396	n/a	1.513	1.319
Kurtosis of log	-0.582	-0.470	-0.620	-0.514	-0.985	n/a	-0.985	0.867

Table 3. Maxterms, 29 cube equations from ciphertext bit c_{19} from 40-round KATAN32.

Maxterm	Degree	Cube equation	Cipher bit
$\texttt{41356548}_X$	12	k_4	c_{19}
$\texttt{2464E14C}_X$	12	k_{15}	c_{19}
$\texttt{1EA26848}_X$	12	k_5+1	c_{19}
$\texttt{E3516900}_X$	12	k_1+k_{16}	c_{19}
$\texttt{4A8E6888}_X$	12	$k_0+k_{17}+1$	c_{19}
$\texttt{EBD02900}_X$	12	$k_3+k_{10}+1$	c_{19}
$\texttt{A0867A0C}_X$	12	$k_{14}+k_{17}+1$	c_{19}
$\texttt{C0C34C43}_X$	12	$k_4+k_{10}+k_{19}$	c_{19}
$\texttt{E2A54302}_X$	12	$k_{11}+k_{15}+k_{23}$	c_{19}
$\texttt{9C045983}_X$	12	$k_2+k_7+k_{11}+k_{16}+k_{24}+k_{26}$	c_{19}
$\texttt{bd30cb11}_X$	15	k_{13}	c_{19}
$\texttt{7c366259}_X$	16	k_{18}	c_{19}
$\texttt{2cd5f264}_X$	16	$k_6+k_{15}+1$	c_{19}
$\texttt{b7351759}_X$	18	$k_3+k_{18}+k_{23}$	c_{19}
$\texttt{cf9df815}_X$	19	k_3+1	c_{19}
$\texttt{75e471ee}_X$	19	$k_{24}+1$	c_{19}
$\texttt{65765d7a}_X$	19	$k_0+k_{10}+k_{16}+k_{18}+k_{19}+k_{26}+k_{30}+k_{43}$	c_{19}
$\texttt{ab7f3a4b}_X$	20	k_7	c_{19}
$\texttt{b61d73f9}_X$	20	k_8+1	c_{19}
$\texttt{3d7f3476}_X$	20	k_2+k_{19}	c_{19}
$\texttt{e4f636be}_X$	20	k_6+k_{16}	c_{19}
$\texttt{acd1bbf6}_X$	20	$k_{12}+k_{20}+k_{29}$	c_{19}
$\texttt{bdcddcac}_X$	20	$k_{16}+k_{21}+k_{26}+1$	c_{19}
$\texttt{deff1456}_X$	20	$k_7+k_9+k_{18}+k_{26}$	c_{19}
$\texttt{37d7d2b3}_X$	20	$k_{16}+k_{23}+k_{26}+k_{43}$	c_{19}
$\texttt{d7035eef}_X$	20	$k_4+k_8+k_{14}+k_{18}+1$	c_{19}
$\texttt{ad754de7}_X$	20	$k_2+k_{16}+k_{19}+k_{20}+k_{26}+k_{43}$	c_{19}
$\texttt{17dfaa6d}_X$	20	$k_{13}+k_{18}+k_{21}+k_{22}+k_{23}+k_{26}+k_{30}+1$	c_{19}
$\texttt{6afeaf85}_X$	20	$k_0+k_9+k_{18}+k_{24}+k_{25}+k_{26}+k_{27}+k_{30}$	c_{19}

Table 4. Attack complexities on KATAN family of block ciphers (memory complexity is negligible).

Cipher	# Rounds	Time$_1$	Time$_2$	Data	Fixed	Attack	Source
KATAN32	40		11 sec	3 KP	0	MiniSat2.2, LI converter	Sect. 3.1
	40	2^{36}		$2^{22.76}$ CP	0	AIDA/Cube	Sect. 4.1
	50	2^{34}		$2^{25.42}$ CP	0	AIDA/Cube	Sect. 4.1
	50		11 sec	3 KP	0	MiniSat2.2, LI converter	Sect. 3.1
	60	2^{39}		$2^{30.28}$ CP	0	AIDA/Cube	Sect. 4.1
	60		18 sec	3 KP	0	MiniSat2.2, LI converter	Sect. 3.1
	65		1.81 min	3 KP	0	MiniSat2.2, LI converter	Sect. 3.1
	66		8.85 min	3 KP	0	MiniSat2.2, LI converter	Sect. 3.1
	67		26 sec	3 KP	30	MiniSat2.2, LI converter	Sect. 3.1
	68		2.55 min	3 KP	30	MiniSat2.2, LI converter	Sect. 3.1
	69		47.76 min	3 KP	35	MiniSat2.2, LI converter	Sect. 3.1
	70		1.64 min	10 CP	35	MiniSat2.2, LI converter	Sect. 3.1
	71		3.58 min	10 CP	35	MiniSat2.2, LI converter	Sect. 3.1
	71		3.58 min	10 CP	35	MiniSat2.2 & Cube, LI converter	Sect. 5
	75		12.50 h	3 KP	35	MiniSat2.2, LI converter	Sect. 3.1
	76		1.59 min	20 CP	45	MiniSat2.2, BCJ converter/Pre-Proc	Sect. 3.2
	76		4.1 min	20 CP	43	MiniSat2.2, BCJ converter/Pre-Proc	Sect. 3.2
	76		3.08 min	20 CP	41	MiniSat2.2, BCJ converter/Pre-Proc	Sect. 3.2
	77		18 sec	20 CP	45	MiniSat2.2, BCJ converter/Pre-Proc	Sect. 3.2
	78		5.80 min	20 CP	45	MiniSat2.2, BCJ converter/Pre-Proc	Sect. 3.2
	79		14.72 min	20 CP	45	MiniSat2.2, BCJ converter/Pre-Proc	Sect. 3.2
	254	2^{51}		$2^{23.80}$ CP	0	Side-Channel	Sect. 6
KATAN48	30	2^{27}		$2^{17.90}$ CP	0	AIDA/Cube	Sect. 4.2
	40	2^{49}		$2^{24.95}$ CP	0	AIDA/Cube	Sect. 4.2
	40		2 sec	5 CP	40	MiniSat2.2, LI converter	Sect. 3.1
	50		7 sec	5 CP	40	MiniSat2.2, LI converter	Sect. 3.1
	60		13.18 min	5 CP	40	MiniSat2.2, LI converter	Sect. 3.1
	61		7.12 min	5 CP	45	MiniSat2.2, LI converter	Sect. 3.1
	62		11.86 min	10 CP	40	MiniSat2.2, LI converter	Sect. 3.1
	63		17.47 min	10 CP	45	MiniSat2.2, LI converter	Sect. 3.1
	64		6.42 h	5 CP	40	MiniSat2.2, LI converter	Sect. 3.1
KATAN64	30	2^{35}		$2^{20.64}$ CP	0	AIDA/Cube	Sect. 4.3
	40		2 sec	5 CP	40	MiniSat2.2, LI converter	Sect. 3.1
	50		12 sec	5 CP	40	MiniSat2.2, LI converter	Sect. 3.1
	60		3.17 h	5 CP	40	MiniSat2.2, LI converter	Sect. 3.1

Time$_1$: time complexity unit for attacking r rounds is number of r-round KATAN computations.
Time$_2$: clock time for algebraic attacks; KP: known plaintext; CP: chosen plaintext;
LI converter: local interpolation converter; BCJ: Bard-Courtois-Jefferson converter
negl: negligible, Pre-Proc: preprocessed system of equations

Table 5. Maxterms, cube degree and equations and ciphertext bit for 40-round KATAN32.

Maxterm	Degree	Cube equation	Cipher bit
03D193AF$_{\mathtt{X}}$	16	k_6	c_{23}
8FF802F4$_{\mathtt{X}}$	16	k_1+1	c_7
315D8EE1$_{\mathtt{X}}$	16	k_2+k_{13}	c_{21}
531DAE2A$_{\mathtt{X}}$	16	$k_9+k_{20}+k_{24}+1$	c_{22}
AD3E0887$_{\mathtt{X}}$	15	k_5	c_{22}
5C3449FA$_{\mathtt{X}}$	16	$k_3+k_7+k_{13}+k_{28}$	c_2
C934392F$_{\mathtt{X}}$	16	$k_5+k_7+k_{10}+1$	c_0
E05946EC$_{\mathtt{X}}$	15	k_{12}	c_5
C2F7904D$_{\mathtt{X}}$	16	$k_{14}+k_{17}$	c_{22}
AAC90EE6$_{\mathtt{X}}$	16	$k_{19}+1$	c_3
C5A473A5$_{\mathtt{X}}$	16	$k_5+k_7+k_{18}+k_{19}$	c_2
24AC9FE1$_{\mathtt{X}}$	16	k_0+1	c_5
F0279C78$_{\mathtt{X}}$	16	$k_{16}+1$	c_0
BD30CB11$_{\mathtt{X}}$	15	$k_{13}+1$	c_{19}
66079CAB$_{\mathtt{X}}$	16	k_0+k_{10}	c_2
EBCB9421$_{\mathtt{X}}$	16	k_9	c_3
41D3D98E$_{\mathtt{X}}$	16	$k_{12}+k_{25}+k_{36}$	c_{21}
41D3D98E$_{\mathtt{X}}$	16	$k_{24}+1$	c_{22}
49A5E4AB$_{\mathtt{X}}$	16	$k_6+k_{12}+k_{16}+k_{29}+1$	c_{21}
F92F9920$_{\mathtt{X}}$	16	$k_{11}+k_{22}$	c_0
DD59A48C$_{\mathtt{X}}$	16	$k_4+k_{12}+k_{14}+k_{15}$	c_2
6DED8883$_{\mathtt{X}}$	16	k_{14}	c_4
7D856271$_{\mathtt{X}}$	16	$k_3+k_{14}+1$	c_{22}
FB4433C1$_{\mathtt{X}}$	16	$k_{15}+1$	c_{21}
8A879E95$_{\mathtt{X}}$	16	$k_{11}+1$	c_0
916A7599$_{\mathtt{X}}$	16	$k_2+k_8+k_{10}+k_{18}+k_{19}+k_{24}$	c_4
8BCD8CCC$_{\mathtt{X}}$	16	$k_9+k_{21}+1$	c_2
F5495155$_{\mathtt{X}}$	16	$k_8+k_{11}+k_{12}+k_{15}+k_{18}+k_{20}+k_{24}+k_{26}+1$	c_1
1B584CCF$_{\mathtt{X}}$	16	$k_3+k_5+k_{23}+1$	c_4
89A3C57C$_{\mathtt{X}}$	16	$k_{10}+k_{12}+k_{20}+k_{37}$	c_{22}
69315AA7$_{\mathtt{X}}$	16	$k_6+k_{10}+k_{25}+k_{28}$	c_{23}
3A4D88E7$_{\mathtt{X}}$	16	$k_0+k_9+k_{14}+k_{17}+k_{27}+k_{37}+1$	c_{22}
9271A58F$_{\mathtt{X}}$	16	$k_2+k_8+k_9+k_{19}+k_{23}+k_{27}+k_{34}$	c_4
7911E746$_{\mathtt{X}}$	16	$k_1+k_8+k_{26}+k_{30}+1$	c_1
4EC6856B$_{\mathtt{X}}$	16	$k_0+k_{10}+k_{13}+k_{18}+k_{35}+1$	c_{23}
E1D99370$_{\mathtt{X}}$	16	$k_{15}+k_{17}+k_{32}$	c_{20}
846AD5F2$_{\mathtt{X}}$	16	$k_6+k_{14}+k_{16}+k_{33}$	c_{24}
82DC78B3$_{\mathtt{X}}$	16	$k_{21}+k_{31}+1$	c_{20}
E4416E9E$_{\mathtt{X}}$	16	$k_1+k_6+k_{14}+k_{16}+k_{22}+k_{23}+k_{39}+1$	c_{21}
65765D7A$_{\mathtt{X}}$	19	$k_0+k_{10}+k_{16}+k_{18}+k_{19}+k_{26}+k_{30}+k_{43}$	c_{19}
EC3F1DF2$_{\mathtt{X}}$	20	$k_6+k_{16}+k_{18}+k_{25}+k_{27}+k_{42}+1$	c_0
AA6C3FDD$_{\mathtt{X}}$	20	$k_7+k_{22}+k_{24}+k_{32}+k_{41}$	c_{20}
CB6F2FD2$_{\mathtt{X}}$	20	$k_{14}+k_{16}+k_{24}+k_{26}+k_{33}+k_{40}+1$	c_1
7B36D5B5$_{\mathtt{X}}$	20	$k_8+k_{23}+k_{28}+k_{31}+k_{38}$	c_2

Table 6. Maxterms, LI cube equations and ciphertext bit for 50-round KATAN32.

Maxterm	Degree	Cube equation	Cipher bit
1B8EE77B$_X$	20	k_2+k_{12}	c_{10}
1B8EE77B$_X$	20	$k_8+k_{26}+1$	c_{31}
B1FF633A$_X$	20	$k_{16}+1$	c_{10}
EB3AEAE6$_X$	20	k_{22}	c_9
EB3AEAE6$_X$	20	$k_{16}+k_{20}+k_{28}+k_{31}+k_{35}+k_{42}$	c_{28}
AEF689F9$_X$	20	$k_{16}+k_{32}+1$	c_{30}
56CE3DFA$_X$	20	k_{12}	c_{10}
7FDAA996$_X$	20	k_5	c_{10}
D77FE20E$_X$	20	k_{25}	c_{28}
AF19DFB4$_X$	20	k_6	c_{12}
DC3C97ED$_X$	20	$k_4+k_{11}+1$	c_{30}
61BC7B9F$_X$	20	k_5+k_{20}	c_9
23B35FD7$_X$	20	k_0	c_{12}
9CF75766$_X$	20	$k_{15}+1$	c_6
9CF75766$_X$	20	k_7+1	c_{26}
E58FB7CA$_X$	20	$k_{22}+k_{26}$	c_{26}
C7E6C7CB$_X$	20	$k_{11}+k_{21}+k_{31}+1$	c_{10}
3E3BE3EA$_X$	20	$k_{13}+1$	c_{10}
3FFCCD62$_X$	20	$k_8+k_{10}+k_{13}+k_{14}+k_{20}+k_{24}$	c_{30}
EF4FD985$_X$	20	$k_6+k_{11}+k_{22}+k_{32}+k_{33}+k_{37}$	c_7
2F6D66FA$_X$	20	k_3+1	c_{31}
BE5E19F3$_X$	20	$k_2+k_7+k_9+k_{12}+k_{16}+k_{17}+k_{21}+k_{26}+k_{27}+k_{30}+k_{34}+k_{39}+k_{43}$	c_{29}
ECDD58BD$_X$	20	$k_6+k_7+k_{16}+k_{17}+k_{20}+k_{23}+k_{25}+k_{26}+k_{27}+k_{34}+1$	c_{29}
DEBCFB22$_X$	20	$k_4+k_{17}+k_{28}+k_{35}+k_{37}+k_{45}$	c_8
FE3E09D7$_X$	20	$k_{11}+k_{22}$	c_9
F83B3AEB$_X$	20	k_{18}	c_{29}
BACCAF37$_X$	20	$k_2+k_{12}+k_{14}+k_{22}$	c_{12}
7FD07B66$_X$	20	$k_{15}+k_{31}+1$	c_8
BAFEA8D3$_X$	20	k_{27}	c_{10}
AF6AAE75$_X$	20	k_1+1	c_9
3ADC3DD7$_X$	20	$k_0+k_2+k_{12}+k_{20}+k_{24}+k_{31}$	c_{10}
A7D3F749$_X$	20	$k_2+k_{10}+k_{12}+k_{21}+k_{23}+k_{30}$	c_8
8FF7D615$_X$	20	$k_{16}+k_{18}+k_{28}+1$	c_{28}
AF88BDFA$_X$	20	k_{17}	c_9
FE1A11FF$_X$	20	$k_5+k_{14}+k_{19}+k_{22}+k_{31}+1$	c_{12}
DFF9C30D$_X$	20	k_{23}	c_{29}
95BF5D4D$_X$	20	$k_2+k_4+k_7+k_8+k_{10}+k_{14}+k_{16}+k_{18}+k_{22}+k_{24}+k_{26}+k_{35}+1$	c_6
9DF2EE93$_X$	20	$k_{21}+k_{38}+1$	c_{27}
6DD3973B$_X$	20	$k_{12}+k_{14}+k_{29}$	c_{11}
A271A7FF$_X$	20	$k_1+k_9+k_{10}+k_{16}+k_{20}+k_{21}+1$	c_{10}
F6CDFA15$_X$	20	$k_8+k_{14}+k_{15}+k_{18}+k_{24}+k_{29}+k_{33}+k_{40}$	c_{29}
5E5AB5EB$_X$	20	$k_{12}+k_{14}+k_{16}+k_{20}+k_{22}+k_{25}+k_{26}+k_{39}$	c_{31}
FD847AF6$_X$	20	$k_{24}+k_{33}+1$	c_9
F770ECEC$_X$	20	$k_5+k_{16}+k_{24}+k_{26}+k_{27}+k_{30}+k_{32}+k_{34}+k_{35}+k_{36}+k_{43}$	c_5
FBAE4E3A$_X$	20	$k_7+k_{28}+k_{29}+k_{33}+k_{44}+1$	c_{27}
EEB6A9A7$_X$	20	$k_6+k_9+k_{16}+k_{18}+k_{19}+k_{25}+k_{28}+k_{37}+k_{38}+k_{41}+k_{43}+k_{48}$	c_7

Table 7. Maxterms, cube degree and equations and ciphertext bit for 60-round KATAN32.

Maxterm	Degree	Cube equation	Cipher bit
B6F7FAFD$_X$	25	$k_{30}+1$	c_{31}
EFE7F6FA$_X$	25	k_{38}	c_{11}
F7CDFFCE$_X$	25	$k_{28}+k_{30}+k_{32}+k_{36}+k_{40}+k_{42}+k_{49}+k_{50}+1$	c_{31}
FEBF7EAE$_X$	25	$k_{28}+k_{30}+k_{40}+k_{54}+1$	c_{14}
63D7FFF7$_X$	25	$k_{16}+k_{26}+k_{30}+k_{38}+k_{40}+k_{43}+k_{44}$	c_{27}
3F7BBF5F$_X$	25	$k_8+k_{18}+k_{20}+k_{26}+k_{32}$	c_{17}
EF2FF9EF$_X$	25	$k_{26}+1$	c_{12}
EF2FF9EF$_X$	25	$k_{22}+k_{32}+1$	c_{31}
FFF3F573$_X$	25	$k_{15}+k_{17}+k_{19}+k_{23}+k_{32}+k_{33}+k_{35}+k_{37}+k_{41}+k_{43}+k_{44}+1$	c_{11}
DA9EFFF7$_X$	25	$k_4+k_{10}+k_{14}+k_{22}+k_{24}+k_{29}+k_{30}+k_{49}+1$	c_{11}
FFD6BABF$_X$	25	k_{46}	c_{31}
FF5777F6$_X$	25	k_{47}	c_{11}
B7F2DFDF$_X$	25	$k_{16}+1$	c_{16}
B7F2DFDF$_X$	25	$k_{13}+k_{16}+1$	c_{17}
BDF7FD97$_X$	25	$k_{27}+k_{34}$	c_{11}
7FDEBFDC$_X$	25	$k_{14}+k_{21}+k_{22}+k_{41}$	c_{15}
EFF9B7ED$_X$	25	$k_{13}+k_{31}+1$	c_{16}
FFF5FDCC$_X$	25	$k_{19}+k_{23}+k_{27}+k_{35}+k_{38}+k_{43}+k_{48}+1$	c_9
EEBB7DF7$_X$	25	$k_{22}+1$	c_{16}
F7C6EDFF$_X$	25	$k_8+k_{18}+k_{35}+k_{43}+1$	c_{14}
FE3BF77E$_X$	25	$k_{17}+k_{33}+1$	c_{15}
9FFE7FAE$_X$	25	$k_{10}+k_{18}+k_{20}+k_{24}+1$	c_{17}
EFFFDD9A$_X$	25	$k_{38}+k_{39}$	c_{12}
FEF7779B$_X$	25	$k_{23}+k_{27}$	c_{14}
CFFF7BE6$_X$	25	k_{12}	c_{14}
EABFF73F$_X$	25	$k_{32}+k_{36}+1$	c_{13}
BC7FCF7F$_X$	25	k_2+1	c_{14}
FADFECFB$_X$	25	$k_{43}+1$	c_{14}
DDD3FF3F$_X$	25	$k_{28}+1$	c_{13}
EB67DDFF$_X$	25	$k_{16}+k_{26}+k_{28}+k_{35}+k_{40}+k_{44}+1$	c_{30}
FEEFB8FE$_X$	25	$k_{30}+k_{32}+k_{42}$	c_{31}
3CFFEF7E$_X$	25	$k_{14}+k_{16}+k_{20}+k_{22}+1$	c_{13}
DFEFF4DD$_X$	25	k_{15}	c_{31}
AFBEFDCF$_X$	25	k_{10}	c_{17}
DAF9FFED$_X$	25	$k_{28}+k_{32}+k_{36}+k_{45}+1$	c_{18}
DF733FEF$_X$	25	$k_{16}+k_{21}+1$	c_{17}
BF7BEE6F$_X$	25	k_1+1	c_{13}
FFEE57FC$_X$	25	$k_{11}+k_{29}+k_{33}+k_{35}+k_{37}+k_{39}+k_{41}+k_{44}+k_{45}+k_{49}+k_{50}+k_{51}+k_{59}+k_{60}+1$	c_{11}
FA7ECFFD$_X$	25	$k_{17}+k_{22}+k_{24}+k_{27}+k_{28}+k_{33}+k_{34}+k_{39}$	c_{16}
B9E77FFE$_X$	25	$k_{10}+k_{20}+k_{24}+k_{26}+k_{28}+k_{30}+k_{36}+k_{37}+k_{38}+k_{44}+k_{53}+1$	c_{11}
FF37EDEB$_X$	25	k_{50}	c_{29}

Table 8. Maxterms, cube degree and equations and ciphertext bit for 71-round KATAN32.

Maxterm	Degree	Cube equation	Cipher bit
FFFFD5FC$_X$	27	$k_{52}+k_{54}+k_{60}$	c_{14}
FFFFF7C5$_X$	27	$k_{31}+k_{51}+k_{53}+k_{54}+k_{64}+k_{71}$	c_{31}
FFF3FA7F$_X$	28	$k_{46}+1$	c_{18}

Table 9. Maxterms, cube degree and equations and ciphertext bit for 30-round KATAN48.

Maxterm	Degree	Cube equation	Cipher bit
080065A4348C$_X$	14	$k_{10}+k_{13}+k_{27}$	c_{30}
816011140125$_X$	12	k_4	c_0
040C10006BA4$_X$	12	k_3	c_{30}
834000025073$_X$	12	k_5+k_{12}	c_0
04800011D1D1$_X$	12	k_2	c_4
4006B04001A9$_X$	12	k_0	c_{35}
0282A2444E00$_X$	12	k_{13}	c_{31}
050E4324024A$_X$	14	$k_0+k_1+k_6+k_{15}+1$	c_0
10A032428412$_X$	12	k_9+1	c_{32}
880391220424$_X$	12	k_2+k_8	c_3
80204850301B$_X$	12	k_6+1	c_{30}
8464022840C4$_X$	12	$k_{11}+1$	c_{30}
094024005E14$_X$	12	$k_1+k_3+k_7+k_{18}$	c_{30}
00C226985028$_X$	13	$k_2+k_4+k_8+k_{10}+k_{25}$	c_3
02160206A070$_X$	12	$k_8+k_{14}+1$	c_{35}
0AAF00027010$_X$	13	$k_3+k_5+k_{10}$	c_{33}
88011180D520$_X$	12	$k_{10}+1$	c_1
430841102A88$_X$	12	k_1	c_0
180028808CD9$_X$	13	$k_0+k_{21}+1$	c_0
40A101501883$_X$	12	$k_2+k_{19}+1$	c_0
0003C11042C3$_X$	12	$k_0+k_{17}+1$	c_2
8C0401084E0E$_X$	13	$k_8+k_{12}+k_{18}$	c_{31}
22A404909D08$_X$	14	$k_0+k_3+k_7+k_{10}+k_{13}+k_{17}+k_{24}$	c_{32}
0F5800240285$_X$	13	$k_5+k_9+k_{16}$	c_3
860051430C08$_X$	12	$k_1+k_7+k_{10}+k_{13}+k_{20}$	c_1
203483004E1A$_X$	14	$k_0+k_3+k_6+k_7+k_{21}+k_{22}+k_{28}$	c_{29}
40E8040512C8$_X$	13	$k_0+k_1+k_2+k_4+k_6+k_8+k_{17}+k_{23}$	c_{33}
64209090B082$_X$	13	$k_3+k_6+k_7+k_9+k_{15}+k_{22}$	c_{30}
013820886215$_X$	13	$k_1+k_6+k_{12}+k_{14}+k_{29}$	c_{36}
A0285800E906$_X$	14	$k_0+k_4+k_5+k_9+k_{10}+k_{11}+k_{13}+k_{26}$	c_0
0C000500DB14$_X$	12	$k_0+k_1+k_2+k_5+k_7+k_8+k_{11}+k_{15}+k_{21}+k_{25}+k_{32}+1$	c_2
048070044F82$_X$	13	$k_0+k_{11}+k_{13}+k_{15}+k_{21}+k_{34}$	c_1
6103080056A9$_X$	14	$k_0+k_1+k_6+k_{10}+k_{11}+k_{16}+k_{17}+k_{23}+k_{30}+1$	c_5

Table 10. Maxterms, cube degree and equations and ciphertext bit for 40-round KATAN48.

Maxterm	Degree	Cube equation	Cipher bit
66140B44FE81$_X$	20	k_3+k_8	c_{37}
2096B841C6F2$_X$	20	k_0+k_6	c_{18}
2004D819B69F$_X$	20	k_8	c_{46}
01E07456499B$_X$	20	$k_3+k_{12}+k_{18}+1$	c_{13}
874108B1E347$_X$	20	k_2+k_9	c_{43}
85DF1310A226$_X$	20	k_{11}	c_{43}
D9F00150D11E$_X$	20	$k_1+k_{12}+1$	c_{12}
204D49C8B56C$_X$	20	k_4	c_{16}
3000F607DC4E$_X$	20	k_6+1	c_{43}
75045046CC5E$_X$	20	k_9	c_{15}
8D705440E2CB$_X$	20	$k_5+k_{14}+k_{18}$	c_{42}
5024603E9A37$_X$	20	$k_3+k_6+k_8+k_{10}+k_{23}+1$	c_{16}
5024603E9A37$_X$	20	k_1+k_2+1	c_{44}
3034E083566D$_X$	20	k_7+1	c_{18}
81E48D04DB19$_X$	20	$k_1+k_3+k_5+k_{14}+k_{15}+1$	c_{41}
41482473ADB4$_X$	20	$k_4+k_{19}+1$	c_{42}
3D4635605382$_X$	20	k_{13}	c_{39}
51902406CABF$_X$	20	$k_3+k_8+k_{10}+1$	c_{44}
583088DB0C6E$_X$	20	$k_1+k_2+k_9+k_{16}$	c_{16}
5040C4CE9AF1$_X$	20	$k_0+k_2+k_4+k_{21}+1$	c_{41}
7749008CBAC1$_X$	20	$k_1+k_5+k_8+k_{10}+k_{11}+k_{13}+1$	c_{42}
96940C46139E$_X$	20	$k_0+k_1+k_9+k_{12}+k_{14}+k_{20}+1$	c_{41}
96800804FF5B$_X$	20	$k_0+k_3+k_8+k_9+k_{15}+k_{17}+1$	c_{46}
211013326F3D$_X$	20	$k_4+k_8+k_{10}+k_{16}+k_{25}$	c_{15}
18574012A577$_X$	20	$k_6+k_8+k_{12}+k_{14}+k_{29}+1$	c_{47}
81668801EE97$_X$	20	$k_0+k_1+k_5+k_6+k_8+k_9+k_{14}+k_{15}+k_{31}+1$	c_{46}
3050A044F5ED$_X$	20	$k_1+k_8+k_9+k_{13}+k_{15}+k_{24}+1$	c_{46}
42BF16A44AA0$_X$	20	$k_7+k_9+k_{22}$	c_{10}
50AD4122AA1F$_X$	20	$k_{10}+k_{27}$	c_{12}
0AAB9004F89D$_X$	20	$k_0+k_3+k_{10}+k_{11}+k_{26}$	c_{20}
C884A100FCF3$_X$	20	$k_3+k_{11}+k_{13}+k_{14}+k_{15}+k_{17}+k_{28}$	c_{18}

Table 11. Maxterms, cube degree and equations and ciphertext bit for 30-round KATAN64.

Maxterm	Degree	Cube equation	Cipher bit
0CB0C29808C10001$_X$	16	k_5	c_{44}
2E2128800020305A$_X$	16	k_4	c_7
10E2002920014471$_X$	16	$k_1 + k_5 + k_{12}$	c_{47}
0A12042100446263$_X$	16	$k_8 + k_{10} + k_{19}$	c_{12}
029290CC02C10140$_X$	16	k_2	c_5
AE0C032002100492$_X$	16	k_9	c_9
4241092108534C00$_X$	16	k_1	c_{44}
0E0864A20828A800$_X$	16	k_0	c_{56}
4104901087403083$_X$	16	k_7	c_8
44010B12812A0124$_X$	16	k_3	c_{49}
0200A0D00305E08A$_X$	16	$k_3 + k_{10}$	c_{48}
041102168238A802$_X$	16	k_6	c_9
439C00A810940044$_X$	16	$k_3 + k_8 + k_{17}$	c_9
60910A0B93000802$_X$	16	$k_1 + k_8$	c_{47}
018C084049C98003$_X$	16	$k_0 + k_1 + k_2 + k_8 + k_{11}$	c_8
3C1500040080C097$_X$	16	$k_4 + k_{15}$	c_{48}
0800FD4900016180$_X$	16	$k_5 + k_9 + k_{18}$	c_{54}
002091443A501C40$_X$	16	$k_2 + k_{13}$	c_{45}
1027118032506001$_X$	16	$k_1 + k_5 + k_{10} + k_{21}$	c_{10}
0080DC00814454A8$_X$	16	$k_5 + k_7 + k_{14}$	c_{49}
11320C0241095220$_X$	16	$k_4 + k_5 + k_7 + k_9 + k_{15} + k_{20} + k_{24}$	c_{50}
8E200808003A8D40$_X$	16	$k_3 + k_6 + k_{12} + k_{16}$	c_{51}
00458C3220521011$_X$	16	$k_0 + k_2 + k_5 + k_{10} + k_{11} + k_{13} + k_{20}$	c_{11}
4024935C01018048$_X$	16	$k_0 + k_5 + k_9 + k_{11} + k_{22}$	c_{49}
8004007882307052$_X$	16	$k_0 + k_6 + k_{12} + k_{23}$	c_6