A New Model for Error-Tolerant Side-Channel Cube Attacks*

Zhenqi Li[1], Bin Zhang[2], Junfeng Fan[3], and Ingrid Verbauwhede[3]

[1] IOS, Chinese Academy of Sciences, Beijing, 100190, China
[2] State Key Laboratory of Information Security, IIE, Chinese Academy of Sciences, Beijing, 100093, China
[3] Katholieke Universiteit Leuven, ESAT SCD/COSIC
{zhangbin,lizhenqi}@is.iscas.ac.cn,
fanjunfeng@gmail.com,
Ingrid.Verbauwhede@esat.kuleuven.be

Abstract. Side-channel cube attacks are a class of leakage attacks on block ciphers in which the attacker is assumed to have access to some leaked information on the internal state of the cipher as well as the plaintext/ciphertext pairs. The known Dinur-Shamir model and its variants require error-free data for at least part of the measurements. In this paper, we consider a new and more realistic model which can deal with the case when *all* the leaked bits are noisy. In this model, the key recovery problem is converted to the problem of decoding a binary linear code over a binary symmetric channel with the crossover probability which is determined by the measurement quality and the cube size. We use the maximum likelihood decoding method to recover the key. As a case study, we demonstrate efficient key recovery attacks on PRESENT. We show that the full 80-bit key can be restored with $2^{10.2}$ measurements with an error probability of 19.4% for each measurement.

Keywords: Side-channel attack, Cube attack, Decoding, PRESENT.

1 Introduction

Cube attacks [8] were formally proposed by Dinur and Shamir at Eurocrypt 2009 as a new branch of algebraic attacks [7]. It is a generic key extraction attack, applicable to any cryptosystem in which at least one single bit can be represented by an unknown low degree multivariate polynomial in the secret and public variables. Several studies [1,2,8,9] have demonstrated that cube attack

* This work was supported by the National Grand Fundamental Research 973 Program of China(Grant No. 2013CB338002), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDA06010701), IIE's Research Project on Cryptography (Grant No. Y3Z0016102) and the programs of the National Natural Science Foundation of China (Grant No. 60833008, 60603018, 61173134, 91118006, 61272476). In addition, This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007) and the KU Leuven Tsinghua fund.

G. Bertoni and J.-S. Coron (Eds.): CHES 2013, LNCS 8086, pp. 453–470, 2013.

is a favorable cryptanalysis approach to many well-designed ciphers. However, mainstream block ciphers tend to resist against cube attacks, since they iteratively apply a highly non-linear round function (based on Sboxes or arithmetic operations) a large number of times and it is unlikely to obtain a low degree polynomial representation for any ciphertext bit.

On the other hand, cube attacks seem to be a promising method for physical attacks, where the attackers can learn some information about the intermediate variables, i.e., state registers. It is likely that the master polynomials of some intermediate variables in the early rounds are of relatively low degree. Since the attack only needs to learn the value of a single wire or register in each execution, it is ideal for probing attacks. The main challenge is overcoming measurement errors. The known Dinur-Shamir model (DS model) treats the uncertain bits as new erasure variables [10,11] and uses more measurements in a larger cube to correct the measurement errors. It is required that the exact knowledge of error positions is known to the adversary and at least part of the measurements are error-free. This is a strong assumption, since in practice each measurement is suspectable to some level of noise.

In this paper, we consider a side-channel cube attack model that can handle errors in each measurement. The data observed by attackers is regarded as the received channel output of some linear code transmitted through a binary symmetric channel (BSC). The crossover probability of the BSC depends on the accuracy of the measurements. Using this model, the problem of recovering the n secret key bits in L linear equations can be considered as the problem of decoding a binary linear $[L, n]$ code with L being the code length and n the dimension. Various decoding techniques can be used to address this problem. In this paper, the maximum likelihood (ML) decoding algorithm is used. We also derive the maximum error probability that each measurement can have in order to successfully retrieve the key.

As a case study, we simulated the proposed model of side-channel cube attack on PRESENT [5]. Since the ML decoding algorithm has a complexity of 2^n, the decoding becomes infeasible for PRESENT ($n = 80$). We solve this problem with a divide-and-conquer strategy. The results are summarized in Table 1.

Table 1. Simulation results on PRESENT under our BSC model

Leakage round	*HW*[a] *leaked bit*	*Data (measurements)*	*Time*[c]	*Key*[b]	*Error tolerance*
2	LSB	$2^{18.9}$	$2^{20.6}$	64	0.6%
2	2nd LSB	$2^{23.1}$	$2^{21.6}$	64	0.4%
1	LSB	$2^{10.2}$	$2^{21.6}$	64	19.4%
1	LSB (partial)	442	$2^{21.6}$	64	9.7%

[a]Hamming weight.
[b]Number of key bits recovered.
[c]Number of key trials.

This paper is structured as follows. We first give a brief review of cube and side-channel cube attacks in Section 2. In Section 3, we present the BSC model of error-tolerant side-channel cube attack (ET-SCCA). The decoding algorithms are developed and analyzed in Section 4. Section 5 describes the evaluation of ET-SCCA based on the application to PRESENT. In Section 6 we compare ET-SCCA with other side-channel attacks and provide some countermeasures. Finally, we conclude the paper in Section 7.

2 Preliminaries

2.1 Cube and Side-Channel Cube Attacks

Cube attacks were introduced by Dinur and Shamir at Eurocrypt 2009 [8]. It is closely related to high-order differential attacks [18] and algebraic IV differential attacks [29][30]. The differences between cube attack and high order differential attack are elaborated in [12]. Cube attacks consist of two phases: the off-line phase and the on-line phase. The off-line phase determines which queries should be made to a cryptosystem during the on-line phase of the attack. It is performed once per cryptosystem. Note that the knowledge of the internal structure of the cipher is not necessary. In the on-line phase, attackers deduce a group of linear equations by querying the cryptosystem with tweakable public variables (e.g., chosen plaintexts). Finally, the attacker solves the linear equations to recover the secret key bits. We give a toy example below.

Consider a block cipher $\mathbb{T}$ and its encryption function $(c_1, ..., c_m) = \mathrm{E}(k_1, ..., k_n, v_1, ..., v_m)$, where c_i, k_j and v_s are ciphertext, encryption key and plaintext bits, respectively. One can always represent c_i, $i \in [1, m]$, with a multivariate polynomial in the plaintext and key bits, namely, $c_i = p(k_1, ..., k_n, v_1, ..., v_m)$. The polynomial p is called a *master polynomial* of c_i.

Let $I \subseteq \{1, ..., m\}$ be an index subset, and $t_I = \prod_{i \in I} v_i$, the polynomial p is divided into two parts:

$$p(k_1, ..., k_n, v_1, ..., v_m) = t_I \cdot p_{S(I)} + q(k_1, ..., k_n, v_1, ..., v_m),$$

where no item in q contains t_I. Here $p_{S(I)}$ is called the superpoly of I in p. A maxterm of p is a term t_I such that $\deg(p_{S(I)}) \equiv 1$, i.e., the superpoly of I in p is a linear polynomial which is not a constant.

Example 1. Let $p(k_1, k_2, k_3, v_1, v_2, v_3) = v_2v_3k_1 + v_2v_3k_2 + v_1v_2v_3 + v_1k_2k_3 + k_2k_3 + v_3 + k_1 + 1$ be a polynomial of degree 3 in 3 secret variables and 3 public variables. Let $I = \{2, 3\}$ be an index subset of the public variables. We can represent p as $p(k_1, k_2, k_3, v_1, v_2, v_3) = v_2v_3(k_1 + k_2 + v_1) + (v_1k_2k_3 + k_2k_3 + v_3 + k_1 + 1)$, where

$$t_I = v_2v_3,$$

$$p_{S(I)} = k_1 + k_2 + v_1,$$

$$q(k_1, k_2, k_3, v_1, v_2, v_3) = v_1k_2k_3 + k_2k_3 + v_3 + k_1 + 1.$$

Let d be the size of I, then a *cube* on I is defined as a set C_I of 2^d vectors that cover all possible combinations of t_I, while setting other public variables to be constant. Any vector $\tau \in C_I$ defines a new derived polynomial $p_{|\tau}$ with $n-d$ variables. Summing these derived polynomials over all the 2^d possible vectors in C_I results in exactly $p_{S(I)}$ (cf. Theorem 1, [8]). For p and I defined in Example 1, we have $C_I = \{\tau_1, \tau_2, \tau_3, \tau_4\}$, where

$$\tau_1 = [k_1, k_2, k_3, v_1, 0, 0], \tau_2 = [k_1, k_2, k_3, v_1, 0, 1],$$

$$\tau_3 = [k_1, k_2, k_3, v_1, 1, 0], \tau_4 = [k_1, k_2, k_3, v_1, 1, 1].$$

It is easy to verify that $p_{|\tau_1} + p_{|\tau_2} + p_{|\tau_3} + p_{|\tau_4} = k_1 + k_2 + v_1 = p_{S(I)}$. Here $p_{S(I)}$ is called the maxterm equation of t_I. In the off-line phase, the attacker tries to find as many maxterms and their corresponding maxterm equations as possible.

In the on-line phase, the secret key is fixed. The attackers choose plaintexts $\tau \in C_I$ and obtain the evaluation of p at τ. By summing up $p_{|\tau_i}$ for all the 2^d vectors in C_I, the attacker obtain $p_{S(I)}$, a linear equation in k_i. The attacker repeats this process for all the maxterms found in the off-line phase, and obtains a group of linear equations. If the number of independent equations is larger than or equal to n, the bit-length of the key, then the attacker can solve the linear equation system and recover the key.

2.2 Side-Channel Cube Attack

Side-channel cube attacks [10] use the knowledge about intermediate variables (i.e., state registers) as the target bits, and consequently the evaluation of p is obtained through side-channel leakage. Since side-channel leakage is likely to contain noise, solving the linear equation system becomes a challenge. To tackle this problem, Dinur and Shamir proposed to use error correction code to remove the measurement errors. In DS model, each measurement can have three possible outputs: 0, 1 and $\perp$, where $\perp$ indicates the measurement cannot be relied upon. The attacker assigns a new variable y_j to each $\perp$ and computes the maxterm equations. As a result, the maxterm equation has y_j on the right hand side. As for example 1, assuming the second measurement was not reliable, the obtained maxterm equation is now $k_1 + k_2 + v_1 = p_{|\tau_1} + p_{|\tau_3} + \perp + p_{|\tau_4}$. DS model replaces the $\perp$ in the maxterm equation with a new variable y_i. As a result, the equation becomes $k_1 + k_2 + v_1 = p_{|\tau_1} + p_{|\tau_3} + y_i + p_{|\tau_4}$. For each cube, there might be new variable introduced. In order to solve these equations, additional measurements are required.

In the off-line phase, the attacker chooses a large cube of size k and computes all the coefficients of all the $\binom{k}{d-1}$ linear equations which are determined by summing over all the possible subcubes of dimension $d-1$. In the on-line phase, the attacker obtains 2^k leaked bits. Let ϵ be the fraction of the $\perp$ among all the measurements. Out of the 2^k values, $\epsilon \cdot 2^k$ values are $\perp$. It is assumed that the errors are uniformly distributed and the leakage function is a d-random multivariate polynomial. More precisely, the definition of d-random polynomial [8] is as follows.

Definition 1. *A d-random polynomial with $n+m$ variables is a polynomial $p \in \mathbb{P}_d^{n+m}$ such that each possible term of degree d which contains one secret variable and $d-1$ public variables is independently chosen to occur with probability 0.5, and all the other terms can be chosen arbitrarily.*

Let n be the number of secret key variables. The attacker chooses a big cube with $k \geq d + \log_d^n$ public variables[1]. The attacker obtains a system of $\binom{k}{d-1}$ linear equations in the $\epsilon \cdot 2^k + n$ variables y_j and k_i. As far as $\binom{k}{d-1} \geq (\epsilon \cdot 2^k + n)$, the attacker can solve the linear equations and obtain the key. The error ratio ϵ should satisfy the following condition:

$$\epsilon \leq \frac{\binom{k}{d-1} - n}{2^k}. \tag{1}$$

The attacker can thus find the key when at most $\frac{\binom{k}{d-1}-n}{2^k}$ fraction of the leaked bits are $\perp$. This model was further enhanced in [11] by using more trivial equations of high dimension cubes to correct the errors. The number of measurements increased exponentially when k increases. Such a large amount of measurements is hard to obtain in side-channel analysis, especially in power analysis. Note that the success of this model is based on the assumption that the attacker knows which measurement is correct and which one is not. This is a strong assumption since in reality every measurement is likely to be noisy. In the following section, we consider a more practical model where each measurement is noisy.

3 A New Error-Tolerant Side-Channel Cube Attack

Note that all the coefficients of maxterm equations can be obtained in the off-line phase. Suppose we can derive L linear equations in the off-line phase and the average cube size of all the corresponding maxterms is $\bar{d}$, then we have a linear equation system as follows:

$$\begin{cases} l_1: \ a_1^1 k_1 + a_1^2 k_2 + ... + a_1^n k_n = b_1 \\ l_2: \ a_2^1 k_1 + a_2^2 k_2 + ... + a_2^n k_n = b_2 \\ \quad \vdots \\ l_L: \ a_L^1 k_1 + a_L^2 k_2 + ... + a_L^n k_n = b_L \end{cases} \tag{2}$$

where $a_i^j \in \{0,1\}$ $(1 \leq i \leq L, 1 \leq j \leq n)$ denotes the coefficient of a linear equation. Note that $b_i \in \{0,1\}$ is obtained by summing up the evaluation of the maxterm equation over the i^{th} cube C_i, namely, $b_i = \sum_{\tau \in C_i} p_{|\tau}$. The value of $p_{|\tau}$ is obtained via measurements. Ideally, the measurement is error-free and the attacker obtains the correct sequence $B = [b_1, b_2, ..., b_L]$. In reality, however, the attacker is likely to observe a different sequence $Z = z_1, z_2, ..., z_L$ due to the measurement errors.

[1] we only need about $d + log_d^n$ tweakable public variables in order to pack n different maxterms among their products, since $\binom{d+log_d^n}{d} \approx d^{log_d^n} = n$.

Let q be the probability that the bit may flip in the observation of each measurement. We can assume $q < 1/2$, then $1 - q = 1/2 + \mu$ is the probability that we get an accurate measurement and $\mu = 0$ means a random guess. Since $b_i = \sum_{\tau \in C_i} p_{|\tau}$, and C_i has $t = 2^{\bar{d}}$ elements, and each measurement can be treated as an independent event, according to the piling-up lemma [16], we can derive

$$Pr\{b_i = z_i\} \triangleq 1 - p = \frac{1}{2} + 2^{t-1}\mu^t. \tag{3}$$

Thus, the observed sequence $Z = z_1, z_2, ..., z_L$ can be regarded as the received channel output and the sequence $B = b_1, b_2, ..., b_L$ is regarded as a codeword from an $[L, n]$ linear block code, where L is the code length and n is the dimension. We can describe each z_i as the output of the binary symmetric channel (BSC, see Fig.1) with $p = 1/2 - \varepsilon$ ($\varepsilon = 2^{t-1}\mu^t$) being the crossover probability.

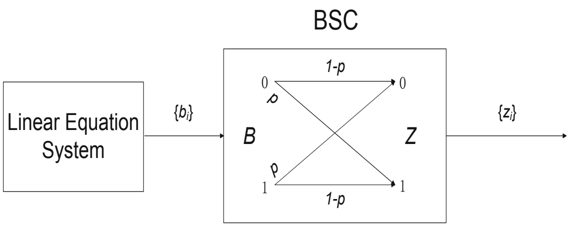

Fig. 1. The error-tolerant side-channel attack model

Therefore, the key recovery problem is now converted to the problem of decoding a $[L, n]$ linear code. Let $H(x) = -x\log_2 x - (1 - x)\log_2(1 - x)$ be the binary entropy function, if the code rate $R = n/L$ is less than the capacity $C(p) = 1 - H(p)$, then in the ensemble of random linear $[L, n]$ codes, the decoding error probability approaches zero. Various decoding techniques can be adopted to recover the secret key.

4 Decoding Algorithms

4.1 Maximum Likelihood Decoding (ML-Decoding)

Siegenthaler [28] firstly proposed the use of ML-decoding in cryptanalysis of a stream cipher by exhaustively searching through all the codewords of the above $[L, n]$-code. The complexity of this algorithm is about $O(2^n \cdot n/C(p))$. We give a brief introduction of ML-decoding below.

Let $A = (a_i^j)_{L \times n}$ $(1 \leq i \leq L, 1 \leq j \leq n)$ be the generator matrix of (2) and A_i denote the i-th row vector of A. The aim of the decoding is to find the closet codeword $(b_1, b_2, ..., b_L)$ to the received vector $(z_1, z_2, ..., z_L)$, and decode the key variables $\mathbf{k} = (k_1, k_2, ..., k_n)$ such that $b_i = \mathbf{k} \cdot A_i^T$, where T denotes the matrix transpose, i.e., find such $\mathbf{k}$ that minimizes $D(\mathbf{k}) = \sum_{i=1}^{L}(z_i \bigoplus b_i)$.

It is known that ML-decoding is optimal since it has the smallest error probability among all decoding algorithms. We can apply the ML-decoding to the code with length L satisfying the inequality $n/L < C(p)$, that is $L > n/C(p)$. Recall that $p = 1/2 - \varepsilon$, we can approximate $C(p)$ as $C(p) \approx \varepsilon^2 \cdot 2/(ln(2))$. Simulations [28] show that the critical length $L = l_0 \approx 0.35 \cdot n \cdot \varepsilon^{-2}$ provides the probability of successful decoding close to $1/2$, while for $L = 2l_0$ the probability is close to 1.

4.2 Error Probability Evaluation

In our model, we can get the following theorem on the theoretical relationship.

Theorem 1. *If we derive L linear equations containing n key variables and the average cube size of all the corresponding maxterms is $\bar{d}$, then we can recover all the n key bits with success probability close to 50% when the error probability q of each measurement satisfies*

$$q \leq \frac{1}{2} \cdot (1 - (\frac{0.35 \cdot n}{L})^{\frac{1}{2 \cdot t}} \cdot 2^{\frac{1}{t}}), \tag{4}$$

where $t = 2^{\bar{d}}$ denotes the number of summations to evaluate each linear equation.

Proof. In order to have a probability of successful decoding close to $1/2$ using the ML-decoding, the code length L should be larger than $0.35 \cdot n \cdot \varepsilon^{-2}$, that is $L \geq 0.35 \cdot n \cdot \varepsilon^{-2}$. Thus we get $\varepsilon \geq \sqrt{\frac{0.35 \cdot n}{L}}$. Since $\varepsilon = 2^{t-1}\mu^t$ holds, then we can derive $\mu \geq (\frac{0.35 \cdot n}{L})^{\frac{1}{2 \cdot t}} \cdot 2^{\frac{1}{t}-1}$. From $q = 1/2 - \mu$, we have $q \leq \frac{1}{2} \cdot (1 - (\frac{0.35 \cdot n}{L})^{\frac{1}{2 \cdot t}} \cdot 2^{\frac{1}{t}})$. $\square$

Suppose the number of key variables is $n = 80$, the error probability can be depicted in the following figure.

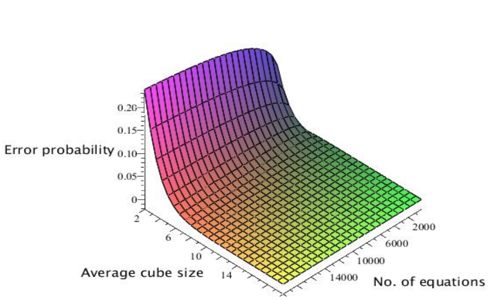

Fig. 2. Error probability q as a function of $\bar{d}$ and L (Given $n = 80$)

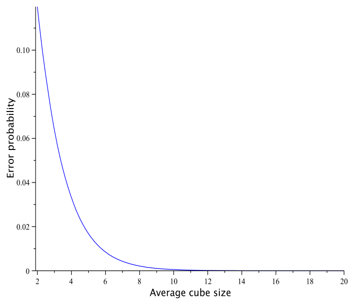

Fig. 3. Error probability q as a function of $\bar{d}$ (Given $L = 1000$, $n = 80$)

Theorem 1 gives an explicit equation to compute the error tolerance q. Fig. 2 shows that the error probability q as a function of L and $\bar{d}$. To ensure a higher error tolerance, the attacker needs to derive as many maxterm equations as possible, while keeping the corresponding cube size as low as possible. Fig. 3 shows the relationship between error probability q and the average cube size when the number of linear equations L is fixed. Note that the error probability q is exponentially decreased when the cube size increases.

Under the assumption that the master polynomial is a d-random multivariate polynomial, $L = \binom{k}{d-1}$ linear equations (containing n key variables) can be derived with the corresponding maxterm size of $d-1$. Then we get the following corollary.

Corollary 1. *If the master polynomial is a d-random multivariate polynomial and we choose a big cube with $k \geq d + log_d^n$ public variables, then we can recover all the n key bits with success probability close to 50% when the error probability q of each measurement satisfies*

$$q \leq \frac{1}{2} \cdot (1 - (\frac{0.35 \cdot n}{\binom{k}{d-1}})^{\frac{1}{2 \cdot t}} \cdot 2^{\frac{1}{t}}), \tag{5}$$

where $t = 2^{d-1}$ denotes the number of summations to evaluate each maxterm equation.

4.3 Improving the Success Rate and Decoding Complexity

When applying side-channel cube attacks to a specific cryptosystem, the number of linear equations we can derive might be limited. In other words, the code length L may not be big enough to reach a high probability of successful decoding. In this case, the decoding algorithm is likely to output wrong key, which is not far from the correct key. To overcome this problem, we output a list of candidates of the key and verify each solution using a valid plaintext/ciphertext pair.

When n becomes larger, the ML-decoding process becomes expensive since it has a time complexity of 2^n. This problem can be solved if the linear equations can be divided into almost disjoint sets. We first divide the set $\{k_1, k_2, ..., k_n\}$ into η groups $G_1, G_2, ..., G_\eta$, each with roughly $\lceil n/\eta \rceil$ key variables. For each group G_i, we collect those linear equations only containing the secret variables in G_i. The ML-decoding in each G_i has a complexity of $O(2^{\lceil n/\eta \rceil} \cdot \lceil n/\eta \rceil / C(p))$. Note that the linear equations are likely to be sparse, which makes the splitting strategy easy to apply. Previous study on Trivium [8], Serpent [11,10] and KATAN [15] shows that the linear equations generated by cube attacks are indeed sparse.

Note that the ML-decoding is not the only decoding algorithm of linear binary codes. In fact, since most of the linear equations derived from the cube summations have a low density, other decoding algorithms [31,14,21,6] that exploit this properties may achieve better results. We do not claim to be experts in the design and usage of coding. However, in this study, we want to highlight the importance of the procedure of transforming the side-channel cube attack within noise leakage to the decoding of a binary linear code.

5 Evaluation of Our ET-SCCA on PRESENT

To check the correctness and the efficiency of the proposed model, we apply it to PRESENT, a standardized round based lightweight block cipher. Details of the cipher structure can be found in [5]. Previous results of cube attacks on PRESENT [19,32,27] assume completely error-free measurements.

5.1 Hamming Weight Leakage

Like previous attacks [25,23,26], we assume the PRESENT cipher is implemented on a 8-bit processor. The attacker exploits the hamming weight leakage when the intermediate variables (state variables) are loaded from the memory to the ALU. Let $w_H(x)$ be the Hamming weight function which outputs the number of 1s in x. Let $S = \{s_0, s_1, ..., s_7\}$ be a 8-bit internal state, then the value of $w_H(S)$ can be represented with a 4-bit value $H = \{h_0, h_1, h_2, h_3\}$ and h_0 denotes the least significant bit (LSB) and h_3 denotes the most significant bit (MSB). Each h_i, $0 \leq i \leq 3$ can be calculated[2] as $h_0 = \sum_{i=0}^{7} s_i$, $h_1 = \sum_{(0 \leq i<j \leq 7)} s_i s_j$, $h_2 = \sum_{(0 \leq i<j<m<l \leq 7)} s_i s_j s_m s_l$, $h_3 = \prod_{i=0}^{7} s_i$. From the expression of each h_i, the algebraic degree increases from LSB to MSB and each h_i contains all the 8 internal state bits.

5.2 Cube Searching Strategy

The cube searching strategy in our attacks is as follows. We keep two types of monomials for each round, one involving a single key variable and the other only involving public variables. Then in the next round we compute the terms in the polynomial of the state bit which are related to the selected terms only. And we discard other terms involving more than one key variables. In this way, we can explicitly compute the multivariate polynomials in the key variables and plaintext variables for each state bit in the first few rounds of PRESENT and treat the coefficient of the linear terms and constant terms as cubes.

5.3 Simulations on the Second Round

As shown above, in order to have a high error tolerance rate, the cube size should not be too big. We start with by attacking the second round. The internal state contains 8 bytes denoted by $byte_1, byte_2, ..., byte_8$. In the off-line phase, we have searched each state byte using our cube searching strategy. If the LSB of the Hamming weight of $byte_8$ after the second round is leaked, we can in total obtain $L = 2232$ linear equations containing 64-bit key variables. The problem of recovering those 64-bit key is now equivalent to the problem of decoding a $[2232, 64]$ linear code.

Since a direct application of the ML-decoding algorithm has a time complexity of 2^{64} attempts, we divide all the key variables into 4 groups G_1, G_2, G_3 and G_4

[2] All the summations are based on finite field $\mathbb{F}_2$.

and apply the ML-decoding in each group. To ensure the success probability, we save a candidate list of the T closest solutions for each group. However, the number of candidates becomes T^4, leading to an expensive verification step. A more efficient way is to use overlapping groups where each group shares with neighboring groups with 3-4 key bits. Now we only have to verify the combination of candidates that agree in the overlapping bits, which can reduce the number of verifications by a factor of about 2^9 to 2^{12}.

The grouping strategy here is to keep the code rate of each group as low as possible by utilizing the sparse structure of the linear maxterm equations, since it can further accelerate the decoding phase and the verification phase. Table 2 shows the configurations of the 4 groups and their overlapping bits.

Table 2. Groups on the LSB of the Hamming weight of $byte_8$

Group	$[L, n]$	*Key bits*	*Overlapping bits*
G_1	$[690, 19]$	$[k_{17}, k_{18}, ..., k_{35}]$	3 with G_2
G_2	$[690, 19]$	$[k_{33}, k_{34}, ..., k_{51}]$	3 with G_1, 3 with G_3
G_3	$[690, 19]$	$[k_{49}, k_{50}, ..., k_{67}]$	3 with G_2, 3 with G_4
G_4	$[558, 16]$	$[k_{65}, k_{66}, ..., k_{80}]$	3 with G_3

Using the same strategy, we also group the key variables in all the $L = 10468$ maxterm equations containing $n = 64$ key variables on the attack of the second LSB leakage of the Hamming weight of $byte_1$. The configurations are listed in Table 3.

Table 3. Groups on the 2nd LSB of the Hamming weight of $byte_1$

Group	$[L, n]$	*Key bits*	*Overlapping bits*
G_1	$[2971, 20]$	$[k_{17}, k_{18}, ..., k_{36}]$	4 with G_2
G_2	$[2971, 20]$	$[k_{33}, k_{34}, ..., k_{52}]$	4 with G_1, 4 with G_3
G_3	$[2971, 20]$	$[k_{49}, k_{50}, ..., k_{68}]$	4 with G_2, 4 with G_4
G_4	$[2671, 16]$	$[k_{65}, k_{66}, ..., k_{80}]$	4 with G_3

Under these configurations, we have simulated the decoding algorithm for 100 runs with $T = 200$. For each run, we randomly generate a key and construct the linear code in each group. The noise was simulated by a random binary number generator according to the crossover probability p (e.g., suppose $k_0 = 1, k_1 = 0$ and there is a linear equation $1 + k_0 + k_1 = z_i$, the value of z_i will flip to 1 with probability p and remain unchanged with probability $1 - p$). We have conducted the simulation for 10 times and the average number of successful decoding out of a batch of 100 runs are recorded. The simulation results with various crossover probability are given in Fig.4.

From Fig.4, with the crossover probability $p = 0.44$, the decoding success probability of the LSB leakage of $byte_8$ is 61.10%. When $p = 0.47$, the decoding success probability of the 2nd LSB leakage of $byte_1$ is 54.10%. Due to the lower code rate, the decoding success probability of 2nd LSB leakage is relatively higher than that of the LSB leakage.

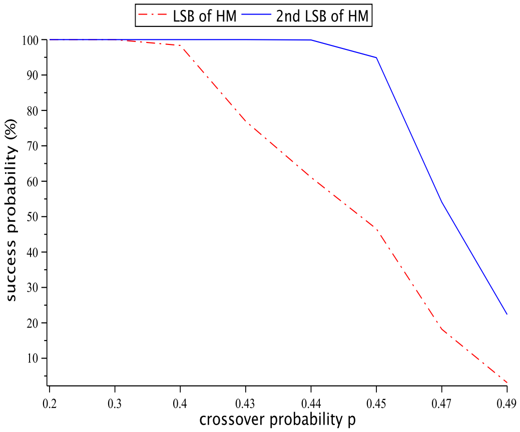

Fig. 4. Simulation results of list decoding

The value of p is the crossover probability for each evaluation of the maxterm equations. In order to derive the crossover probability (error probability) $q = 1/2-\mu$ for each measurement from equation (3). We need to calculate the average cube size for all the maxterms. The results are summarized in Table 4.

Table 4. Noise level under different leakage positions

HW Leakage position	*Round*	p	*Average cube size d*	q
LSB ($byte_8$)	2	44%	7.7	0.6%
2nd LSB ($byte_1$)	2	47%	8.4	0.4%

The whole attack contains two phases, the first phase is the decoding in each group. The results in this phase are the candidate lists. Let t_i denote the time complexity of decoding in group G_i, m denote the number of the groups and n_i denote the code dimension in G_i, thus the time complexity in this phase is $\sum_{i=1}^{m} t_i$ where $t_i = 2^{n_i}$ key trials. The second phase is the verification phase, the time complexity in this phase is $Q(T) = T^m/2^r$ encryptions, where T denotes the size of candidate list and 2^r is the reduction factor. Therefore, the total attack complexity is bounded by $max\{\ \sum_{i=1}^{m} t_i,\ T^m/2^r\ \}$. The attack results on PRESENT are given in Table 5.

It is clear to see that we can have an average successful probability of 61.1% to restore 64 key bits with a time complexity of $2^{20.6}$, negligible memory requirement and $q = 0.6\%$ error probability for each measurement. The rest 16 key bits can be exhaustively searched. Although the BSC model can tolerant noise in each measurement, the error tolerances are very low. The reason is that the cube size in the second round is relatively big. The bigger cube size will lead to an exponential increase of t in equation (3). Thus the error probability q become

Table 5. Attack results on PRESENT

Leakage position	*Time*[b]	*Memory requirement*[a]	*Data (measurements)*	r	*Success probability*	*Error probability*
LSB	$2^{20.6}$	3 KB	$2^{18.9}$	9	61.1%	0.6%
2nd LSB	$2^{21.6}$	3 KB	$2^{23.1}$	12	54.1%	0.4%

[a]4 candidate lists of $4 \cdot 200$ entries, with each entry of 4 bytes.
[b]The number of key trials.

very low. In the following section, we evaluate the model based on the leakage of the first round, which shows better results.

5.4 Simulation on the First Round

The diffusion of the first round is far from complete, thus our attack need to utilize more leaked bits instead of a single one to ensure the decoding success probability. Using our cube searching, we derived all the possible cubes from the LSB leakage of all the 8 bytes:$byte_1, byte_2, ..., byte_8$ after the first round. Then we perform the off-line phase by utilizing all those cubes and obtained hundreds of maxterm equations (see Appendix B). According to the key variables distribution in these maxterm equations, all the 8 bytes can be classified into 2 classes in Table 6.

Table 6. Classification of state bytes after the first round

Class	*State byte*	*Key variables*[a]	*No. of maxterm equations*	*Average cube size*
$Class_1$	$byte_1, byte_3, byte_5, byte_7$	$k_{17}, k_{18}, ..., k_{48}$	150	1.90
$Class_2$	$byte_2, byte_4, byte_6, byte_8$	$k_{49}, k_{50}, ..., k_{80}$	152	1.89

[a]The number of key variables for both classes are 32.

From Table 6, the average cube size for both classes are relatively smaller than that of the second round. The grouping strategy is the same to that described in section 5.2. We combine $Class_1$ and $Class_2$ and divide them into 4 groups in Table 7.

Table 7. Groups on the LSB of the Hamming weight after the first round

Group	$[L, n]$	*Key bits*	*Overlapping bits*
G_1	$[93, 20]$	$[k_{17}, k_{18}, ..., k_{36}]$	4 with G_2
G_2	$[95, 20]$	$[k_{33}, k_{34}, ..., k_{52}]$	4 with G_1, 4 with G_3
G_3	$[95, 20]$	$[k_{49}, k_{50}, ..., k_{68}]$	4 with G_2, 4 with G_4
G_4	$[76, 16]$	$[k_{65}, k_{66}, ..., k_{80}]$	4 with G_3

We also simulate the decoding algorithm for 100 runs with $T = 200$ and various crossover probabilities. We conduct the simulation for 10 times and record the average number of successful cases. Results show that when $p = 42\%$ the decoding success probability is 50.1%. Thus the error probability of each measurement is $q = 19.4\%$. The results are summarized in the following Table 8.

Table 8. ET-SCCA on the first round

Leakage position	*Time*[b]	*Memory requirement*[a]	*Data (measurements)*	r	*Success probability*	*Error probability*
LSB	$2^{21.6}$	3 KB	$2^{10.2}$	12	50.1%	19.4%

[a] 4 candidate lists of $4 \cdot 200$ entries, with each entry of 4 bytes.
[b] The number of key trials.

Compared to the attack on the second round, a higher error probability is achieved based on the leakage of the LSB of the state bytes after the first round. We can have a success probability of 50.1% to recover all the 64 master key bits diffused in the first round with time complexity of $2^{21.6}$, negligible memory requirement and $2^{10.2}$ data complexity, when the error probability of each measurement is at most 19.4%. We can further reduce the data complexity to 442 by utilizing partial leaked bits after the first round, while the error tolerance level also reduced to about 9.7% accordingly (see Appendix A for details). The data complexity (measurements) in our estimation is the upper bound, since when we target on the multiple bytes in the first round, we can reduce the measurements by reusing the duplicate cubes.

Note that the time complexity can be further reduced by splitting the maxterms into more groups with similar size to each other. Thus the cost for decoding can be reduced, while the number of candidates increases (we can also reduce the time complexity of verification phase by introducing more overlapping bits) and the decoding success probability and error probability may also change accordingly. There is a tradeoff between time, success probability and error probability. We do not claim that our grouping is optimal as there may be better choices.

These results demonstrate that the cube size has a great influence on the error tolerance, which is consistent with our previous analysis. To maximize the efficiency, it is required to apply the attack to the early rounds of a cipher, in which the algebraic degree of the state is relatively low. Even though we may derive more maxterm equations in the later round to gain a higher crossover probability p for each equation, the error probability q for each measurement will drop very quickly due to the higher cube size.

6 Comparison and Discussion

Our model can be viewed as an instantiation of side-channel cube attack in the presence of noise. Compared to the DS model, our model has weaker assumptions, namely, we allow possible errors in each measurement. As a consequence, the error-correction strategy is also different. We regard the key recovery problem as the decoding of some linear codes transmitted through the BSC, while the DS model considers it as erasure codes.

Note that the DS model performs 2^k measurements, while our model performs less than $L \cdot 2^{\bar{d}}$ measurements. For both models, the number of traces grows exponentially when the cube size goes up. The DS model targets the leakage round where a complete diffusion is achieved [11]. Taking PRESENT as an example, after three rounds, d is around 20. Thus, to maximize the error tolerance according to (1), k should be roughly 40. The attacker needs to perform 2^{40} measurements, which is order of magnitude larger than that of standard DPA.

The original algebraic side-channel attack (ASCA) [25,26] is sensitive to measurement noise and the theoretical estimates of the attack complexities are hard to derive. In addition, a large number of leakage information is required when feeding the Biryukov-Cannière system [3] into the algebraic solvers. This attack was later improved by Oren *et al.* [23] to handle more noisy leakage. They consider the key recovery problem as a pseudo-Boolean optimization (PBOPT) problem. However, a theoretic estimation of the error tolerance is missing. The ASCA exploits multiple information leakage on a single power trace, while the side-channel cube attack uses leakage from a single wire in many executions. The ASCA also claims to be able to break masked implementations, while it is not clear yet if ET-SCCA can also break masking.

In order to prevent side-channel cube attack, the design should add more noise to increase the probability of measurement errors. Many known techniques can be used here.

- Noise generation. The noise generator actively flattens the power trace with noise.
- Dual-rail logic. Dual-rail logic hides data-dependent flips inside the combinational logic and registers. It helps to reduce the signal-to-noise ratio.
- Data-bus encryption. When the bus between the ALU and memory is encrypted, it is more difficult for the attackers to obtain the hamming weight of the data.
- Random execution order. The order of internal operations (e.g. substitution) in each round can be randomized. It becomes more difficult for the attackers to locate the leakage of the target bit.

Note that countermeasures listed above are originally designed to thwart power analysis. If the EM analysis is used, some of this countermeasures may be ineffective. For example, noise generation is not likely to prevent EM analysis that uses only local EM leakage.

7 Conclusion and Open Problems

In this paper, we have presented a new and more realistic model for side-channel cube attacks. The new model regards the measurement as the output of a binary symmetric channel and the key recovery problem is converted to decoding a linear code. We theoretically analyzed the error tolerance capacity of the new model and verified our results with simulations on PRESENT. We also observed that since the encoding matrix is sparse, it's possible to speed up the decoding process using a divide-and-conquer strategy. Our simulation results on PRESENT show that given about $2^{10.2}$ measurements, each with an error probability of 19.4%, our model achieves 50.1% of success rate for the key recovery.

The study of side-channel cube attack is still at its early age. Here we list several open problems.

1. How to select the best target bit and find more maxterm equations. The current random walk method is very time-consuming.
2. Can side-channel cube attacks break masked implementations?
3. How to increase the error tolerance efficiently?
4. When using our new model, can we speed up the decoding process further by exploiting the sparse structure of encoding matrix?

References

1. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009)
2. Aumasson, J.-P., Dinur, I., Henzen, L., Meier, W., Shamir, A.: Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128. In: Special Purpose Hardware for Attacking Cryptographic Systems, SHARCS 2009 (2009)
3. Biryukov, A., De Cannière, C.: Block Ciphers and Systems of Quadratic Equations. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 274–289. Springer, Heidelberg (2003)
4. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. Journal of Computer and System Sciences 47, 549–595 (1993)
5. Bogdanov, A., Knudsen, L., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
6. Chung, S.-Y., Forney, G.D., Richardson, T., Urbanke, R.: On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. IEEE Communications Letters 5(2), 58–60 (2001)
7. Courtois, N., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002)
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. Dinur, I., Shamir, A.: Breaking Grain-128 with Dynamic Cube Attacks. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 167–187. Springer, Heidelberg (2011)
10. Dinur, I., Shamir, A.: Side Channel Cube Attacks on Block Ciphers. Cryptology ePrint Archive. Report 2009/127 (2009)
11. Dinur, I., Shamir, A.: Generic Analysis of Small Cryptographic Leaks. In: 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 39–48 (2010)

12. Dinur, I., Shamir, A.: Applying cube attacks to stream ciphers in realistic scenarios. Cryptography and Communications (4), 217–232 (2012)
13. Farebrother, R.W.: Linear Least Squares Computations. STATISTICS: Textbooks and Monographs (1988) ISBN 978-0-8247-7661-9
14. Gallager, R.G.: Low-density parity-check codes. IRE Transactions on Information Theory 8(1), 21–28 (1962)
15. Bard, G.V., Courtois, N.T., Nakahara Jr., J., 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)
16. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
17. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
18. Lai, X.: Higher Order Derivatives and Differential Cryptanalysis. Communications and Cryptography: Two Sides of One Tapestry, 227 (1994)
19. Yang, L., Wang, M., Qiao, S.: Side Channel Cube Attack on PRESENT. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 379–391. Springer, Heidelberg (2009)
20. Luby, M.G., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D.A.: Efficient erasure correcting codes. IEEE Transactions on Information Theory 47(2), 569–584 (2001)
21. MacKay, D.: Good error correcting codes based on very sparse matrices. IEEE Transactions on Information Theory 45(2), 399–431 (1999)
22. Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
23. Oren, Y., Kirschbaum, M., Popp, T., Wool, A.: Algebraic Side-Channel Analysis in the Presence of Errors. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 428–442. Springer, Heidelberg (2010)
24. Quisquater, J.J., Samyde, D.: A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions: the SEMA and DEMA methods[EB/OL]. In: Eurocrypt Rump Session (2000)
25. Renauld, M., Standaert, F.-X.: Algebraic Side-Channel Attacks, Cryptology ePrint Archive, report 2009/179 (2009), http://eprint.iacr.org/2009/279
26. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N.: Algebraic side-channel attacks on the AES: Why time also matters in DPA. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 97–111. Springer, Heidelberg (2009)
27. Shekh Faisal, A.-L., Mohammad, R.R., Willy, S., Jennifer, S.: Extended Cubes: Enhancing the cube attack by Extracting Low-Degree Non-linear Equations. In: Bruce, C., Lucas, C.K.H., Ravi, S., Duncan, S. (eds.) ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, pp. 296–305 (2011)
28. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Transactions on Computers C-34, 81–85 (1985)
29. Vielhaber, M.: Breaking ONE.TRIVIUM by AIDA and Algebraic IV Differential Attack. IACR Cryptology ePrint Archive, 413 (2007)
30. Vielhaber, M.: AIDA Breaks (BIVIUM A and B) in 1 Minute Dual Core CPU Time. IACR Cryptology ePrint Archive, 402 (2009)

31. Wiberg, N.: Codes and decoding on general graphs. Ph.D. dissertation. Linköping University, Linköping, Sweden (1996)
32. Zhao, X.J., Wang, T., Guo, S.Z.: Improved Side Channel Cube Attacks on PRESENT. Cryptology ePrint Archive. Report 2011/165 (2011)

A Reducing Data Complexity

We can further reduce the data complexity by utilizing the partial leaked bits after the first round listed in the following Table 9.

Table 9. Classification of partial state bytes after the first round

Class	*State byte*	*Key variables*[a]	*No. of maxterm equations*	*Average cube size*
$Class_3$	$byte_1, byte_3$	$k_{17}, k_{18}, ..., k_{48}$	62	1.70
$Class_4$	$byte_2, byte_4$	$k_{49}, k_{50}, ..., k_{80}$	64	1.75

[a]The number of key variables for both classes are 32.

Using the same grouping strategy described in section 5.3, the attack results are summarized in the following Table 10.

Table 10. ET-SCCA on the first round

Leakage position	*Time*	*Memory requirement*[a]	*Data (measurements)*	*r*	*Success probability*	*Error probability*
LSB	$2^{25.9}$	3 KB	442	12	53.2%	9.7%

[a]4 candidate lists of $4 \cdot 200$ entries, with each entry of 4 bytes.

These results demonstrate that we can further reduce the data complexity to 442. Since we only utilize the partial leaked information to decode, the error probability also reduced.

B Maxterms and Maxterm Equations

Table 11. 150 maxterms and maxterm equations obtained from the LSB of $byte_1, byte_3, byte_5, byte_7$

Cube Indexes	Maxterm equations	Cube indexes	Maxterm equations
{2}	k19	{3}	1 + k18
{6}	k23	{7}	1 + k22
{11}	1 + k26	{14}	k31
{15}	1 + k30	{18}	k35
{19}	1 + k34	{22}	k39
{23}	1 + k38	{26}	k43
{27}	1 + k42	{30}	k47
{31}	1 + k46	{1, 3}	k18 + k20
{1, 4}	k18 + k19	{2, 3}	k17
{2, 4}	1 + k17	{3, 4}	1 + k17
{5, 6}	k23 + k24	{5, 7}	k22 + k24
{5, 8}	k22 + k23	{6, 7}	k21
{6, 8}	1 + k21	{7, 8}	1 + k21
{1, 2}	1 + k20	{1, 3}	k20
{1, 4}	1 + k18 + k19	{2, 4}	1 + k17
{3, 4}	k17	{5, 6}	1 + k24
{5, 7}	k24	{5, 8}	1 + k22 + k23
{6, 8}	1 + k21	{7, 8}	k21
{1, 2}	k19 + k20	{1, 3}	k18 + k20
{1, 4}	k18 + k19	{2, 3}	1 + k17
{2, 4}	k17	{3, 4}	k17
{5, 6}	k23 + k24	{5, 7}	k22 + k24
{5, 8}	k22 + k23	{6, 7}	1 + k21
{6, 8}	k21	{7, 8}	k21
{9, 10}	k27 + k28	{9, 11}	k26 + k28
{9, 12}	k26 + k27	{9, 10}	1 + k28
{9, 11}	k28	{9, 12}	1 + k26 + k27
{9, 10}	k27 + k28	{9, 11}	k26 + k28
{9, 12}	k26 + k27	{10, 11}	k25
{10, 12}	1 + k25	{11, 12}	1 + k25
{13, 14}	k31 + k32	{13, 15}	k30 + k32
{13, 16}	k30 + k31	{14, 15}	k29
{14, 16}	1 + k29	{15, 16}	1 + k29
{17, 18}	k35 + k36	{17, 19}	k34 + k36
{17, 20}	k34 + k35	{18, 19}	k33
{18, 20}	1 + k33	{19, 20}	1 + k33
{21, 22}	k39 + k40	{21, 23}	k38 + k40
{21, 24}	k38 + k39	{22, 23}	k37
{22, 24}	1 + k37	{23, 24}	1 + k37
{25, 26}	k43 + k44	{25, 27}	k42 + k44
{25, 28}	k42 + k43	{26, 27}	k41
{26, 28}	1 + k41	{27, 28}	1 + k41
{29, 30}	k47 + k48	{29, 31}	k46 + k48
{29, 32}	k46 + k47	{30, 31}	k45
{30, 32}	1 + k45	{31, 32}	1 + k45
{10, 12}	1 + k25	{11, 12}	k25
{13, 14}	1 + k32	{13, 15}	k32
{13, 16}	1 + k30 + k31	{14, 16}	1 + k29
{15, 16}	k29	{17, 18}	1 + k36
{17, 19}	k36	{17, 20}	1 + k34 + k35
{18, 20}	1 + k33	{19, 20}	k33
{21, 22}	1 + k40	{21, 23}	k40
{21, 24}	1 + k38 + k39	{22, 24}	1 + k37
{23, 24}	k37	{25, 26}	1 + k44
{25, 27}	k44	{25, 28}	1 + k42 + k43
{26, 28}	1 + k41	{27, 28}	k41
{29, 30}	1 + k48	{29, 31}	k48
{29, 32}	1 + k46 + k47	{30, 32}	1 + k45
{31, 32}	k45	{10, 11}	1 + k25
{10, 12}	k25	{11, 12}	k25
{13, 14}	k31 + k32	{13, 15}	k30 + k32
{13, 16}	k30 + k31	{14, 15}	1 + k29
{14, 16}	k29	{15, 16}	k29
{17, 18}	k35 + k36	{17, 19}	k34 + k36
{17, 20}	k34 + k35	{18, 19}	1 + k33
{18, 20}	k33	{19, 20}	k33
{21, 22}	k39 + k40	{21, 23}	k38 + k40
{21, 24}	k38 + k39	{22, 23}	1 + k37
{22, 24}	k37	{23, 24}	k37
{25, 26}	k43 + k44	{25, 27}	k42 + k44
{25, 28}	k42 + k43	{26, 27}	1 + k41
{26, 28}	k41	{27, 28}	k41
{29, 30}	k47 + k48	{29, 31}	k46 + k48
{29, 32}	k46 + k47	{30, 31}	1 + k45
{30, 32}	k45	{31, 32}	k45

Table 12. 152 maxterms and maxterm equations obtained from the LSB of $byte_2, byte_4, byte_6, byte_8$

Cube Indexes	Maxterm equations	Cube indexes	Maxterm equations
{34}	k51	{35}	1 + k50
{38}	k55	{39}	1 + k54
{42}	k59	{43}	1 + k58
{46}	k63	{47}	1 + k62
{50}	k67	{51}	1 + k66
{54}	k71	{55}	1 + k70
{58}	k75	{59}	1 + k74
{62}	k79	{63}	1 + k78
{33, 34}	k51 + k52	{33, 35}	k50 + k52
{33, 36}	k50 + k51	{34, 35}	k49
{34, 36}	1 + k49	{35, 36}	1 + k49
{37, 38}	k55 + k56	{37, 39}	k54 + k56
{37, 40}	k54 + k55	{38, 39}	k53
{38, 40}	1 + k53	{39, 40}	1 + k53
{41, 42}	k59 + k60	{41, 43}	k58 + k60
{41, 44}	k58 + k59	{42, 43}	k57
{42, 44}	1 + k57	{43, 44}	1 + k57
{45, 46}	k63 + k64	{45, 47}	k62 + k64
{45, 48}	k62 + k63	{46, 47}	k61
{46, 48}	1 + k61	{47, 48}	1 + k61
{49, 50}	k67 + k68	{49, 51}	k66 + k68
{49, 52}	k66 + k67	{50, 51}	k65
{50, 52}	1 + k65	{51, 52}	1 + k65
{53, 54}	k71 + k72	{53, 55}	k70 + k72
{53, 56}	k70 + k71	{54, 55}	k69
{54, 56}	1 + k69	{55, 56}	1 + k69
{57, 58}	k75 + k76	{57, 59}	k74 + k76
{57, 60}	k74 + k75	{58, 59}	k73
{58, 60}	1 + k73	{59, 60}	1 + k73
{61, 62}	k79 + k80	{61, 63}	k78 + k80
{61, 64}	k78 + k79	{62, 63}	k77
{62, 64}	1 + k77	{63, 64}	1 + k77
{33, 34}	1 + k52	{33, 35}	k52
{33, 36}	1 + k50 + k51	{34, 36}	1 + k49
{35, 36}	k49	{37, 38}	1 + k56
{37, 39}	k56	{37, 40}	1 + k54 + k55
{38, 40}	1 + k53	{39, 40}	k53
{41, 42}	1 + k60	{41, 43}	k60
{41, 44}	1 + k58 + k59	{42, 44}	1 + k57
{43, 44}	k57	{45, 46}	1 + k64
{45, 47}	k64	{45, 48}	1 + k62 + k63
{46, 48}	1 + k61	{47, 48}	k61
{49, 50}	1 + k68	{49, 51}	k68
{49, 52}	1 + k66 + k67	{50, 52}	1 + k65
{51, 52}	k65	{53, 54}	1 + k72
{53, 55}	k72	{53, 56}	1 + k70 + k71
{54, 56}	1 + k69	{55, 56}	k69
{57, 58}	1 + k76	{57, 59}	k76
{57, 60}	1 + k74 + k75	{58, 60}	1 + k73
{59, 60}	k73	{61, 62}	1 + k80
{61, 63}	k80	{61, 64}	1 + k78 + k79
{62, 64}	1 + k77	{63, 64}	k77
{33, 34}	k51 + k52	{33, 35}	k50 + k52
{33, 36}	k50 + k51	{34, 35}	1 + k49
{34, 36}	k49	{35, 36}	k49
{37, 38}	k55 + k56	{37, 39}	k54 + k56
{37, 40}	k54 + k55	{38, 39}	1 + k53
{38, 40}	k53	{39, 40}	k53
{41, 42}	k59 + k60	{41, 43}	k58 + k60
{41, 44}	k58 + k59	{42, 43}	1 + k57
{42, 44}	k57	{43, 44}	k57
{45, 46}	k63 + k64	{45, 47}	k62 + k64
{45, 48}	k62 + k63	{46, 47}	1 + k61
{46, 48}	k61	{47, 48}	k61
{49, 50}	k67 + k68	{49, 51}	k66 + k68
{49, 52}	k66 + k67	{50, 51}	1 + k65
{50, 52}	k65	{51, 52}	k65
{53, 54}	k71 + k72	{53, 55}	k70 + k72
{53, 56}	k70 + k71	{54, 55}	1 + k69
{54, 56}	k69	{55, 56}	k69
{57, 58}	k75 + k76	{57, 59}	k74 + k76
{57, 60}	k74 + k75	{58, 59}	1 + k73
{58, 60}	k73	{59, 60}	k73
{61, 62}	k79 + k80	{61, 63}	k78 + k80
{61, 64}	k78 + k79	{62, 63}	1 + k77
{62, 64}	k77	{63, 64}	k77