Automated Dynamic Cube Attack on Block Ciphers: Cryptanalysis of SIMON and KATAN

Zahra Ahmadian[1], Shahram Rasoolzadeh[2,3], Mahmoud Salmasizadeh[3], and Mohammad Reza Aref[2]

[1] Department of Electrical Engineering, Shahid Beheshti University, Tehran, Iran
[2] Information Systems and Security Lab (ISSL), Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran,
[3] Electronic Research Institute, Sharif University of Technology, Tehran, Iran.
z_ahmadian@sbu.ac.ir,sh_rasoolzadeh@ee.sharif.edu,salmasi@sharif.edu, aref@sharif.edu

Abstract. A little work has ever been performed in cryptanalysis of block ciphers using cube technique. This paper presents a new framework for an efficient key recovery attack on block ciphers using a kind of dynamic cube attack. In this method, a cube tester is positioned at the middle of the cipher which is extended in two directions over the maximum possible upper and lower rounds, provided that some subkey bits are guessed. It is shown that an automated algorithm for this dynamic cube attack on block ciphers can be realized. Furthermore, we show its effectiveness on two lightweight block ciphers KATAN and SIMON. Our results show that this method can break 118 and 155 out of 254 rounds of KATAN32 in non-full-codebook and full-codebook attack scenarios, respectively. In the case of SIMON32/64, we succeed to cryptanalyse 17 and 22 out of 32 rounds, in the same scenarios. Both results show that it can absolutely compete with the well-established and mature methods of cryptanalysis of block ciphers, such as linear, differential and meet in the middle attack families.

Keywords: block cipher, cryptanalysis, cube attack, SIMON, KATAN

1 Introduction

In 2007, Vielhaber proposed the idea of Algebraic IV Differential Attack (AIDA) [1], which was then continued by Dinur and Shamir under the new title of *Cube attack* [2]. In this attack, the cipher is treated as a black box whose input consists of a public tweakable variable (e.g. IV in stream ciphers and plaintext in block ciphers) and a secret fixed parameter, (the key of the algorithm). The main idea behind this attack is to find some information about the secret key of the algorithm by choosing all values for an appropriate subset of the public input variables and summing up all the corresponding outputs.

This technique can also introduce a distinguisher, the so called *cube tester* in [3], where instead of retrieving some information from the key bits, a non-random property of the cipher is discovered by such an oracle access to the cipher.

Dynamic cube attack can be regarded as a more advanced version of the cube attack, in which a cube tester is employed not to detect a non-random property, but to determine if a specific guess for a subset of key bits could be correct or not [4]. In spite of the classic cube attack where the attacker sees the cipher as the black box and does not use the structural information of the algorithm, dynamic cube attack make use of such information, hence it can potentially reach better results than classic cube attack.

Cube attack family has shown to be very effective in cryptanalysis of lightweight stream ciphers. Grain [5] and Trivium [6] are two lightweight stream ciphers, for both of which the most successful attacks are a kind of cube attack family [1,2,3,4,7,8,9]. In contrast to its significant results in cryptanalysis of lightweight stream ciphers, apart from a combination of cube attack and algebraic attack [10], it has not yet been applied to block ciphers efficiently[4].

In this paper, we aim to take the first steps for an efficient application of cube technique for cryptanalysis of block ciphers. The technique that we propose is apparently similar to dynamic cube attack on stream ciphers [4], since it makes use of some distingushers to discard some wrong guesses of the secret key. Furthermore, some specific functions of the cube variables and secret key bits should be assigned to the input variables, similar to the so called *dynamic* variables in [4]. But, the mechanism by which these functions are derived are totally different. The main approach in the dynamic cube attack on stream ciphers is to use the recursive description of the cipher's output function in order to nullify/simplify some appropriate intermediate variables which consequently leads to a simplified algebraic function for the output bit. This process demands a "complex process that can not be fully automated and involves manual work to analyse the cipher" [4]. However, the situation is completely different in block ciphers, where a key-only-dependent function, i.e. the key schedule, is always available in every round of the cipher which enables the cryptanalyst to do partial encryption/decryption anywhere in the cipher, conditioned that she has guessed the involved subkeys. This option is not available in stream ciphers in any case, where the secret key as well as the IV are loaded into the state of the cipher at the first step, then they are mixed during the initialization phase so a key-only dependent function would never be conceivable then.

In spite of the very complex manual procedure of the dynamic cube attack on stream ciphers, our attack on block ciphers can be made fully automated. We put the r_c-round cube tester (distinguisher) in the middle of the cipher. Hence, in spite of all the cube attacks, the cube variables in our attack are not among the public input variables (i.e. plaintext bits for block ciphers) but they are basically some bits of an intermediate state. Then, we extend the attack to r_u rounds before, and r_l rounds after the distinguisher. Both the extensions potentially involve guessing some key bits. In addition, the backward r_u-round

[4] There are some instances of combinations of cube and side channel attacks [10,11,12,13] and cube and fault injection attacks [14] in the literature. These attacks are defined in the *leakage* and *fault injection* attack models, both of which are out of the model that we used for the attacker in this work.

Table 1. Result of Previous Attacks on KATAN32 and SIMON32/64

Algorithm	Type	Round	Time	Data	Memory	Ref.
KATAN32	Cube/Algebraic	79	14.72 min	20	-	[10]
	Conditional Differantial	78	2^{22}	2^{22}	-	[17]
	MITM ASR	110	2^{77}	138	$2^{75.1}$	[19]
	Differential	114	2^{77}	$2^{31.9}$	-	[18]
	Dynamic Cube	**118**	$\mathbf{2^{78.3}}$	$\mathbf{2^{19}}$	$\mathbf{2^{7.5}}$	**Sec. 4.2**
	MITM ASR	119	$2^{79.1}$	144	$2^{79.1}$	[19]
	Matchbox MITM	121	$2^{77.5}$	4	2^{5}	[20]
	Dynamic Cube	**138**	$\mathbf{2^{78.5}}$	$\mathbf{2^{31}}$	$\mathbf{2^{32.5}}$	**Sec. 4.2**
	Matchbox MITM	153	$2^{78.5}$	2^{5}	2^{76}	[20]
	Dynamic Cube	**155**	$\mathbf{2^{78.3}}$	$\mathbf{2^{32}}$	$\mathbf{2^{33.5}}$	**Sec. 4.2**
	Multidimensional MITM	175	$2^{79.3}$	3	$2^{79.58}$	[21]
SIMON32/64	Linear	11	-	2^{23}	-	[25]
	Linear (Matsui's 1st Alg.)	13	2^{32**}	2^{32}	-	[22]
	Impossible Differential	13	$2^{50.1}$	2^{30}	2^{20}	[25]
	Linear (Matsui's 2nd Alg.)	16	2^{54}	2^{32}	-	[22]
	Dynamic Cube	**17**	$\mathbf{2^{57}}$	$\mathbf{2^{18}}$	$\mathbf{2^{8}}$	**Sec. 5.2**
	Matchbox MITM	18	$2^{62.6}$	2^{3}	2^{55}	[23]
	Impossible Differential	18	$2^{61.1}$	2^{32}	$2^{47.7}$	[27]
	Differential*	18	2^{46}	$2^{31.2}$	2^{15}	[25]
	Multiple Linear	18	2^{32}	2^{32}	-	[22]
	Impossible Differential	19	$2^{62.6}$	2^{32}	2^{44}	[24]
	Differential	19	2^{32}	2^{31}	-	[26]
	Zero Correlation	20	2^{57}	2^{32}	$2^{41.4}$	[27]
	Linear Hull	20	$2^{59.7}$	$2^{31.7}$	-	[22]
	Integral	21	2^{63}	2^{31}	2^{54}	[27]
	Dynamic Cube	**22**	$\mathbf{2^{59}}$	$\mathbf{2^{32}}$	$\mathbf{2^{32.2}}$	**Sec. 5.2**

MITM: Meet In The Middle ASR: All Subkey Recovery

* Its probability of success is 63%.

** Its needed time for computing one bit of key. So, the total time complexity is 2^{63}.

extension determines what each bit of the plaintext must be exactly (i.e. its precise description in terms of the guessed key bits and cube variables). Finally, the set of key bits guessed in the two directions are tested by the cube tester.

Having introduced our attack framework in a general view, we examine its efficiency and flexibility on two lightweight block ciphers: the recently proposed NSA cipher, SIMON32/64 [15], and KATAN32 [16]. We found them good targets for our attack since both has a low-degree round function compensated by relatively large number of rounds, similar to stream cipher potential targets of cube attack [5,6].

The algorithm of our attack is flexible enough to set the maximum allowable value for data and time complexities. So, we report our results in two scenarios: full-codebook and non-full-codebook attacks. In case of KATAN, we could analyse up to 155, 138 and 118 rounds out of 254 rounds by full-codebook, half full-codebook and non-full-codebook attack scenarios, respectively, which absolutely

outperform the only instance of cube attack [10] which could break 79 rounds. In case of SIMON32/64, we could break 22-round and 17-round version of 32-round cipher in full-codebook and non-full-codebook scenarios, respectively.

As it can be seen in details in Table 1, for SIMON32, the 22-round full-codebook attack outperforms all other attacks formally published on this cipher [22,23,24,25,26,27], and in case of KATAN32 the 155-round full-codebook attack exceeds any other attacks proposed on this cipher, except the multidimensional meet in the middle attack [21], which has a significant distance not only from ours, but also from all other attacks proposed on KATAN [10,17,18,19,20].

Our results shows that in spite of the conventional view that cube attack is an appropriate tool just for cryptanalysis of stream ciphers, this newcomer attack can absolutely compete with the well-established and accepted methods for cryptanalysis of block ciphers such as differential, linear and meet in the middle attack families and even it can overtake most of them.

This paper is organized as follows: In Section 2, we give some preliminaries and notations for cube attacks. In Section 3, the proposed framework for dynamic cube attack on block ciphers is explained in a general view. We present our results on KATAN32 and SIMON32/64 in Sections 4 and 5, respectively. Finally we conclude our work in Section 6.

2 Preliminaries and Notations

Suppose $f : \{0,1\}^l \to \{0,1\}$ is the boolean function representing one output bit of the cipher based on the m-bit public input variable $P = \{p_{m-1}, ..., p_0\}$ and n-bit secret key $K = \{k_{n-1}, ..., k_0\}$ where $l = m + n$. Let $X = P \cup K$ be the set of all inputs of f. f can be represented as follows:

$$f(x_{l-1}, ..., x_0) = \sum_{i \in F_2^l} a_i \cdot x_{l-1}^{i_{l-1}} ... x_1^{i_1} x_0^{i_0} \tag{1}$$

where $\{i_{l-1}, ..., i_0\}$ is the binary representation of i, and a_i is a binary constant.

Now suppose that I is a certain index subset of $\{0, 1, ..., l-1\}$ of size d, namely *cube* where d is called the *cube dimension*. We denote the set of cube variables by $X_I = \{x_i | x_i \in I\}$. If we factorize function f by the monomial $t_I = \prod_{i \in I} x_i$, it can be written in the following form

$$f(x_{l-1}, ..., x_0) = t_I \cdot P_{S(I)} + q(x_{l-1}, ..., x_0) \tag{2}$$

where $P_{S(I)}$ is a polynomial that has no variable in common with t_I, and no monomial in q contains t_I. The $P_{S(I)}$ is called the *superpoly* of I.

The main idea behind all the variants of cube attack is to confine the cube variables to public input variables (i.e. $X_I \subseteq P$) and deal with the lower degree and simpler polynomial $P_{S(I)}$ rather than the potentially very complex polynomial f . To do so, we enjoy the following property of Boolean functions:

$$\sum_{X_I \in \{0,1\}^d} f(x_{l-1}, ..., x_0) = P_{S(I)} \tag{3}$$

Table 2. Notations

Symbol	Definition
m/n	block/key size for block cipher
I	cube
d	cube dimension
X_I	set of cube variables $\{x_i \mid i \in I\}$
$P_{S(I)}$	the superpoly for cube I
U/Z	index set for neutral/static bits
r_c	number of rounds in the cube tester part
r_u/r_l	number of rounds in the upper/lower extension part
$\mathcal{K}_u/\mathcal{K}_l$	set of key bits involved in the upper/lower extension part
$S^{(i)}$	intermediate state at the end of round i
S_j/S_J	bit j of intermediate state S, $S_J = \{S_j \mid j \in J\}$
T	distinguisher bit
N	number of tests
$p^{(j)}(X_I, \mathcal{K}_u)$	j^{th} description for the plaintext in X_I and $\mathcal{K}_u$
$Time/Data$	time/(upper bound for) data complexity
$Time_{max}/Data_{max}$	maximum allowable time/data complexity
$enc(S^{(i)}, \mathcal{K})$	one-round symbolic encryption of $S^{(i)}$ under key $\mathcal{K}$
$dec(S^{(i)}, \mathcal{K})$	one-round symbolic decryption of $S^{(i)}$ under key $\mathcal{K}$
$Enc(A, K, r_1, r_2)$	r_2-round partial encryption of $A \in \{0,1\}^m$ starting at round r_1 under key $K \in \{0,1\}^n$
$Dec(A, K, r_1, r_2)$	r_2-round partial decryption of $A \in \{0,1\}^m$ starting at round r_1 under key $K \in \{0,1\}^n$

In other words, if the attacker queries all the possible values of $X_I \in \{0,1\}^d$ from the encryption oracle where the other public variables are fixed, and sum up all the corresponding outputs, she will come up with the evaluation of $P_{S(I)}$ in which the other public variables have the same fixed values. The notations used in this paper are listed in Table 2.

3 Dynamic Cube Attack on Block ciphers

Apart from distinguishing attacks on hash functions MD5 and Keccak [3,28], all the noteworthy results published for the cube attack family imply its efficiency in crptanalysis of lightweight stream ciphers with a low degree round function compensated by a large number of initialization rounds. For Trivium [6], classic cube attack has led to cryptanalysis of the highest rounds analysed ever [2,3,7,8] and for Grain family [5], the best cryptanalytic results are reported by dynamic cube attack [4,9]. In spite of significant results in stream ciphers, no remarkable results on block ciphers using cube technique have appeared, by now. The only one is cryptanalysis of KATAN block cipher which analysed 79 rounds out of 254 rounds using a combination of cube and algebraic attack.

In this section we will present a framework for cryptanalysis of block ciphers using a kind of dynamic cube attack. We will show that, despite stream ciphers, this method can also be applied automatically in cryptanalysis of block ciphers. Due to the algebraic essence of this attack, those block ciphers with a low-degree round function compensated by a large number of rounds are potentially good targets for this attack. SIMON32/64 [15] and KATAN32 [16] are two instance of such ciphers that we have adopted to examine our method. For more details of our attacks parameters, see Table 1.

3.1 Attack Framework

As for all methods of cube attack family, this attack proceeds in two phases: preprocessing phase and online phase. In the preprocessing phase, the attacker tries to find a cube tester for a subset of key bits as well as sufficiently many corresponding plaintext descriptions (in cube variables and that subset of key bits). In the online phase, for each key guess, the attacker runs the cube tester by appropriate queries from the encryption oracle and checks if such collections of (P, C) pairs conform the cube tester. If so, the guessed key is treated as a correct key candidate to be rechecked then.

Preprocessing In this phase, inspiring from the distinguisher-based attacks on block ciphers, we position the cube tester at the *middle* of the cipher (e.g. exactly after round r_u) and *extend* it in two directions over the maximum possible upper and lower rounds. Such a position for the cube tester immediately implies that cube variables are defined among the intermediate state $S^{(r_u)}$ rather than the plaintext bits while the other bits of $S^{(r_u)}$ can be *static* variables (those which must be statically zero) or *neutral* variables (those which can take any value, zero or one, since the distinguisher bit, the bit whose non-randomness is supposed to be distinguished, is independent of them). At this moment, it is not specified yet what the role of each non-cube bit of $S^{(r_u)}$ is: static or neutral variable.

Both the upper and lower extensions demands guessing some key bits. We denote the subset of key bits (or equivalent bits) that should be guessed in the upper extension by $\mathcal{K}_u$ and that of the lower extension by $\mathcal{K}_l$. A schematic view of the attack is shown in Fig. 1.

In general, the lower and upper extensions of a distinguisher for a block cipher is a straightforward procedure. At least, in none of the distinguisher-based attacks, the distinguisher itself is affected by those extensions. But in our cube attack, the upper extension procedure is not so straightforward insofar as it determines the role of the non-cube variables in $S^{(r_u)}$, hence it affects directly on the cube tester behaviour. In other words, after determining the upper round of the distinguisher, r_u, and choosing the cube variables among $S^{(r_u)}$, we are not free enough to statically assign zero to the other variables (such a strategy is possible indeed but so costly). In fact, the upper extension procedure determines a subset of key bits, $\mathcal{K}_u$, which must be guessed in order to find a specification for the plaintext in X_I and $\mathcal{K}_u$, denoted by $P = p(X_I, \mathcal{K}_u)$. It is guaranteed

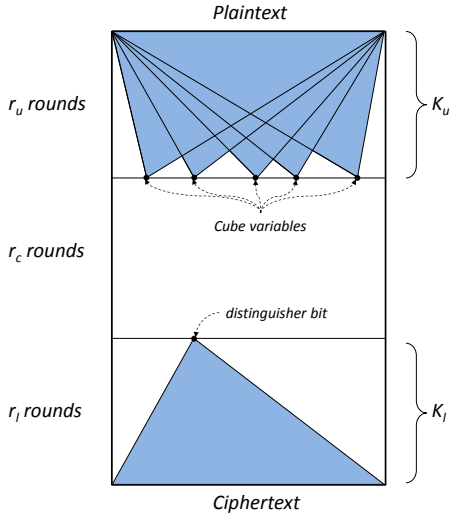

Fig. 1. An overview of the attack

that K_u has a minimum size and r_u-round encrypting of $p(X_I, \mathcal{K}_u)$ yields a fashion for $S^{(r_u)}$ in which exactly the bits indexed by I contain a cube variable. Additionally, this procedure clearly determines the status of non-cube bits of $S^{(r_u)}$: static or neutral.

Having determined the status of all bits of $S^{(r_u)}$ whether cube, static or neutral, we can find the maximum length cube tester for this $S^{(r_u)}$ as the input of cube tester. Suppose that the length of the cube tester is r_c. So, the distinguisher bit T, which is expected to show a non-random behavior, is one bit among the intermediate state $S^{(r_u+r_c)}$. In our analysis we have assumed only deterministic cube testers, but it can be any function with a easily detectable non-random property. Once r_c is known, we find the maximum possible rounds that the lower extension part covers. The limiting factor here is the number of key bits that should be guessed, i.e. $|\mathcal{K}_l|$, in order to compute the distinguisher bit from the ciphertext. This extension is not as complicated as the upper extension. But, in some cases, there might be some room for more efficient implementation.

Therefore, the total number of rounds attacked by this method would be $r = r_u + r_c + r_l$. An optimum partitioning is the one maximizing r. To get such a partitioning, Algorithm 1 of Appendix C is proposed. In this algorithm, the cube tester *slides* along a pre-defined range of $r_{u,min} \leq r_u \leq r_{u,max}$ for which it tests randomly chosen cube sets of dimension d at round r_u and returns the one, whose attack covers the maximum rounds $r = r_u + r_c + r_l$, satisfying the complexity constraints $Data < Data_{max}$ and $Time < Time_{max}$ where $Data_{max}$ and $Time_{max}$ are maximum allowable complexities given as inputs to this algorithm. For each r_u and cube set X_I of dimension d, this algorithm proceeds

in 3 subroutines: Upper-Extension-Subroutine, Cube-Tester-Subroutine and Lower-Extension-Subroutine which are explained in details in the following.

Upper-Extension-Subroutine

The input parameters for this subroutine are $\{r_u, I\}$ and it is supposed to return the following outputs.

- the subset of upper extension (equivalent) key bits, i.e. $\mathcal{K}_u$, which should be guessed.
- the plaintext description $p(X_I, \mathcal{K}_u)$ in such a way that its r_u-round partial encryption yields $S^{(r_u)}$ where

$$S_i^{(r_u)} = \begin{cases} x_i + f_i(\mathcal{K}) & i \in I \\ f_i(\mathcal{K}) & i \notin I \end{cases}, \quad i = 0, \ldots, m-1 \tag{4}$$

  where $S_i^{(r_u)}$ is the i^{th} bit of $S^{(r_u)}$, x_i is a cube variable and f_i is a function of key bits only.
- the set of neutral and static variables of $S^{(r_u)}$, namely U and Z, respectively. It means that $Z = \{0 \leq i < m | i \notin I, f_i(\mathcal{K}) = 0\}$ and $U = \{0 \leq i < m | i \notin I, f_i(\mathcal{K}) \neq 0\}$.

So, this subroutine returns $\{\mathcal{K}_u, p(X_I, \mathcal{K}_u), U, Z\}$ for a given $\{r_u, I\}$. The procedure of this subroutine is given in Algorithm 2 of Appendix C, where all the variables including cube variables, key bits and intermediate states have symbolic values.

To see how this subroutine works, first assume that the attacker is going to extend the upper part for one round ($r_u = 1$). At first, all the bits of $S^{(r_u)}$ are set to zero except those whose indexes are in I which are assigned symbolic cube variables $\{x_i | i \in I\}$. Then, the attacker partially decrypts $S^{(r_u)}$ for one round while setting $\mathcal{K} = 0$ to get the pure dependency of $S^{(r_u-1)}$ to cube variables. Then, she partially encrypt the resulted $S^{(r_u-1)}$ again, this time taking $\mathcal{K}$ into account, to find out how the subkey of round $r_u - 1$ affects on $S^{(r_u)}$. There would be no need to guess any bits of $\mathcal{K}$ as far as all the bits of $S^{(r_u)}$ can be written in a fashion consistent with (4).

But, once there is a state bit whose representation violates (4), e.g. those containing AND of a x_i with a key bit, she has to zero an appropriate internal state bit to avoid such an event. This zero forcing is equivalent to one bit key guess which will be stored in $\mathcal{K}_u$ list. Having discovered such key bits, the attacker repeats the one-round partial decryption of $S^{(r_u)}$, this time with a subkey with nonzero symbolic value $\mathcal{K}_u$. So, $S^{(r_u-1)}$ would be a function of X_I and $\mathcal{K}_u$ while $S^{(r_u)}$ is a function of X_I and $\mathcal{K}$ consistent with (4).

The procedure for $r_u > 1$ is similar. Suppose that the attacker has already finished step t, $0 \leq t < r_u$ of upper extension. By now, she has driven $S^{(r_u-t)}$ and a list $\mathcal{K}_u$ of key bits to be guessed, each of which corresponds to one internal state bit to be zero. $S^{(r_u-t)}$ is specified based on X_I and $\mathcal{K}_u$ which guarantees that all bits of $S^{(r_u)}$ do not violate the form given in (4). This necessitate also the following representation for the other internal states

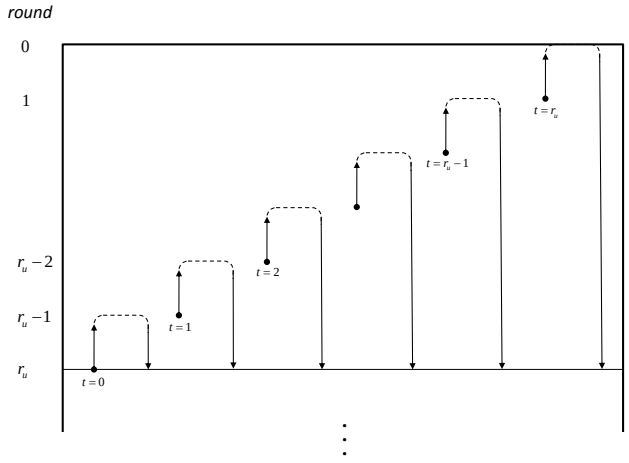

Fig. 2. schematic view of Upper-Extension-Subroutine

$$S_i^{(r_u-t+j)} = h_i^{(j)}(X_I, \mathcal{K}_u) + g_i^{(j)}(\mathcal{K}), \quad 1 \le j < t, \quad 0 \le i < m. \tag{5}$$

This condition means that the symbolic representation of any intermediate variable must be divisible into two separate parts: a function of X_I and $\mathcal{K}_u$ bits and a $\mathcal{K}$-only dependent function. In other word, those parts of the key mixing with the cube variables must be confined to $\mathcal{K}_u$.

Now, for step $t+1$ the attacker performs a one-round partial decryption of $S^{(r_u-t)}$ under a key in which all bits are zero except those specified as $\mathcal{K}_u$ and gets $S^{(r_u-t-1)}$. This is followed by a $(t+1)$-round partial encryption of $S^{(r_u-t-1)}$ under $\mathcal{K}$. Throughout the partial encryption, she should check if any bit of intermediate state violates condition (5) for all $S^{(r_u-t-j)}$, $1 \le j < t$ and condition (4) for $S^{(r_u)}$. Once such an event occurs, she breaks the partial encryption procedure, detects the internal state bit which should be forced to zero in order to avoid this event, finds the appropriate key bit to be guessed, and updates $\mathcal{K}_u$.

Having updated $\mathcal{K}_u$, the attacker repeats the one-round partial decryption of $S^{(r_u-t)}$ under the updated $\mathcal{K}_u$. Then, she performs the $(t+1)$-round partial encryption using $\mathcal{K}$, regularly checks conditions (4) and (5) and repeat the above scenario until these conditions are completely satisfied. In this way, at the end of step $t+1$, the attacker has derived an updated list $\mathcal{K}_u$ of key bits to be guessed as well as the description of $S^{(r_u-t-1)}$ based on X_I and $\mathcal{K}_u$ which guarantees conditions (4) and (5).

This subroutine terminates at $t = r_u$, when all the required outputs $\{\mathcal{K}_u, p(X_I, \mathcal{K}_u), U, Z\}$ are provided. A schematic view can be seen in Fig. 2. It should be noted that, for a given $\{r_u, I\}$, this attack requires N different plaintext descriptions, denoted by $p^{(i)}(X_I, \mathcal{K}_u), i = 1, 2, \ldots N$ all for the same $\mathcal{K}_u, U$ and Z. For

each guess of $\mathcal{K}_u$, the *batch* of plaintext corresponding to a plaintext description $p(X_I, \mathcal{K}_u)$ contains 2^d plaintexts which are generated when $p(X_I, \mathcal{K}_u)$ takes all 2^d values for X_I. As we will see in the case studies in Appendix A and B, given one $p(X_I, \mathcal{K}_u)$, it is an easy task to derive the other ones.

This subroutine solely determines the upper bound of data complexity for a given $\{r_u, I\}$ and a number of N tests. Since each of the N tests requires 2^d plaintexts for $2^{|\mathcal{K}_u|}$ key guesses, the data complexity would be bounded to $N \cdot 2^{d+|\mathcal{K}_u|}$. It is an upper bound since the batches corresponding to different $\mathcal{K}_u$ guesses and different tests may overlap. This is more probable when $d+|\mathcal{K}_u|$ gets greater.

After calling Upper-Extension-Subroutine for a given $\{r_u, I\}$ in Algorithm 1 of Appendix C, the (upper bound of) data complexity is computed and this algorithm will continue if the computed data complexity is less than D_{max}, otherwise it turns into the next $\{r_u, I\}$ pair. Note that in the case of full-codebook scenario, $Data_{max} = 2^m$ and the data complexity never exceeds $Data_{max}$ even if the numerical value for $N \cdot 2^{d+|\mathcal{K}_u|}$ does. So, this condition is omitted in this case. Furthermore there are some possibilities for shortening $\mathcal{K}_u$ list in the full-codebook attack, which will be discussed more in the case studies.

Cube-Tester-Subroutine

Once the attacker defined the status of all bits of $S^{(r_u)}$, i.e. U and Z for a given I, she should find the longest cube tester. To do so, for a given I, Z, U she runs the Cube-Tester-Subroutine proposed in Algorithm 3 of Appendix C. At first, based on the cipher structure, the potentially lowest algebraic degree bit in one round of the cipher is identified as the distinguisher bit T (i.e. the bit whose non-randomness is supposed to be distinguished).

This algorithm starts from $r_c = 1$, increases r_c in each step, and checks if a cube tester can be found for T at round $r_u + r_c$, while the input cube, static and neutral variables are set at $S^{(r_u)}$ according to the given I, Z and U sets, respectively. The tool used here is similar to the probabilistic linear test in [2] which is modified to tolerate a linear dependency to the neutral bits (which are functions of the key bits, in turn) as well as a linear dependency to the key bits of the cube part (i.e. subkeys of rounds $r_u + 1$ to $r_u + r_c$). So, this subroutine returns the maximum r_c, for which the distinguisher bit can be described as

$$\begin{aligned} T(r_c) &= L(\mathcal{K}) + \sum_{i \in U} b_i \cdot S_i^{(r_u)} \\ &= L(\mathcal{K}) + \sum_{i \in U} b_i \cdot f_i(\mathcal{K}) = F(\mathcal{K}) \end{aligned} \tag{6}$$

where, L is a linear function. If $F(\mathcal{K})$ is not a constant function, $\mathcal{K}_u$ should be updated as $\mathcal{K}_u = \mathcal{K}_u \cup F(\mathcal{K})$. Therefore, for a given $\{U, Z, I, r_u\}$, the Cube-Tester-Subroutine returns the maximum possible r_c, for which the distinguisher bit has a non-random property.

The reader should be noticed that we called the variables whose index are in U neutral variables, since they contain an unknown key dependent constant value.

Although the distinguisher should not naturally depend on a neutral variable by definition, our algorithm accepts a distinguisher as far as its dependency on neutral bits is at most a linear dependency.

Lower-Extension-Subroutine

Once the cube tester length is determined, the only thing remaining in the preprocessing phase is to trace the distinguisher bit at round $r_u + r_c$ towards the ciphertext to find a subset of (equivalent) key bits, $\mathcal{K}_l$, whose guess is sufficient for computation of the distinguisher bit from the ciphertext. This subroutine returns the maximum number of rounds r_l which can be covered by the lower extension part. The limiting factor here is the total time complexity of the attack which is directly influenced by $d, |\mathcal{K}_u|$ and $|\mathcal{K}_l|$ according to (7). The procedure of this subroutine is detailed in Algorithm 4 of Appendix C.

Having run the three subroutines for all candidate pairs of $\{r_u, I\}$, the pair which covers the maximum r, as well as all the relevant parameters $\{r, r_l, \mathcal{K}_u, \mathcal{K}_l, p^{(j)}(X_I, \mathcal{K}_u)|_{j=1,\ldots,N}, Time, Data\}$ is returned as the output of preprocessing phase.

Online Phase This phase of the attack is detailed in Algorithm 5 of Appendix C. By this algorithm, the whole secret key is retrieved as follows. For each guess for $\mathcal{K}_u$, the attacker computes N batch of plaintexts according to $p^{(j)}(X_I, \mathcal{K}_u), 1 \leq j \leq N$ and queries the encryption oracle to get the corresponding batch of ciphertexts. Then, she checks whether this specific guess for $\mathcal{K}_u \cup \mathcal{K}_l$ passes all the N tests. If so, this guess is regarded as a candidate for these subset of key bits and will be rechecked by some (P, C) pairs while the remaining key bits have been guessed. So, the attack complexities are:

$$\begin{aligned} \text{Data Complexity} &\leq N \cdot 2^{d+|\mathcal{K}_u|} \\ \text{Time Complexity} &= Data + \frac{r_l}{r} N \cdot 2^{d+|\mathcal{K}_u \cup \mathcal{K}_l|} + 2^{|\mathcal{K}|-N} + 2^{|\mathcal{K}|-N-m} + .. \quad (7) \\ \text{Memory Complexity} &= N \cdot 2^d \end{aligned}$$

According to Algorithm 5, this memory is required for storing all N batch of plaintext/ciphertext pairs.

3.2 Choosing Parameters

Selecting appropriate values for input parameters $r_{u,min}, r_{u,max}$, and d plays an important role in the efficiency of the resulted attack. For a fixed d, both the upper and lower extension parts make a few key bits active in the first rounds of the extensions while it approach to 2 key bits per round in the higher rounds which we call the *saturated* domain for the upper/lower extension parts. So, the best strategy is to make the best use of the both unsaturated domains. The optimum r_u locates the cube distinguisher in such a way that both of the unsaturated parts are included in the rounds covered by the attack. A too long

upper or lower extension part makes the opposite side too short which does not allow an efficient use of the unsaturated part.

The other important choice is the cube dimension d. Although the larger d makes r_c potentially larger, it also makes the second term of the attack time complexity (7) larger. In addition, a larger d increases the chance of guessing key bits in the upper extension part, i.e. increasing $|\mathcal{K}_u|$, which in turn increases the time complexity. Furthermore, the reader should be noticed that data complexity depends exponentially on d and $|\mathcal{K}_u|$, but it is independent of $|\mathcal{K}_l|$. On the other hand, according to the above discussion, for very large values for d, e.g. near the block size values, we expect that the upper extension reach very soon to the saturated domain, hence r_u should be too short. But in this special case we can make use of a method called *reseting cube symbols* which will be introduce later. By applying this method to the Upper-Extension-Subroutine, a great reduction in the size of $\mathcal{K}_u$ would be possible. That is why we found the two versions of small dimension and large dimension cubes more efficient than the middle values.

The number of tests N, on the one hand reduces the last term of time complexity, but on the other hand it increases the second term.

4 Cryptanalysis of KATAN32

4.1 Specification of KATAN32

KATAN32 is the smallest member of KATAN family of block ciphers with a 32-bit and 80-bit block and key sizes. This cpher is a lightweight block cipher with a simple algebraic round function and a large number of rounds. In KATAN32, the plaintext is loaded into two registers $L1$, and $L2$ with 13 and 19 bits length, respectively. The least significant bit of each register, nunbered by 0, is the rightmost one, and the LSB of plaintext is loaded into the LSB of $L2$ while its MSB is loaded into the MSB of $L1$. In each round, $L1$ and $L2$ are shifted to the left for one bit, where the newly computed bits, according to the following equations, are loaded into the LSB of $L2$ and $L1$, respectively.

$$\begin{aligned} f_a(L1) &= L1[12] + L1[7] + (L1[8] \cdot L1[5]) + (L1[3] \cdot IR) + a \\ f_b(L2) &= L2[18] + L2[7] + (L2[12] \cdot L2[10]) + (L2[8] \cdot L2[3]) + b \end{aligned} \tag{8}$$

where IR is a round-dependent constant, and a, b are two subkey bits generated by the key schedule of the cipher. After 254 rounds, the contents of the registers are then exported as the ciphertext.

Key schedule. KATAN32 has a linear key schedule based on a LFSR structure which generates $2 \times 254 = 508$ subkey bits according to the following rule:

$$rk_i = \begin{cases} k_i & 0 \le i < 80 \\ rk_{i-80} + rk_{i-61} + rk_{i-50} + rk_{i-13} & 80 \le i < 508 \end{cases} \tag{9}$$

For round $i = 1, \ldots, 254$, a_i is defined to be rk_{2i-2}, whereas b_i is rk_{2i-1}.

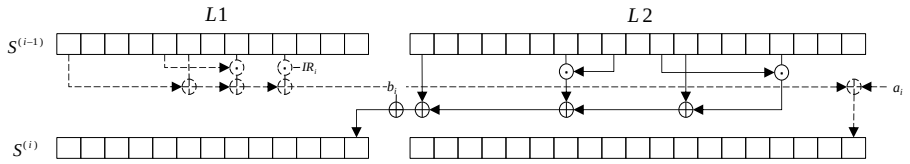

Fig. 3. One round of KATAN32

4.2 Cube Attack on KATAN32

For $Time_{max} = 2^{79}$, we set $N = 2$ and select the MSB of register $L2$ as the distinguisher bit which had the lowest degree compared to the other state bits at the same round. We perform our attack in two scenarios: non-full-codebook and full-codebook attacks.

In non-full-codebook attack we once set $Data_{max} = 2^{20}$ and searched through the small dimension cubes i.e. $d < 8$. The best results belong to a cube of dimension 5 which could break up to 118 rounds of KATAN32. Then, we set $Data_{max} = 2^{31}$ where we searched through the high dimension cubes ($d = 30, 31$) where a cube of $d = 31$, breaking 138 rounds, was the most efficient one.

In full-codebook scenario, a cube of $d = 31$ could reach up to 155 rounds of KATAN32. The details of the best found cubes are reported in Table 3. As an example, all the parameters of attack Case I are reported in details in Appendix A.

Improvements in the lower extension. The relations given in (7) are actually the attack complexities in the worst case. Depending on the target cipher, there might be some possibilities to slightly modify the online phase of the attack given in Algorithm 5 to improve the time complexity or even extend the number of rounds attacked. In the case of KATAN32, in the lower extension part for the partial decryption of the ciphertext, subkeys of the last four rounds linearly contribute to the state of round $r - 4$. So, instead of guessing the whole $\mathcal{K}_l$, and decrypting r_l rounds at once for each guess, we can decrypt four rounds supposing $K = 0$, store the state in a memory and then guess and add that part of $\mathcal{K}_l$ corresponding to these four rounds. This scenario can be used repeatedly for all consecutive four rounds of the cipher in lower part untill all the r_l rounds are covered. Moreover, for determining the value of the distinguisher bit from the ciphertext, there is no need to decrypt all rounds of the lower extension part. Using these two techniques we will be able to reduce the second term of time complexity by a factor of $\frac{1}{r_l}$. But, it increases memory complexity less than triple to save decrypted bits of every ciphertext for each guessed part of $\mathcal{K}_l$.

Improvements in the upper extension. For the case of high dimension cubes, e.g. $d = 30, 31$, we can get a significant improvement in the upper extension using a technique which we call *resetting cube symbols*. Consider Case III of Table 3, the internal state at round $r_u = 37$ at step $t = 0$ of the upper extension

Table 3. Details of the best dynamic cube attacks on KATAN32

Case	d	N	I	r_u	r_c	r_l	$\lvert\mathcal{K}_u\rvert$	$\lvert\mathcal{K}_l\rvert$	r	Time	Data
I	5	2	7, 13, 14, 17, 18	22	50	46	14	63	118	$2^{78.3}$	2^{19}
II	30	2	0, 1, ..., 16, 19, ..., 31	23	74	41	0	53	138	$2^{78.5}$	2^{31}
III	31	2	0, 1, ..., 17, 19, ..., 31	37	83	35	10	41	155	$2^{78.3}$	2^{32}

subroutine is $S^{(37)} = (L1^{(37)}, L2^{(37)})$ where:

$$L1^{(37)} = \{x_{31}, ..., x_{19}\} \quad L2^{(37)} = \{0, x_{17}, ..., x_0\}$$

After running the first step of the Upper-Extension-Subroutine, the intermediate states at rounds 37 and 36 are as follows:

$$\begin{aligned} L1^{(36)} &= \{x_0 + x_{27} + x_{28}x_{25} + x_{23} \cdot IR_{36}, x_{31}, ..., x_{20}\} \\ L2^{(36)} &= \{x_{19} + x_8 + x_4 \cdot x_9 + x_{11} \cdot x_{13}, 0, x_{17}, ..., x_1\} \\ L1^{(37)} &= \{x_{31}, ..., x_{19} + b_{36}\} \\ L2^{(37)} &= \{0, x_{17}, ..., x_0 + a_{36}\} \end{aligned}$$

which shows that some new dynamic variables are introduced in the MSBs of the two registers at round 36. Now, consider the following state description $S'^{(36)}$:

$$L1'^{(36)} = \{y_{31}, ..., y_{19}\} \quad L2'^{(36)} = \{y_{18}, 0, y_{16}, ..., y_0\}$$

A set of intermediate states $S'^{(36)}$ in which y_is take all 2^{31} possible values is equivalent to a set of intermediate states $S^{(37)}$ in which x_is take all 2^{31} possible values. Since $S'^{(36)}$ guaranties that the static variable of $S^{(37)}$ is zero. In fact for high dimension cubes, it is more efficient to trace the neutral/static variables to the upward, rather than the cube variables as opposed to what we did in Upper-Extension-Subroutine for small dimension cubes. This method for resetting cube symbols not only simplifies the upper part computations, but also it causes a great reduction in $|\mathcal{K}_u|$, when the number of upper rounds increases.

5 Cryptanalysis of SIMON32/64

5.1 Specification of SIMON32/64

SIMON32/64 is a 32-round Feistel block cipher whose round function composed of AND, XOR and rotations. The internal state at the input and output of round i are denoted by $S^{(i-1)} = (L^{(i-1)}, R^{(i-1)})$ and $S^{(i)} = (L^{(i)}, R^{(i)})$ respectively, where

$$\begin{aligned} L^{(i)} &= R^{(i-1)} + F(L^{(i-1)}) + k^{(i)} \\ R^{(i)} &= L^{(i-1)} \end{aligned}$$

$$F(L^{(i-1)}) = (L^{(i-1)} \lll 1) \cdot (L^{(i-1)} \lll 8) + (L^{(i-1)} \lll 2)$$

Table 4. Details of the best dynamic cube attacks on SIMON32/64

Case	d	N	I	r_u	r_c	r_l	$\lvert\mathcal{K}_u\rvert$	$\lvert\mathcal{K}_l\rvert$	r	Time	Data
I	5	8	$8, 14, 16, 22, 24$	3	8	6	10	45	17	2^{57}	2^{18}
II	31	6	All except 30 or 23 or 16	4	14	4	16	16	22	2^{59}	2^{32}

where $\lll$ denotes the left rotation operation and $k^{(i)}$ is the subkey of round i, $1 \leq\ i \leq 32$.

Key schedule. SIMON has a linear key schedule as follows. Assume that $K = K_3, K_2, K_1, K_0$ is the representation of secret key in four 16-bit words K_i. The subkey of SIMON32/64 at round i is generated as follows.

$$k^{(i)} = \begin{cases} K_{i+1} & 0 \leq i \leq 3 \\ k^{(i-4)} + Y + (Y \ggg 1) + c + (z_0)_i & 4 \leq i \leq 32 \end{cases}$$

where $Y = k^{(i-3)} + (k^{(i-1)} \ggg 3)$, c is a constant and z_0 is a binary stream whose precise value is given in [15].

5.2 Cube Attack on SIMON32/64

Similar to KATAN, we consider the non-full-codebook and full-codebook scenarios, where in the former the cubes with small dimensions are searched and for the latter, cubes with dimension 31 are examined. In each round, the right half variables has the same algebraic degree which is less than the left half. So, we select bit 15 as the distinguisher bit.

For the non-full-codebook scenario, we set $D_{max} = 2^{20}$ and $N = 11$. Amongst all the the small dimension cubes which were examined, the best result belongs to a cube of dimension 5 which could break 17 rounds of SIMON32/64.

In the full-codebook scenario, we could attack 22-round SIMON32/64 by a number of $N = 6$ tests. These six tests are actually three pairs of distinguishers, each pair has a neutral variable (in position 16 or 23 or 30) and 31 cube variables. The tests we used are the same as those reported in [27] for the integral attack on SIMON. Each pair of distinguishers demands guessing a set of 16 key bits for $\mathcal{K}_u$. Despite the previous attacks, these sets are not completely the same, though two of them overlap. In more details, $\mathcal{K}_u$ bits corresponding to distinguisher pairs with neutral bit in 16 and 23 have 11 bits in common. Hence, $|\mathcal{K}_u|$ would be 37 bits which is a huge size at first glance. But, it can be treated and checked independently for each pair of distinguishers. After checking the subsets of key bits $\mathcal{K}_u$ for each pair of tests, we save the satisfying candidates in a table which is indexed by $\mathcal{K}_l$ and 11 common bits of $\mathcal{K}_u$. In these tables, in average, for each index there is a number of $2^{5-2} = 2^3$ elements that shows the 5 uncommon bits of $\mathcal{K}_u$. After checking for these 3 pair of tests, we will have 2^{37} candidate for 43 bits of $\mathcal{K}_l \cup \mathcal{K}_{u1} \cup \mathcal{K}_{u2} \cup \mathcal{K}_{u3}$. The cost for this improvement is a memory for three tables that need about $2^{30} \times 3 \times 5$ bits.

The parameters related to our attacks are reported in Table 4. Furthermore, as an example all the parameters of attack Case I are given in details in Appendix B.

Improvements. In the lower extension part of both cases we enjoyed the same method used in KATAN attack for saving time in guessing $\mathcal{K}_l$ bits (reduction factor is $\frac{1}{8}\frac{1}{r_l}$). Furthermore, we enjoyed the resetting cube symbols method for Case II of SIMON attack, as well.

6 Conclusions and Discussions

In this paper we proposed a new framework for dynamic cube attack on block ciphers. The algorithm proposed for this attack can be made fully automated, in spite of its counterpart in stream ciphers. We analysed the efficiency of attack on two block ciphers KATAN and SIMON and the results show that this newcomer method for cryptanalysis of block cipher can compete with and often outperform from the mature and well-organized cryptanalysis methods in this domain.

Although our attack is similar to the dynamic cube attack on stream ciphers [4] in some properties, it is completely different in the tools, which are used. In more details, our attack shares the following properties with the dynamic cube attack on stream ciphers.

- Assigning the public variables some functions of cube variables and guessed keys (dynamic variables).
- Using a cube tester to check the validity of guessed key bits.

But, the main differentiations of our work with the dynamic cube attack on stream ciphers are as follows.

- The cipher is explicitly divided into three *distinct* parts: upper extension part, cube part, and lower extension part. Each part are treated in sequence and semi-independently. In dynamic cube attack on stream ciphers, the lower part is not present and the first two parts can not be separated.
- This partition enables the attacker to *fully automate* the attack and get rid of the manual complex process of dynamic cube attack for deriving the description of dynamic variables.
- The cube variables are defined among the intermediate state bits rather than the public variables.
- The keys guessed in the attack are not only those involved in the dynamic variables. They are also involved in the lower rounds of the cipher.

In fact, the main feature of block ciphers from which most of these differences arise, especially the possibility to automate the attack, is the availability of an key-only-dependent function in any round of the cipher, i.e. the key schedule. Key schedule enables the attacker to perform partial encryption/decryption anywhere within the cipher rounds conditioned that she has guessed the required subkeys. Such a facility is not provided in stream ciphers, where the secret key along with the IV are loaded into the stream cipher state in the first clock and after that

there would be no pure access to the secret key, either in the initialization or key stream generation phases.

There might be a debate about the similarity of this attack and square attack on block ciphers. Although the cube attack and square attack may have a similar essence, there are some difference as well. The similarities are as follows:

- From the theoretical point of view, both the distinguishers basically detect a nonrandom property (usually equality to zero) for the summation of the outputs of the all possible values of a subset of input bits.
- The distinguisher is positioned at the middle of the cipher. Then, a key recovery attack is constructed based on that by extending it from two directions.

But the main differences between the two techniques are as follows:

- From the practical point of view, the integral attack is efficiently applied to Sboxed-based structures and, despite cube attack, it does not derive useful distinguishers against block ciphers with non-bijective functions, bit-oriented structures, and low degree functions (such as KATAN and SIMON) [29].
- The main property that makes our attack different is the procedure of the upper-extension-subroutine, which affects the distinguisher parameters and turns our attack from a classic cube attack (integral attack) into a dynamic one.

References

1. M. Vielhaber, "Breaking ONE.FIVIUM by AIDA, an Algebraic IV Differential Attack", Cryptology ePrint Archive, Report 2007/413, 2007.
2. I. Dinur, and A. Shamir, "Cube Attacks on Tweakable Black Box Polynomials", EUROCRYPT'09, LNCS, vol. 5479, pp. 278-299, Springer, 2009.
3. J. Aumasson, I. Dinur, W. Meier, and A. Shamir, "Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium", FSE'09. LNCS, vol. 5665, pp. 1-22, Springer, 2009.
4. I. Dinur, and A. Shamir, "Breaking Grain-128 with Dynamic Cube Attacks", FSE'11, LNCS, vol. 6733, pp. 167187, Springer, 2011.
5. M. Hell, T. Johansson, A. Maximov, and W. Meier, "The Grain Family of Stream Ciphers", New Stream Cipher Designs - The eSTREAM Finalists, LNCS, vol. 4986, pp.179190, Springer, 2008.
6. C. De Canniere, and B. Preneel, "Trivium", New Stream Cipher Designs - The eSTREAM Finalists. LNCS, vol. 4986, pp. 244-266, Springer, 2008.
7. P. -A. Fouque and T. Vannet, "Improving Key Recovery to 784 and 799 rounds of Trivium using Optimized Cube Attacks", FSE'13, LNCS, vol. 8424, pp. 502-517, Springer, 2013.
8. A. Vardasbi, M. Salmasizadeh, and J. Mohajeri, "Superpoly Algebraic Normal Form Monomial Test on Trivium", IET Information Security, vol. 7, no. 3, pp. 230-238, 2013.
9. M. Rahimi, M. Barmshory, M. H. Mansouri, and M. R. Aref, "Dynamic Cube Attack on Grain-v1", accepted in IET Information Security, 2015.

10. G. V. Bard, N. T. Courtois, J. Nakahara, P. Sepehrdad, and B. Zhang, "Algebraic, Aida/Cube and Side Channel Analysis of Katan Family of Block Ciphers", INDOCRYPT'10, LNCS, vol. 6498, pp. 176196, Springer, 2010.
11. I. Dinur and A. Shamir, "Side Channel Cube Attacks on Block Ciphers", IACR Cryptology ePrint Archive, vol. 2009, Report 2009/127, 2009.
12. Z. Li, B. Zhang, J. Fan, and I. Verbauwhede, "A New Model for Error-Tolerant Side-Channel Cube Attacks", CHES'13, LNCS, vol. 8086, pp. 453-470, 2013.
13. L. Yang, M. Wang, and S. Qiao, "Side Channel Cube Attack on PRESENT", CANS'09, LNCS, vol. 5888, pp. 379-391, 2009.
14. S. F. Abdul-Latip, R. Reyhanitabar, W. Susilo, and J. Seberry, "Fault Analysis of the Katan Family of Block Ciphers", ISPEC'12, LNCS, vol. 7232, pp.319-336, Springer, 2012.
15. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The Simon and Speck Families of Lightweight Block Ciphers", IACR Cryptology ePrint Archive, vol. 2013, Report 2013/404, 2013.
16. C. D. Cannire, O. Dunkelman, and M. Knezevic, "KATAN and KTANTAN - a Family of Small and Efficient Hardware-oriented Block Ciphers", CHES'09, LNCS, vol. 5747, pp. 272-288, Springer, 2009.
17. S. Knellwolf, W. Meier, and M. Naya-Plasencia, "Conditional Differential Cryptanalysis of NLFSR-based Cryptosystems", ASIACRYPT'10, LNCS, vol. 6477, pp. 130-145, Springer, 2010.
18. M. R. Albrecht and G. Leander, "An all-in-one Approach to Differential Cryptanalysis for Small Block Ciphers, SAC'12, LNCS, vol. 7707, pp. 115, Springer, 2013.
19. T. Isobe and K. Shibutani, "Improved All-Subkeys Recovery Attacks on Fox, Katan and Shacal-2 Block Ciphers, FSE'14, LNCS, vol. 8540, pp. 104-126, Springer, 2015.
20. T. Fuhr and B. Minaud, "Match Box Meet-in-the-Middle Attack Against Katan", FSE'14, LNCS, vol. 8540, pp. 61-81, Springer, 2015.
21. B. Zhu and G. Gong, "Multidimensional Meet-in-the-Middle Attack and Its Applications to KATAN32/48/64", Cryptography and Communications, vol. 6, Issue 4, pp. 313-333 , Springer, 2014.
22. J. Alizadeh, H. A. Alkhzaimi, M. R. Aref, N. Bagheri, P. Gauravaram, A. Kumar, M. M. Lauridsen, and S. K. Sanadhya, "Cryptanalysis of Simon variants with Connections", RFIDsec'14, LNCS, vol. 8651 , pp. 90-107, Springer, 2014.
23. L. Song, L. Hu, B. Ma, D. Shi, "Match Box Meet-in-the-Middle Attacks on the SIMON Family of Block Ciphers", LightSec'14, LNCS, vol. 8898, pp. 140-151, Springer, 2015.
24. C. Boura, M. Naya-Plasencia, and V. Suder, "Scrutinizing and Improving Impossible Differential Attacks: Applications to Clefia, Camellia, Lblock and Simon", ASIACRYPT'14, LNCS, vol. 8873, pp. 179-199, Springer, 2014.
25. Farzaneh Abed, E. List, S. Lucks, and J. Wenzel, "Differential Cryptanalysis of Round-Reduced Simon and Speck", FSE'14, LNCS, vol. 8540, pp. 525-545, Springer, 2015.
26. A. R. Alex Biryukov and V. Velichkov, "Differential Analysis of Block Ciphers Simon and Speck", FSE'14, LNCS, vol. 8540, pp. 546-570, Springer, 2015.
27. Q. Wang, Z. Liu, K. Varc, Y. Sasaki, V. Rijmen, and Y. Todo, "Cryptanalysis of Reduced-round SIMON32 and SIMON48", INDOCRYPT'14, LNCS, vol. 8885, pp. 143-160, Springer, 2014.
28. I. Dinur, P. Morawiecki, J. Pieprzyk, M. Srebrny, M. Straus, "Cube Attacks and Cube-Attack-Like Cryptanalysis on the Round-Reduced Keccak Sponge Function", EUROCRYPT'15, LNCS, vol. 9056, pp 733-761, Springer, 2015.

29. Y. Todo, "Structural Evaluation by Generalized Integral Property", EUROCRYPT'15, LNCS, vol. 9056, pp 287-314, Springer, 2015.

Appendix A. Details of attack case I on KATAN32

In this appendix the details of attack case I on KATAN32 are reported. The set of upper extension key is $\mathcal{K}_u = \{u_i | 1 \le i \le 14\}$ where

$$u_1 = b_1,\ \ u_2 = a_2,\ \ u_3 = a_3,\ \ u_4 = a_4,\ \ u_5 = b_5,\ \ u_6 = a_{10},\ \ u_7 = a_{13},$$
$$u_8 = b_2 + a_6,\ \ u_9 = b_3 + a_7,\ \ u_{10} = b_3 + b_7 + a_{11},\ \ u_{11} = b_6 + b_{11} + a_{14},$$
$$u_{12} = b_4 \cdot b_7,\ \ u_{13} = b_2 + b_4 \cdot (b_6 + b_{11} + a_{14}),\ \ u_{14} = a_9 + b_3 \cdot (a_{13} + b_4 \cdot b_7)$$

The first descryption for plaintext is $p^{(1)}(X_I, \mathcal{K}_u) = S^{(0)} = (L1^{(0)}, L2^{(0)})$ where

$$\begin{aligned}
L1^{(0)} = \{\ & u_6 + u_{10} + u_{13} + u_8 + (\mathbf{x_3} + u_7) \cdot (\mathbf{x_2} + \mathbf{x_5} + u_{11} + u_6) + (u_7 + u_{12}) \cdot \\
& (\mathbf{x_5} + u_6 + u_{13}),\ \ u_2 + u_9 + u_{10} + u_{10} \cdot u_{11} + (\mathbf{x_3} + u_{11} + u_{14}) \cdot \\
& (\mathbf{x_5} + u_8 + u_{10} + (\mathbf{x_5} + u_6 + u_{13}) \cdot (u_7 + u_{12})),\ \ \mathbf{x_3} + u_3 + u_7 + \\
& (u_9 + u_{10} \cdot u_{11}) \cdot (\mathbf{x_5} + u_8),\ \ \mathbf{x_1} + \mathbf{x_4} + u_4 + u_7 + u_{11} + u_{12} + u_{14} + \\
& (\mathbf{x_3} + u_7) \cdot u_{10},\ \ \mathbf{x_2} + \mathbf{x_5} + u_6 + u_{11},\ \ \mathbf{x_5} + u_8 + u_{10} + (u_7 + u_{12}) \cdot \\
& (\mathbf{x_5} + u_6 + u_{13}),\ \ u_9 + u_{11} \cdot u_{10},\ \ \mathbf{x_3} + u_7,\ \ \mathbf{x_4} + u_{11} + u_{14}, \\
& \mathbf{x_5} + u_6 + u_{13},\ \ u_{10},\ \ 0,\ \ u_7 + u_{12}\} \\
L2^{(0)} = \{\ & \mathbf{x_1} + u_1 + u_{11},\ \ \mathbf{x_5} + \mathbf{x_2},\ \ 0,\ \ 0,\ \ \mathbf{x_3} + u_5,\ \ \mathbf{x_4},\ \ 0,\ \ 0,\ \ 0,\ \ 0,\ \ 0, \\
& \mathbf{x_1},\ \ \mathbf{x_2} + \mathbf{x_1} \cdot \mathbf{x_4},\ \ 0,\ \ 0,\ \ \mathbf{x_3},\ \ \mathbf{x_4},\ \ 0,\ \ 0\}
\end{aligned} \tag{10}$$

which yields the following descrption for $S^{(22)} = (L1^{(22)}, L2^{(22)})$ which is completely consistent with (4).

$$\begin{aligned}
L1^{(22)} = \{\ & b_{10},\ b_{11},\ b_{12},\ a_5 + b_{13} + b_2 + b_4 \cdot u_{11},\ b_{14},\ b_{15},\ a_8 + b_{16} + b_7 \cdot b_4, \\
& b_{17},\ b_{18},\ b_{19},\ a_1 + b_6 \cdot b_3 + b_4 + b_8 + a_{12} + b_{20}, \\
& b_{21} + (b_6 \cdot b_3 + b_4 + b_8 + a_{12}) \cdot (b_4 + b_{11} \cdot b_8 + a_1 + b_9 + a_{17}),\ b_{22}\} \\
L2^{(22)} = \{\ & \mathbf{x_1}, \mathbf{x_2} + a_5 + b_2 + b_4 \cdot (a_{14} + b_{10} + b_6),\ 0,\ 0,\ \mathbf{x_3} + a_8 + b_7 \cdot b_4, \\
& \mathbf{x_4},\ 0,\ 0,\ b_6 \cdot b_3 + b_4 + b_8 + a_{12},\ 0,\ 0, \\
& \mathbf{x_5} + b_2 + b_6 \cdot (a_1 + b_9) + b_7 + a_{15},\ b_3 + b_{10} \cdot b_7 + b_8 + b_{12} + a_{16}, \\
& b_4 + b_{11} \cdot b_8 + a_1 + b_9 + a_{17}, b_{12} \cdot (a_1 + b_9) + b_{10} + b_{14} + a_{18}, \\
& b_6 + b_{10} \cdot (a_5 + b_{13} + b_2 + b_4 \cdot (a_{14} + b_{10} + b_6)) + b_{11} + a_{19}, \\
& b_7 + b_{14} \cdot b_{11} + b_{12} + a_8 + b_{16} + b_7 \cdot b_4 + a_{20}, \\
& b_8 + b_{15} \cdot b_{12} + a_5 + b_{13} + b_2 + b_4 \cdot (a_{14} + b_{10} + b_6) + b_{17} + a_{21}, \\
& b_9 + a_1 + (a_8 + b_{16} + b_7 \cdot b_4) \cdot (a_5 + b_{13} + b_2 + b_4 \cdot (a_{14} + b_{10} + b_6)) + \\
& b_{14} + b_{18} + a_{21}\}
\end{aligned}$$

The set of neutral and static variables are as follows.

$$U = \{0, 1, 2, 3, 4, 5, 6, 10, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31\}$$
$$Z = \{8, 9, 11, 12, 15, 16\}]$$

To get the second plaintext description, let $p^{(2)}(X_I, \mathcal{K}_u) = p^{(1)}(X_I, \mathcal{K}_u) + \Delta_2$ where Δ_2 and its corresponding difference at round 22, $\Delta_2^{(22)}$, are as follows.

$$\begin{aligned}\Delta_2 &= \{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0\}\\ \Delta_2^{(22)} &= \{0,0,0,0,0,0,0,0,0,0,0,b_6 \cdot b_3 + b_4 + b_8 + a_{12} + b_4 + b_{11} \cdot b_8 + a_1 + b_9 +\\ &\quad a_{17} + 1, 0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,b_{11},0,0,0,1,0\}\end{aligned}$$

which shows that all the nonzero differences are constant and confined to neutral bits. Finally, the set of key bits to be guessed in the lower extension part are as follows.

$$\mathcal{K}_l = \{a_i | i \in \{90, 93, ..., 118\}\} \cup \{b_i | i \in \{81, 83, 85, 87, ..., 118\}\} \cup \{b_{79} + a_{92}\}$$

Appendix B. Details of attack case I on SIMON32/64

In this appendix the details of attack case I on SIMON32/64 are reported. The set of upper extension key is $\mathcal{K}_u = \{u_i | 1 \leq i \leq 10\}$ where

$$\begin{aligned}&u_1 = c_{15}, \quad u_2 = c_1, \quad u_3 = c_5, \quad u_4 = b_{15}, \quad u_5 = b_{13},\\ &u_6 = b_9, \quad u_7 = b_7, \quad u_8 = b_3, \quad u_9 = b_1, \quad u_{10} = b_5 + c_7,\end{aligned}$$

The first descryption for plaintext is $p^{(1)}(X_I, \mathcal{K}_u) = S^{(0)} = (L^{(0)}, R^{(0)})$ where

$$\begin{aligned}L^{(0)} &= \{u_1,\ \mathbf{x_2},\ 0,\ \mathbf{x_1},\ 0,\ \mathbf{x_5},\ 0,\ \mathbf{x_1} + \mathbf{x_4},\ u_{10},\ 0,\ u_3,\ 0,\ 0,\ \mathbf{x_2} + \mathbf{x_3},\ u_2,\ 0\}\\ R^{(0)} &= \{\mathbf{x_2} \cdot u_{10} + u_4,\ \mathbf{x_1},\ \mathbf{x_1} \cdot u_3 + u_6,\ \mathbf{x_5},\ 0,\ \mathbf{x_4}, u_2 \cdot (\mathbf{x_1} + \mathbf{x_4}) + u_{10} + u_6,\\ &\quad \mathbf{x_5},\ u_3 + u_7,\ \mathbf{x_4} + \mathbf{x_2} + u_3,\ 0,\ \mathbf{x_2} + \mathbf{x_3},\ u_2 + u_8,\ \mathbf{x_5} \cdot u_2,\ u_1 + u_9,\\ &\quad \mathbf{x_2} + u_1 \cdot (\mathbf{x_1} + \mathbf{x_4})\}\end{aligned}$$

which yields the following descrption for $S^{(3)} = (L^{(3)}, R^{(3)})$ which is completely consistent with (4).

$$
\begin{aligned}
L^{(3)} = \{&k_5^{(1)} \cdot k_{12}^{(1)} + k_{11}^{(1)} + k_{13}^{(2)} + k_{15}^{(3)},\\
&(k_5^{(1)} \cdot k_{14}^{(1)} + k_4^{(1)} + k_6^{(2)}) \cdot (k_5^{(1)} \cdot k_{12}^{(1)} + k_{11}^{(1)} + k_{13}^{(2)}) + \ k_{10}^{(1)} + k_4^{(1)} \cdot k_{11}^{(1)} +\\
&k_{12}^{(2)} + k_{14}^{(1)} + k_{14}^{(3)} k_{11}^{(2)} + k_{13}^{(3)},\ k_{11}^{(2)} \cdot (k_2^{(1)} + k_4^{(2)}) + k_8^{(1)} + k_{10}^{(2)} + k_{12}^{(1)} + k_{12}^{(3)},\\
&(k_8^{(1)} + k_{10}^{(2)}) \cdot (k_2^{(1)} \cdot k_{11}^{(1)} + k_3^{(2)}) + k_9^{(2)} + k_{11}^{(1)} + k_{11}^{(3)},\\
&k_6^{(1)} + k_{10}^{(1)} + k_8^{(2)} + k_9^{(2)} \cdot (k_0^{(1)} + k_2^{(2)}) + k_{10}^{(3)},\ \ k_9^{(3)},\\
&\mathbf{x_5} + k_5^{(1)} \cdot k_{14}^{(1)} + k_4^{(1)} + k_6^{(2)} + k_8^{(1)} + k_8^{(3)},\ \ k_7^{(3)},\ \ \mathbf{x_4} + k_2^{(1)} + k_4^{(2)} + k_6^{(1)} + k_6^{(3)},\\
&(k_2^{(1)} + k_4^{(2)}) \cdot (k_5^{(1)} \cdot k_{12}^{(1)} + k_{11}^{(1)} + k_{13}^{(2)}) + k_2^{(1)} \cdot k_{11}^{(1)} + k_3^{(2)} + k_5^{(1)} + k_5^{(3)},\\
&(k_2^{(1)} \cdot k_{11}^{(1)} + k_3^{(2)}) \cdot (k_{10}^{(1)} + k_4^{(1)} \cdot k_{11}^{(1)} + k_{12}^{(2)}) + k_0^{(1)} + k_2^{(2)} + k_4^{(1)} + k_4^{(3)},\\
&k_{11}^{(2)} \cdot (k_0^{(1)} + k_2^{(2)}) + k_3^{(3)},\ \ k_{14}^{(1)} + k_0^{(2)} + k_2^{(1)} + k_2^{(3)},\\
&k_9^{(2)} \cdot (k_{14}^{(1)} + k_0^{(2)}) + k_1^{(3)},\\
&\mathbf{x_3} + k_0^{(1)} + k_{12}^{(1)} + k_{14}^{(2)} + k_0^{(3)}\}\\
R^{(3)} = \{&0,\ \ \mathbf{x_2} + k_{12}^{(1)} + k_{14}^{(2)},\ \ k_5^{(1)} \cdot k_{12}^{(1)} + k_{11}^{(1)} + k_{13}^{(2)},\ \ k_{10}^{(1)} + k_4^{(1)} \cdot k_{11}^{(1)} + k_{12}^{(2)},\\
&k_{11}^{(2)}, k_8^{(1)} + k_{10}^{(2)}, k_9^{(2)},\ \ \mathbf{x_1} + k_6^{(1)} + k_8^{(2)},\ \ 0,\ \ k_5^{(1)} \cdot k_1^{(1)} 4 + k_4^{(1)} + k_6^{(2)},\ \ 0,\\
&k_2^{(1)} + k^{(2)},\ \ , k_2^{(1)} \cdot k_{11}^{(1)} + k_3^{(2)},\ \ k_0^{(1)} + k_2^{(2)},\ \ 0,\ \ k_{14}^{(1)} + k_0^{(2)}\}
\end{aligned}
$$

The set of neutral and static variables are identified as follows.

$$
\begin{aligned}
Z &= \{1, 5, 7, 15\}\\
U &= \{0, 2, 3, 4, 6, 9, 10, 11, 12, 13, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31\}
\end{aligned}
$$

To get the remaining 15 plaintext descriptions, let $p^{(j)}(X_I, \mathcal{K}_u) = p^{(1)}(X_I, \mathcal{K}_u) + \Delta_j$, $j = 2, \ldots, 16$. Δ_j could be any difference in the right half of $p^{(1)}(X_I, \mathcal{K}_u)$ since it would result in a difference at round 3, $\Delta_j^{(3)}$, with nonzero constant values confined to U and I indexes. For example see the following pair of differences:

$$
\begin{aligned}
\Delta_j &= \{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1\}\\
\Delta_j^{(3)} &= \{0,0,0,0,0,0,0,0,0,0,0,1,k_{11}^{(2)},0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0\}
\end{aligned}
$$

Finally, the set of key bits to be guessed in the lower extension part are as follows.

$$
\begin{aligned}
&\mathcal{K}_l = \{k_i^{(13)} | i \in I_{13}\} \cup \{k_i^{(14)} | i \in I_{14}\} \cup \{k_i^{(15)} | i \in I_{15}\} \cup \{k_i^{(16)} | i \in I_{16}\} \cup \{k_i^{(17)} | i \in I_{17}\}\\
&I_{13} = \{7, 14\}, \qquad I_{14} = \{5, 6, 12, 13, 15\} \qquad I_{15} = \{3, 4, 5, 7, 10, 11, 12, 13, 14\}\\
&I_{16} = \{1, \ldots, 6, 8, \ldots, 13, 15\}, \qquad I_{17} = \{0, \ldots, 15\}
\end{aligned}
$$

Appendix C. Attack algorithms

Algorithm 1 Preprocessing Phase

Input: $\{r_{u,min}, r_{u,max}, d, N, Time_{max}, Data_{max}\}$.
Output: $\{r, r_l, \mathcal{K}_u, \mathcal{K}_l, p^{(j)}(X_I, \mathcal{K}_u), 1 \leq j \leq N, Time, Data\}$.
$r = 0$; $Out' = \{\}$;
for $r_u = r_{u,min}$ **to** $r_{u,max}$ **do**
  for randomly chosen I of size d **do**
    $(\mathcal{K}_u, p(X_I, \mathcal{K}_u), U, Z) =$ Upper-Extension-Subroutine(r_u, I)
    $Data = N \cdot 2^{d+|\mathcal{K}_u|}$;
    if $Data \leq Data_{max}$ **then**
      $r_c =$ Cube-Tester-Subroutine(I, U, r_u);
      $(r_l, \mathcal{K}_l, Time)$=Lower-Extension-Subroutine$(r_c + r_u, d, \mathcal{K}_u, Time_{max}, N)$;
      if $r_u + r_c + r_l > r$ **then**
        $r = r_u + r_c + r_l$;
        $Out' \leftarrow \{r, r_l, \mathcal{K}_u, \mathcal{K}_l, p(X_I, \mathcal{K}_u), Time, Data\}$;
      end if
    end if
  end for
end for
$p^{(1)}(X_I, \mathcal{K}_u) = p(X_I, \mathcal{K}_u)$;
for $j = 2$ **to** N **do**
  $p^{(j)}(X_I, \mathcal{K}_u) = p^{(1)}(X_I, \mathcal{K}_u) + \Delta_j$; ▷ for some appropriate Δ_j chosen by the attacker.
end for
$Out \leftarrow \{r, r_l, \mathcal{K}_u, \mathcal{K}_l, p^{(j)}(X_I, \mathcal{K}_u), 1 \leq j \leq N, Time, Data\}$; **return** Out;

Algorithm 2 Upper-Extension-Subroutine

Input: $\{r_u, I\}$.
Output: $\{\mathcal{K}_u, p(X_I, \mathcal{K}_u), U, Z\}$.
$X_I = \{x_i | i \in I\}$; ▷ x_i is symbolic.
$\mathcal{K} = \{k_{n-1}, \ldots, k_0\}$; ▷ k_i is symbolic.
$S^{(r_u)} = 0$, $S_I^{(r_u)} = X_I$, $\mathcal{K}_u = \{\}$;
for $t = 0$ **to** $r_u - 1$ **do**
  $\tilde{S}^{(r_u - t - 1)} = dec(S^{(r_u - t)}, \mathcal{K}_u)$;
  for $j = 0$ **to** t **do**
    $\tilde{S}^{(r_u - t + j)} = enc(\tilde{S}^{(r_u - t + j - 1)}, \mathcal{K})$;
    if conditions (4) and (5) are not satisfied due to the key bit k_f **then**
      update $\mathcal{K}_u$ by k_f;
      go to 7;
    end if
  end for
  $S^{(r_u - t - 1)} = \tilde{S}^{(r_u - t - 1)}$;
end for
$p(X_I, \mathcal{K}_u) = S^{(0)}$;
$S^{(r_u)} = \tilde{S}^{(r_u)}$;
Identify sets U and Z from $S^{(r_u)}$;
return $\{\mathcal{K}_u, p(X_I, \mathcal{K}_u), U, Z\}$;

Algorithm 3 Cube-Tester-Subroutine

Input: $\{I, U, r_u\}$.
Output: $\{r_c\}$.
for $r_c = 1$ **to** $r_{c,max}$ **do** ▷ $r_{c,max}$ is set by the attacker.
  $T_0 = 0$;
  for $i = 0$ **to** $2^d - 1$ **do**
    $S = 0, S_I = i$; ▷ writes the binary representation of i in bits I of S.
    $T_0 = T_0 + Enc(S, 0, r_u, r_c)$;
  end for
  for $j = 1$ **to** M **do** ▷ M is the number of linearity tests, set by the attacker
    Randomly choose $x, y \in \{0,1\}^{|U|}$ and $k_1, k_2 \in \{0,1\}^{|\mathcal{K}|}$;
    $S_{1_Z} = 0$, $S_{1_U} = x$;
    $S_{2_Z} = 0$, $S_{2_U} = y$;
    $S_{3_Z} = 0$, $S_{3_U} = x + y$;
    $T_1 = 0, T_2 = 0, T_3 = 0$;
    for $i = 0$ **to** $2^d - 1$ **do**
      $S_{1_I} = i$, $S_{2_I} = i$, $S_{3_I} = i$;
      $T_1 = T_1 + Enc(S_1, k_1, r_u, rc)$;
      $T_2 = T_2 + Enc(S_2, k_2, r_u, rc)$;
      $T_3 = T_3 + Enc(S_3, k_1 + k_2, r_u, rc)$;
    end for
    if $T_1 + T_2 + T_3 + T_0 \neq 0$ **then return** $r_c - 1$; halt.
    end if
  end for
end for

Algorithm 4 Lower-Extension-Subroutine

Input: $\{r', \mathcal{K}_u, d, Time_{max}\}$.
Output: $\{r_l, \mathcal{K}_l, Time\}$
$\mathcal{K}_l = \{\}, r_l = 0, Time = N \cdot 2^{d+|\mathcal{K}_u|} + 2^{|\mathcal{K}|-N}$;
while $Time < Time_{max}$ **do**
  $\tilde{\mathcal{K}_l} \leftarrow \mathcal{K}_l, \tilde{Time} \leftarrow Time$;
  Trace T in forward direction for one round;
  Update $\mathcal{K}_l$;
  $Time = N \cdot 2^{d+|\mathcal{K}_u|} + \frac{r_l}{r'+r_l} N \cdot 2^{d+|\mathcal{K}_u \cup \mathcal{K}_l|} + 2^{|\mathcal{K}|-N}$;
  $r_l = r_l + 1$;
end while
return $\{r_l - 1, \tilde{\mathcal{K}_l}, \tilde{Time}\}$;

Algorithm 5 Online Phase

Input: $d, N, r_l, \mathcal{K}_u, \mathcal{K}_l, p^{(j)}(X_I, \mathcal{K}_u), 1 \leq j \leq N$
Output: secret key K.
for $k_1 = 0$ **to** $2^{|\mathcal{K}_u|} - 1$ **do**
  for $i = 0$ **to** $2^d - 1$ **do**
    for $j = 0$ **to** N **do**
      Compute plaintext $P(j, i, k_1) = p^{(j)}(X_I, \mathcal{K}_u)|_{X_I = i, \mathcal{K}_u = k_1}$;
      Query the r-round encryption oracle and save the corresponding ciphertext $C(j, i, k_1)$;
    end for
  end for
  for $k_2 = 0$ **to** $2^{|\mathcal{K}_l - \mathcal{K}_l \cap \mathcal{K}_u|} - 1$ **do**
    $fail = 0$; $j = 0$;
    while $j < N$ **and** $fail = 0$ **do**
      $T = 0$;
      for $i = 0$ **to** $2^d - 1$ **do**
        $T = T + Dec(C(j, i, k_1), (k_1, k_2), r, r_l)$;
      end for
      if $T \neq 0$ **then**
        $fail = 1$;
      end if
      $j = j + 1$;
    end while
    if $fail = 0$ **then**
      Randomly choose P and query its ciphertext C;
      for $k_3 = 0$ **to** $2^{|\mathcal{K} - \mathcal{K}_l \cup \mathcal{K}_u|} - 1$ **do**
        if $C = Enc(P, (k_1, k_2, k_3), 0, r)$ **then return** (k_1, k_2, k_3);
        end if
      end for
    end if
  end for
end for