Guess and Determine Analysis Based on Set Split *

Zhe CEN[1], Xiutao FENG[2**], Zhangyi WANG[3], Yamin ZHU[4] and Chunping CAO[1]

[1] Department of Computer Science and Technology, University of Shanghai for Science and Technology, Shanghai 200093, China
[2] Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Sciences, CAS, Beijing 100089, China
[3] School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China
[4] School of Science, Xi'an Technological University, Xi'an 710021, China

Abstract. The guess and determine attack is a common method in cryptanalysis. Its idea is to firstly find some variables which can deduced all remaining variables in a cipher and then traverse all values of these variables to find a solution. People usually utilize the exhausted search to find these variables. However, it is not applicable any more when the number of variables is a bit large. In this work we propose a guess and determine analysis based on set split to find as few variables as possible in the first step of guess and determine attack, which is a kind of exhausted search based on trading space for time and is more effective than the latter. Firstly we give an idea of set split in detail by introducing some conceptions such as base set, likely solution region and so on. And then we discuss how to utilize the set split to achieve a guess and determine analysis and give its specific implementation scheme. Finally, comparing it with the other two guess and determine analysis based on the exhausted search and the MILP method, we illustrate the effectiveness of our method by two ciphers Snow 2.0 and Enocoro-128v2. Our method spends about 0.000103 seconds finding a best solution of 9 variables for the former and 0.13 seconds finding a best solution of 18 variables for the latter in a personal Macbook respectively, which are better than those of both the exhausted search and the MILP method.

Keywords: guess and determine analysis, exhausted search, set split, Snow 2.0, Enocoro-128v2

1 Introduction

In recent years the guess and determine method as a common method has been wildly used in various fields such as logical reasoning, graph theory, equation

* This work was supported by National Natural Science Foundation (61972297) and National Key Research and Development Project(Grant 2018YFA0704705).
** Corresponding author: fengxt@amss.ac.cn

solving, cryptanalysis and so on. For a given deduction system of some variables and relations among them, it includes two phases: firstly find some key variables which can deduce all other variables by these relations; and then go through all values of these variables to find a solution. In this course people are more concerned about the minimum number of these variables, which is directly related to the complexity of searching a solution in the second phase. In cryptography we usually use it to find a trail of a guess and determine attack, and call it a guess and determine analysis.

The idea of guess and determine attacks first appeared in 1985 when Siegenthaler proposed a divide and conquer attack recovering the unknown initial state from a known key stream sequence [1]. In 1997, Golic applied the guess and determine attack to the alleged A5/1 and broke it theoretically [2]. In 2000, Philip and Gregory described a dummy SOBER-like cipher which was used to demonstrate how the guess and determine attack was performed and improved by exploiting multiples in some cases [3]. In 2003, they utilized the guess and determine attack to break the stream cipher SNOW [4]. In 2005 and 2009, Hadi et. al and Ding et. al introduced a guess and determine attack on the stream cipher SOSEMANUK respectively [5, 6]. The former's time complexity is 2^{226} and the latter's time complexity is 2^{192}. In 2010, Feng et. al presented a new byte-based guess and determine attack on SOSEMANUK, where they viewed a byte as a basic data unit and guessed some certain bytes of the internal states instead of the original 32-bit words, and dramatically reduced the time complexity to 2^{176} [7]. Furthermore, combining the idea of the guess and determine method and the time-memory tradeoff method, they further presented realtime key or state recovering attacks against a series of ciphers including A2U2 [8], FASER128/FASER256 [9], Sablier [10] and PANDA-s [11]. In 2006, Zhang and Feng proposed a new type of guess and determine attack on the self-shrinking generator [12]. Enes proposed a guess and determine method for filter generator [13] and Wei et.al further improved his method in 2012 [14]. In 2011, Charles et al. proposed a guess and determine attack on the round-reduced AES by local pruning and global pruning [15]. Their method is suitable for an equation system derived from a round-reduced block cipher with less variables. In 2013, Maria et. al proposed a method to store and propagate linear relations of variables efficiently in the guess and determine attack for Hash functions [16]. In 2014, Mohammad and Taranehan proposed an improved heuristic guess and determine attack of 5 guessed variables for SNOW 3G [17]. We find that M4 and M5 derived from M1 (see Section C in [17]) are always linearly dependent on M1, which can not reduce the number of guessed variables. Therefore their result is incorrect. In 2015, Mehmet et. al proposed a new guess and determine attack on 14-round Khudra where only 2 known plaintext-ciphertext pairs were required to mount the attack in a time complexity of 2^{64} encryption operations [18]. In 2017, Oleg and Stepan adopted the idea of the guess and determine method to simplify the system of equations and solved it by the SAT optimizer [19]. Recently we proposed a method of solving a minimizing problem of deduction systems based on mixed integer linear programming (in short, MILP) [20] and

applied it to the guess and determine attack for two ciphers Snow 2.0 [21] and Enocoro-128v2 [22].

The original guess and determine analysis is useful when the number of variables is small. However, it can not find a best solution quickly while the number of variables is a bit large. In this work we propose a guess and determine analysis based on set split, which is a kind of exhausted search based on trading space for time and is more effective than the latter. Firstly we give an idea of set split in detail by introducing some conceptions such as base set, likely solution region and so on. And then we discuss how to utilize the set split to achieve a guess and determine analysis and give its specific implementation scheme. Finally, comparing it with the other two guess and determine analyses based on the exhausted search and the MILP method, we illustrate the effectiveness of our method by two ciphers Snow 2.0 and Enocoro-128v2. Our method spends about 0.000103 seconds finding a best solution of 9 variables for the former and 0.13 seconds finding a best solution of 18 variables for the latter in a personal Macbook respectively, which are better than those of both the exhausted search and the MILP method.

The rest of this paper is organized as follows: in Section 2, we propose a theoretical guess and determine analysis based on set split; in Section 3, we give its specific implementation scheme; in Section 4, we apply it into two ciphers Snow 2.0 and Enocoro-128v2 and compare it with the other two methods.

2 Guess and determine attack based on set split

2.1 Guess and determine attack

In a guess and determine attack for a cipher, people usually extract some variables and rules from the cipher itself, where the rules characterize the relations among the variables. Let n and m be the number of the variables and the rules respectively. We represent them in a 2 tuple $(\mathcal{V}, \mathcal{R})$, where $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ is a set of the variables and $\mathcal{R} = \{r_1, r_2, \cdots, r_m\}$ is a set of the rules. The course of the guess and determine attack usually includes two steps: 1) find k variables which can deduce all other variables according to $\mathcal{R}$; 2) recover a solution by traversing all values of these k variables. Most of the time, people care more about the minimum of k since it is directly related to the complexity of searching all possible candidates in the second step. At present, people usually utilize the exhausted search to find these k variables and its complexity is about $\binom{n}{k}$. It is apparent that the exhausted search is not applicable any more when n is a bit large. In order to surmount this problem, we adopt the idea of exchanging space for time to split a huge search space into some small search spaces and remove some infeasible search spaces gradually. We call this course a set split. In the next section we will introduce how to use the idea of set split to achieve an efficient guess and determine attack.

2.2 Some conceptions on set split

For any nonempty subset B of $\mathcal{V}$, we call it a *base set* of $\mathcal{V}$. Below we introduce some key conceptions on set split.

Definition 1. *Let*

$$\mathcal{B} = \{B_1, B_2, \cdots, B_\tau\},$$

where τ is a positive integer and B_i is a base set of $\mathcal{V}$, $1 \leq i \leq \tau$. Then $\mathcal{B}$ is called a base set representative of $\mathcal{V}$ if $B_i \cap B_j = \varnothing$ for $1 \leq i \neq j \leq \tau$.

Let k be the number of guessed variables in a guess and determine attack. We fix k and call any a group of k variables in $\mathcal{V}$ to be a *likely solution*. Further it is a *solution* if it can deduce all other variables.

Definition 2. *Let $\mathcal{B} = \{B_1, B_2, \cdots, B_\tau\}$ be a base set representative of $\mathcal{V}$, where τ is a positive integer. Then $\mathcal{B}$ is called a likely solution region if it meets the following two conditions:*

1. $\tau \leq k$;
2. $\sum_{i=1}^{\tau} |B_i| \geq k$,

where $|B_i|$ means the size of B_i.

For an arbitrary base set representative $\mathcal{B} = \{B_1, B_2, \cdots, B_\tau\}$, we make convention that a likely solution X selected from $\mathcal{B}$ must satisfy that $X \cap B_i \neq \varnothing$ for all $1 \leq i \leq \tau$. It is obvious that we can always select some variables from each base set in $\mathcal{B}$ to make up a likely solution X if $\mathcal{B}$ is a likely solution region. Since we have to select at least one variable from each base set in $\mathcal{B}$, the number of selected variables is always not less than τ. When $\mathcal{B}$ is not a likely solution region, it is easy to check that any a group of selected variables is not a solution if $\tau > k$. On the other hand, if $\sum_{i=1}^{\tau} |B_i| < k$, the number of selected variables is always less than k. Therefore we have the following conclusion:

Theorem 1. *If a base set representative $\mathcal{B}$ is not a likely solution region, then $\mathcal{B}$ does not contain any solutions.*

Let M be a nonempty subset of $\mathcal{V}$. We call M an *infeasible solution set* if any a group of k variables of M is not a solution. It is easy to get the following conclusion:

Theorem 2. *Let M be an infeasible solution set. Then an arbitrary subset of M is also an infeasible solution set.*

2.3 Splitting a likely solution region

For a given base set B and infeasible solution set M, there are three relationships among them as shown in Fig.1.

- As shown in Fig.1.(1), $B \subseteq M$. We denote it by R_1.
- As shown in Fig.1.(2), $B \cap M = \varnothing$. We denote it by R_2.

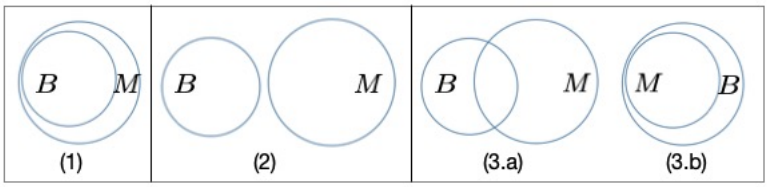

Fig. 1. Three relationships among B and M

– As shown in Fig.1.(3), $B \cap M \neq \varnothing$ and $B - M \neq \varnothing$. We denote it by R_3.

Below we introduce how to use an infeasible solution set M to split a likely solution region $\mathcal{B} = \{B_1, B_2, \cdots, B_\tau\}$ into some smaller likely solution regions. We discuss it in three cases:

Case 1. For all base sets in $\mathcal{B}$, all of them meet R_3, as shown in Fig.2. In this case each base set B_i is split into two base sets $B_{i,1}$ and $B_{i,2}$, where $B_{i,1} = B_i \cap M$ and $B_{i,2} = B_i - M$. For each $1 \leq i \leq \tau$, we have 3 ways to select base sets from $B_{i,1}$ and $B_{i,2}$, that is, $B_{i,1}$ or $B_{i,2}$ or both $B_{i,1}$ and $B_{i,2}$. Therefore we get totally 3^τ regions. Since the union of all base sets in the region $\{B_{1,1}, B_{2,1}, \cdots, B_{\tau,1}\}$ is a subset of M, there is no solution in this region by Theorem 2. We discard it. For the remaining $(3^\tau - 1)$ regions, we further discard all regions which are not likely solution regions by Definition 2.

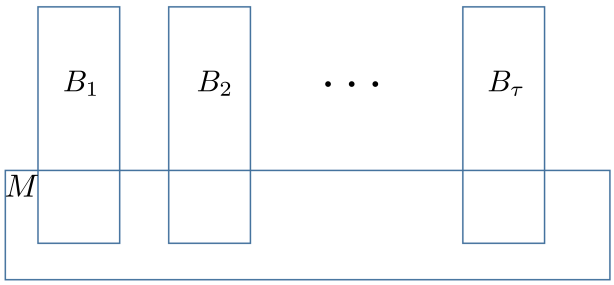

Fig. 2. The first split of the likely solution region $\mathcal{B}$

Case 2. For all base sets in $\mathcal{B}$, some of them meet R_1 and the rest meet R_2. Without loss of generality, we assume that λ base sets meet R_1, denoted by $\{B_1, B_2, \cdots, B_\lambda\}$, where $1 \leq \lambda \leq \tau$, as shown in Fig.3. In this case, $\mathcal{B}$ is not split since no base set is split. What is more, if $\lambda = \tau$, $\mathcal{B}$ is not a likely solution region any more since the union of all base sets in $\mathcal{B}$ is a subset of M.

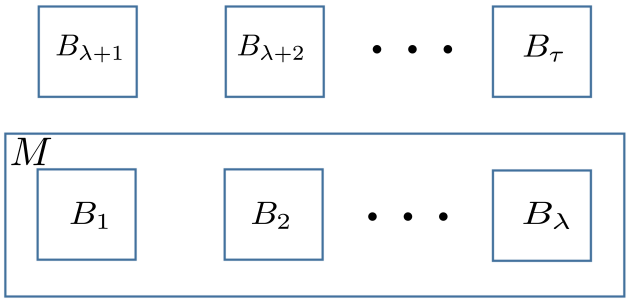

Fig. 3. The second split of the likely solution region $\mathcal{B}$

Case 3. For all base sets in $\mathcal{B}$, some of them meet R_1, some meet R_3, and the rest meet R_2. Without loss of generality, we assume that λ base sets meet R_1 and $(\gamma - \lambda)$ base sets meet R_3, denoted by $\{B_1, B_2, \cdots, B_\lambda\}$ and $\{B_{\lambda+1}, B_{\lambda+2}, \cdots, B_\gamma\}$ respectively, where $1 \leq \lambda < \gamma \leq \tau$, as shown in Fig.4. In this case each base set B_i is split into two base sets $B_{i,1}$ and $B_{i,2}$ for all $\lambda < j \leq \gamma$, where $B_{i,1} = B_i \cap M$ and $B_{i,2} = B_i - M$. For each $\lambda < i \leq \gamma$, we have 3 ways to select base sets from $B_{i,1}$ and $B_{i,2}$, that is, $B_{i,1}$ or $B_{i,2}$ or both $B_{i,1}$ and $B_{i,2}$. For $1 \leq i \leq \lambda$ or $\gamma < i \leq \kappa$, we only have 1 way to select a base set, that is itself. Therefore we get totally $3^{\gamma-\lambda}$ regions. If $\gamma = \tau$, since the union of all base sets in the region $\{B_1, B_2, \cdots, B_\lambda, B_{\lambda+1,1}, B_{\lambda+2,1}, \cdots, B_{\tau,1}\}$ is a subset of M, there is no solution in this region by Theorem 2. We discard it. For the remaining $(3^{\gamma-\lambda} - 1)$ regions, we further discard all regions which are not likely solution regions by Definition 2. If $\gamma \neq \tau$, we discard directly all regions which are not likely solution regions from all $3^{\gamma-\lambda}$ regions by Definition 2.

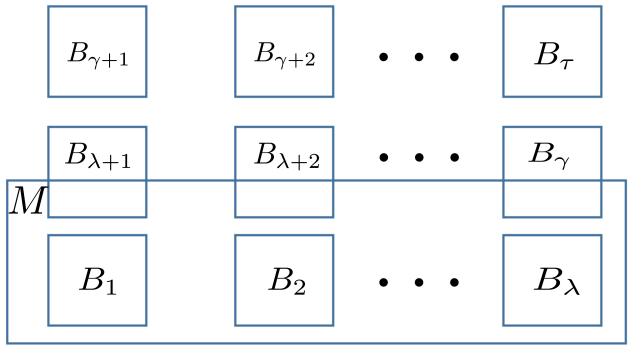

Fig. 4. The third split of the likely solution region $\mathcal{B}$

According to the above discussion of three cases, the split of a likely solution region is shown in Algorithm 1.

Algorithm 1 The split of a likely solution region

Input: A likely solution region $\mathcal{B} = \{B_1, B_2, \cdots, B_\tau\}$, an infeasible solution set M

Output: Several smaller likely solution regions or $\mathcal{B}$ or no likely solution region

Judge the relationship between M and B_i, $1 \leq i \leq \tau$, and denote by n_1, n_2 and n_3 the number of R_1, R_2 and R_3 respectively;
if $n_3 = \tau$ **then**
    According to Case 1, get $(3^\tau - 1)$ regions;
else
    if $n_3 = 0$ **then**
        if $n_2 = 0$ **then**
            Return no likely solution region;
        else
            Return $\mathcal{B}$;
        end if
    else
        if $n_2 = 0$ **then**
            According to Case 3, get $(3^{n_3} - 1)$ regions;
        else
            According to Case 3, get 3^{n_3} regions ;
        end if
    end if
end if
Return several smaller likely solution regions by Definition 2;

2.4 Maximizing deduction

In the guess and determine attack based on set split, it is a key to find an infeasible solution set whose size is as large as possible. Here we introduce two conceptions of maximizing deduction and maximal infeasible solution set.

Definition 3. *For an infeasible solution set M of $\mathcal{V}$, it is called a maximal infeasible solution set if any variable in $\mathcal{V} - M$ is added into M, M can deduced all other variables of $\mathcal{V}$. The course of getting a maximal infeasible solution set by an infeasible solution set is called a maximizing deduction.*

Below we give a method of maximizing deduction. Let M be an empty set. Firstly select k variables randomly from a likely solution region $\mathcal{B}$ and add them into M. And then add all possible variables deduced by M into M. If $M = \mathcal{V}$, it is ok and we find a solution luckily. If not, select a variable v from $\mathcal{V} - M$

and go on deducing by v and M. If all other variables in $\mathcal{V} - M$ are deduced, we will try another variable in $\mathcal{V} - M$ and go on deducing; otherwise, add v and variables deduced by v and M into M. Repeat the above step until all variables in $\mathcal{V} - M$ will be deduced no matter which variable in $\mathcal{V} - M$ is selected. At this time M is a maximal infeasible solution set. The maximizing deduction is shown in Algorithm 2.

Algorithm 2 Maximizing deduction.

Input: The 2 tuple $(\mathcal{V}, \mathcal{R})$, $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$, $\mathcal{R} = \{r_1, r_2, \cdots, r_m\}$, a likely solution region $\mathcal{B}$, the number k of guessed variables

Output: A maximal infeasible solution set M or a solution

1: Set $M = \varnothing$;
2: Select k variables from $\mathcal{B}$ and add them into M;
3: Deduce all possible variables by M and $\mathcal{R}$ and add them into M;
4: If $M = \mathcal{V}$, return a solution made of these k variables;
5: Let $N = \mathcal{V} - M$;
6: **for all** v in N **do**:
7: Let $M' = M \cup \{v\}$;
8: Deduce all possible variables by M' and $\mathcal{R}$ and add them into M';
9: If $M' \neq \mathcal{V}$, let $M = M'$ and go to Step 5;
10: **end for**
11: Return M;

2.5 Apply the set split to guess and determine analysis

For a guess and determine analysis, the set $\mathcal{V}$ of all variables itself is a likely solution region. According to the maximizing deduction, we can get a maximal infeasible solution set. As shown in Fig.5, we firstly select a likely solution region randomly and select a group of k variables from it as a likely solution, and then utilize it to deduce, maximize deduction and split likely solution regions in turn. Repeat the above step until a solution is found or no likely solution region is left. The specific course of the guess and determine analysis based on set split is shown in Algorithm 3.

3 An Implementation

In the previous section we provide a framework of the guess and determine attack based on set split. In this section we give its concrete implementation based on multi-thread parallel technology.

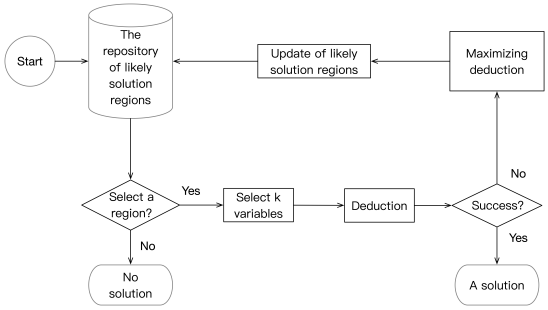

Fig. 5. A framework of the guess and determine analysis based on set split

Algorithm 3 The guess and determine analysis based on set split.

Input: The 2-tuple $(\mathcal{V}, \mathcal{R})$, $\mathcal{V} = (v_1, v_2, \cdots, v_n)$, $\mathcal{R} = (r_1, r_2, \cdots, r_m)$, the number k of guessed variables

Output: A solution or no solution

1: Let the repository $\mathcal{L} = \{\mathcal{V}\}$;
2: If $\mathcal{L} = \varnothing$, return no solution;
3: Select a likely solution region $\mathcal{B}$ from $\mathcal{L}$ and select k variables from $\mathcal{B}$;
4: Utilize these k variables and R to deduce other variables;
5: If all variables can be deduced, return a solution made of these k variables;
6: Utilize these k variables to conduct the maximizing deduction, and get a maximal infeasible solution set M;
7: Utilize M to split all likely solution regions in $\mathcal{L}$, and go to Step 2;

3.1 Overall framework

As shown in Fig.6, we utilize two kinds of threads to achieve a guess and determine attack based on set split. The task of the dispatch thread is to split large likely solution regions into some small likely solution regions or send some small likely solution regions to the task pool $\mathcal{T}$. Each search thread takes a likely solution region from $\mathcal{T}$ and searches all likely solutions in it.

3.2 Task pool $\mathcal{T}$

We use a circular queue of size s to store all likely solution regions in the task pool $\mathcal{T}$ so that we can synchronize the dispatch thread and the search threads conveniently, where s is a positive integer. As shown in Fig. 7, the dispatch thread always sends a likely solution region to the tail of the queue and the search thread always takes a likely solution region from the head of the queue. If $\mathcal{T}$ is full, the dispatch thread will wait. If $\mathcal{T}$ is empty, the search thread will wait.

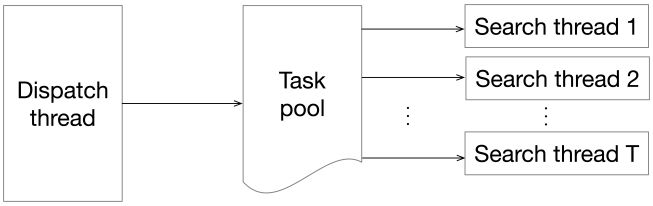

Fig. 6. Overall framework

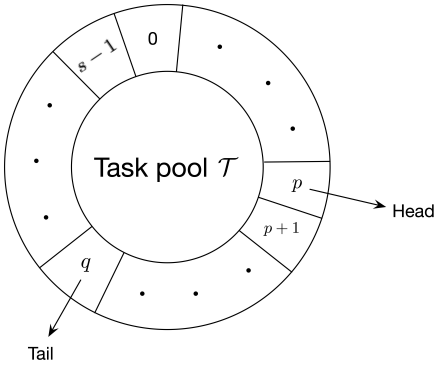

Fig. 7. The structure of the task pool $\mathcal{T}$

3.3 Dispatch thread

3.3.1 Work mode of dispatch thread

The main work of the dispatch thread is to split large likely solution regions in the main repository $\mathcal{M}$ and send small likely solution regions into $\mathcal{T}$. For a given likely solution region $\mathcal{B} = \{B_1, B_2, \cdots, B_\tau\}$, we define its *weight* as

$$wt(\mathcal{B}) = \sum_{i=1}^{\tau} |B_i|.$$

Let H be a threshold. We call $\mathcal{B}$ to be dispatchable if $wt(\mathcal{B}) < H$. When $\mathcal{B}$ is dispatchable, the dispatch thread sends it into $\mathcal{T}$. As shown in Fig. 8, we use Algorithm 4 to depict the work mode of the dispatch thread.

3.3.2 Update of main repository $\mathcal{M}$

The main repository $\mathcal{M}$ adopts a layer structure to store all likely solution regions. As shown in Fig.9, $\mathcal{M}$ totally includes d layers of storage units and each layer store at least L likely solution regions. We call L a minimum storage length. Since most likely solution regions in a layer have some common base

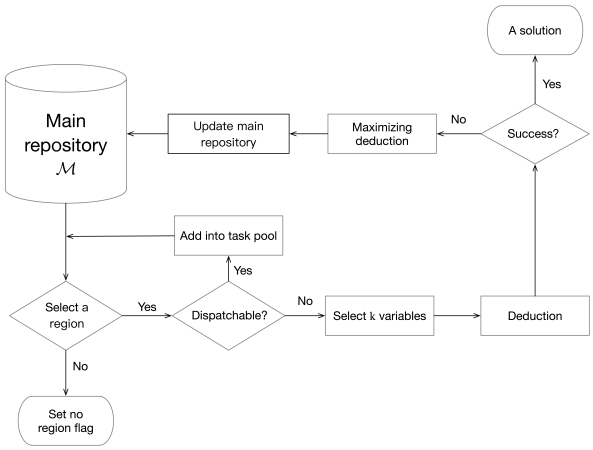

Fig. 8. The work mode of the dispatch thread

Algorithm 4 The work mode of the dispatch thread.

Input: The 2-tuple $(\mathcal{V}, \mathcal{R})$, $\mathcal{V} = (v_1, v_2, \cdots, v_n)$, $\mathcal{R} = (r_1, r_2, \cdots, r_m)$, the number k of guessed variables, the threshold H

Output: A solution or a flag of no region

1: Let $\mathcal{M} = \{\mathcal{V}\}$;
2: If $\mathcal{M} \neq \varnothing$, select a likely solution region $\mathcal{B}$ from $\mathcal{M}$; otherwise, wait or return a flag of no region;
3: If $wt(\mathcal{B}) < H$, add $\mathcal{B}$ into $\mathcal{T}$ and remove $\mathcal{B}$ from $\mathcal{M}$, and then go to Step 2;
4: Select k variables from $\mathcal{B}$ and deduce other variables by $\mathcal{R}$;
5: If all variables are deduced, return a solution made of these k variables;
6: Conduct a maximizing deduction by these k variables and get a maximal infeasible solution set M;
7: Invoking Algorithm 1 to update $\mathcal{M}$ by M and go to Step 2;

sets, in order to avoid splitting these base sets repeatedly, we use a base set representative to store all base sets appearing in this layer and a set of their positions to represent a likely solution region.

Below we introduce an interaction between the dispatch thread and $\mathcal{M}$. The dispatch thread always selects the last likely solution region in the last layer, denoted by $\mathcal{B}_{\cdot,j}$. If it is dispatchable, we send it into $\mathcal{T}$ and remove it from the current layer. If all likely solution regions are removed at the current layer, go back to the previous layer. If $\mathcal{B}_{\cdot,j}$ is not dispatchable and $j > L$, we put it into the next layer and regard it as a base set representative of this layer. Then we select k variables randomly from $\mathcal{B}_{\cdot,j}$ as a likely solution X. If all variables are deduced by X, it is ok and we find a solution luckily; Otherwise, we conduct a maximizing deduction and get a maximal infeasible solution set M. And then

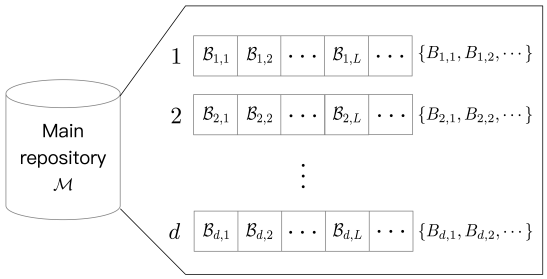

Fig. 9. The structure of the main repository $\mathcal{M}$

we utilize M to split all likely solution regions in the last layer and update its base set representative.

Here we consider how many likely solution regions a layer contains at most. Since at most L likely solution regions are split in a layer and each likely solution region can be split into at most 3^k regions, each layer contains at most $3^k L$ likely solution regions. When k is a bit large, both the storage space and computational complexity are too large. In order to overcome the above problem, we set an upper bound for the number of split base sets, denoted by K. When the number of split base sets is more than K, the other base sets will not be split. Below we illustrate the update of a base set representative.

For a given base set representative $\mathcal{B} = \{B_1, B_2, \cdots, B_\tau\}$, we assume that total w base sets in $\mathcal{B}$ are split into two base sets, $0 \leq w \leq K$. Without loss of generality, we assume that these w base sets are $B_1, B_2, \cdots, B_w$ and the remaining base sets are $B_{w+1}, B_{w+2}, \cdots, B_\tau$. For all $1 \leq i \leq w$, each B_i can be split into two base sets $B_{i,1}$ and $B_{i,2}$, where $B_{i,1} = B_i \cap M$ and $B_{i,2} = B_i - M$. We get a new base set representative:

$$\mathcal{B}' = \{B_{1,1}, B_{1,2}, B_{2,1}, B_{2,2}, \cdots, B_{w,1}, B_{w,2}, B_{w+1}, B_{w+2}, \cdots, B_\tau\}.$$

3.3.3 Random work mode of dispatch thread

In the above section the dispatch thread goes through every likely solution region in $\mathcal{M}$ in turn. When the number of all likely solution regions is too large, the complexity of going through the whole likely solution regions is still high. What is more, since all likely solution regions in a layer are from a common likely solution region in the previous layer, they are similar to each other in some extent. The deeper the layer is, the more similar these likely solution regions are. If we can not find a solution from a likely solution region, it is difficult to find a solution in its similar likely solution regions. That means it is not necessary to split all likely solution regions every time when the layer depth is a bit large. Therefore we design a random work mode for the dispatch thread. For a layer where the number of likely solution regions is more than L, the dispatch

thread does not select all likely solution regions any more, but selects several likely solution regions randomly, and then splits them or sends them into $\mathcal{T}$.

3.4 Search thread

Unlike the dispatch thread, the aim of the search thread is to pick up a likely solution region from $\mathcal{T}$ and search its all likely solutions. In order to speed up the search, we still utilize set split to search the likely solution region. As shown in Fig. 10, we use Algorithm 5 to depict the search thread.

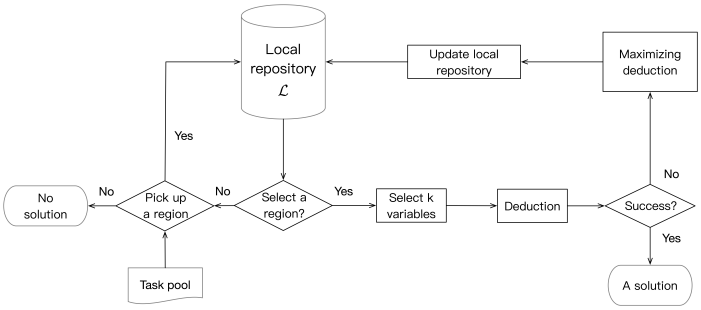

Fig. 10. The workflow of the search thread

Algorithm 5 The workflow of the search thread.

Input: The 2-tuple $(\mathcal{V}, \mathcal{R})$, $\mathcal{V} = (v_1, v_2, \cdots, v_n)$, $\mathcal{R} = (r_1, r_2, \cdots, r_m)$, the number k of guessed variables

Output: A solution or no solution

1: If $\mathcal{T} \neq \varnothing$, pick up a likely solution region $\mathcal{B}$ from $\mathcal{T}$; otherwise, wait or return no solution;
2: Let $\mathcal{L} = \{\mathcal{B}\}$;
3: If $\mathcal{L} \neq \varnothing$, select a likely solution region $\mathcal{B}'$ from $\mathcal{L}$; otherwise, go to Step 1;
4: Select k variables from $\mathcal{B}'$ and deduce other variables by $\mathcal{R}$;
5: If all variables are deduced, return a solution made of these k variables;
6: Conduct a maximizing deduction by these k variables and get a maximal infeasible solution set M;
7: Invoking Algorithm 1 to update $\mathcal{L}$ by M and go to Step 3;

Since the scale of likely solution regions in $\mathcal{T}$ is small, we design a local repository $\mathcal{L}$ which only contains a layer of storage units, as shown in Fig. 11. Unlike the update of $\mathcal{M}$, there is no limitation about the number of split base sets or the minimum storage length of $\mathcal{L}$.

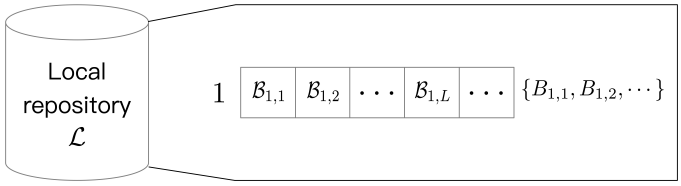

Fig. 11. The structure of the local repository $\mathcal{L}$

4 Experiment

In order to illustrate the effectiveness of the guess and determine analysis based on set split, we apply it to two stream ciphers Snow 2.0 and Enocoro-128v2 and compare it with the other two guess and determine attacks based on the exhausted search and the MILP method respectively. Below we recall Snow 2.0 and Enocoro-128v2 briefly.

4.1 Snow 2.0 and Enocoro-128v2

Snow 2.0 is one of the most important stream ciphers and has been an ISO/IEC standard [23]. Enocoro-128v2 is Japanese lightweight stream cipher standard. At present, the best solution is to guess 9 variables for the guess and determine attack to Snow 2.0 and 18 variables to Enocoro-128v2 [20]. We assume that 10 and 17 key words are known for Snow 2.0 and Enocoro-128v2 respectively. Referring to [20], the numbers of variables and rules after the permeation criteria and independent variable criteria are shown in Table 1.

Table 1. The numbers of variables and rules of Snow 2.0 and Enocoro-128v2.

Cipher	Key words	Variables	Rules
Snow 2.0	10	35	27
Enocoro-128v2	17	101	74

4.2 Experimental results

For Snow 2.0 and Enocoro-128v2, we conduct guess and determine analysis based on set split under the same parameter settings according to Section 3, where the number of search threads is 2 and the minimum storage length is 512. And then we compare the results of our method with the other two methods, as shown in Table 2. It is shown that our method is significantly better than the other two methods.

[5] For Snow 2.0, we only split the search space of Snow 2.0 twice and then go through all likely solution regions. We spend 0.000103 seconds finding a solution on average and reduce the size of search space by about 11 percent.

Table 2. The comparison of three methods.

Method/Cipher	SNOW 2.0 (k = 9)	Enocoro-128v2 (k = 18)
Exhaustive Search	0.000112 s	N/A
MILP	0.1 s	3.5 m
Base Splitting	0.000103 s [5]	0.13 s

References

1. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", IEEE Transaction on Computer, Vol. 34, pp.81–85, 1985.
2. J. Golic, "Cryptanalysis of alleged A5 stream cipher", EUROCRYPT'97, LNCS 1233, pp.239-255, 1997.
3. H. Philip and R. G. Gregory, "Exploiting multiples of the connection polynomial in word-oriented stream ciphers", ASIACRYPT, pp.302-316, 2000.
4. H. Philip and R. G. Gregory, "Guess-and-determine attacks on SNOW", SAC 2002, LNCS 2595, pp.37–46, 2002.
5. A. Hadi and E. Taraneh, "Advanced guess and determine attacks on stream cipher", IST, pp.87-91, 2005.
6. L. Ding and J. Guan, "guess and determine attack on SOSEMANUK", International Conference on Information Assurance and Security 2009, pp.658-661, 2009.
7. X. T. Feng, J. Liu, Z. C. Zhou, C. K. Wu and D. G. Feng, "A byte-based guess and determine attack on SOSEMANUK", ASIACRYPT 2010, pp.146-157, 2010.
8. Z. Q. Shi, X. T. Feng, D. G. Feng and C. K. Wu, "A real-time key recovery attack on the lightweight stream cipher A2U2", CANS 2012, pp.12-22, 2012.
9. X. T. Feng and F. Zhang, "A realtime key recovery attack on the authenticated cipher FASER128", IACR Cryptology ePrint Archive, https://eprint.iacr.org/2014/258, 2014.
10. X. T. Feng and F. Zhang, "Cryptanalysis on the authenticated cipher sablier", NSS 2014, Vol. 8792, pp.198-208, 2014.
11. X. T. Feng, F. Zhang and H. Wang, "A practical forgery and state recovery attack on the authenticated cipher PANDA-s", IACR Cryptology ePrint Archive, https://eprint.iacr.org/2014/325, 2014.
12. B. Zhang and D. G. Feng, "New guess-and-determine attack on the self-shrinking generator", ASIACRYPT, LNCS 4284, pp.54–68, 2006.
13. P. Enes, "On guess and determine cryptanalysis of LFSR-based stream ciphers", IEEE Transactions on Information Theory, Vol. 55, Issue 7, pp.3398-3406, 2009.
14. Y. Z. Wei, P. Enes and Y. P. Hu, "Guess and determine attacks on filter generators-revisited", IEEE Transactions on Information Theory, Vol. 58, Issue 4, pp.2530-2539, 2012.
15. B. Charles, D. Patrick and F. A. Pierre, "Automatic search of attacks on round-reduced AES and applications", CRYPTO 2011, LNCS 6841, pp.169–187, 2011.
16. E. Maria, M. Florian, N. Tomislav, R. Vincent and S. Martin, "Linear propagation in efficient guess-and-determine attacks", WCC 2013, pp.1-10, 2013.
17. N. N. S. Mohammad and E. Taraneh, "Improved heuristic guess and determine attack on SNOW 3G stream cipher", International Symposium on Telecommunications 2014, pp.972–976, 2014.

18. Ö. Mehmet, Ç. Mustafa, K. Ferhat, "A guess-and-determine attack on reduced-round Khudra and weak keys of full cipher", IACR Cryptology ePrint Archive, http://eprint.iacr.org/2015/1163, 2015.
19. Z. Oleg and K. Stepan, "An improved SAT-based guess-and-determine attack on the alternating step generator", International Conference on Information Security 2017, LNCS 10599, pp.21-38, 2017.
20. Z. Cen, X. T. Feng, Z. Y. Wang and C. P. Cao, "Minimizing deduction system and its application", https://arxiv.org/abs/2006.05833, 2020.
21. E. Patrik and J. Thomas, "A new version of the stream cipher SNOW", SAC 2002, LNCS 2595, pp.47–61, 2002.
22. D. Watanabe, K. Okamoto and T. Kaneko, "A hardware-oriented light weight pseudo-random number generator Enocoro-128v2", The Symposium on Cryptography and Information Security, 3D1–3, 2010.
23. ISO/IEC 18033–4: Information technology - Security techniques - Encryption algorithms - Part 4: Stream ciphers, 2011.