Theoretical Computer Science 676 (2017) 52–68

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Solving polynomial systems with noise over $\mathbb{F}_2$: Revisited

Zhenyu Huang [a,b,*], Dongdai Lin [a]

[a] SKLOIS, Institute of Information Engineering, CAS, Beijing 100093, China
[b] KLMM, Academy of Mathematics and Systems Science, CAS, Beijing 100190, China

ARTICLE INFO

Article history:
Received 15 July 2016
Received in revised form 9 February 2017
Accepted 13 March 2017
Available online 18 March 2017
Communicated by G. Persiano

Keywords:
Boolean polynomial system with noise
Max-PoSSo
ISBS method
Cold Boot attack
Serpent

ABSTRACT

Solving polynomial systems with noise over $\mathbb{F}_2$ is a fundamental problem in computer science, especially in cryptanalysis. **ISBS** is a new method for solving this problem based on the idea of incrementally solving the noisy polynomial systems and backtracking all the possible noises, and it has better performance than other methods in solving some problems generated from cryptanalysis. In this paper, some further researches on **ISBS** are presented. The structure and size of the search tree of **ISBS** are theoretically analyzed. Then two major improvements, artificial noise-bound strategy and s-direction approach, are proposed. Based on these improvements, a modified **ISBS** algorithm is implemented, and the experiments of solving the Cold Boot key recovery problems of the block cipher Serpent with symmetric noise, show that this modified algorithm is more efficient than the original one.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Solving polynomial systems with noise over $\mathbb{F}_2$, which is called as the Max-PoSSo problem over $\mathbb{F}_2$, is the problem of finding a solution of a given Boolean polynomial system, such that the solution can satisfy the maximum number of polynomials. It is a fundamental problem in several areas of cryptography, such as algebraic attacks, side-channel attacks and the cryptanalysis of LPN/LWE-based schemes. For example, in the Cold Boot attack, which is a kind of side-channel attack, one can recover the initial key of a block cipher from noisy round keys by solving a Max-PoSSo problem [1,9]. In the computation complexity field, this problem is also significant and is known as the maximum equation satisfying problem [8,13]. In the general case, this problem is NP-hard even when the polynomials are linear.

In this paper, we focus on the Max-PoSSo problems with all input polynomials being nonlinear. Obviously, Max-PoSSo problems over $\mathbb{F}_2$ are analogous to the well-known Max-SAT problems, thus a natural way to solve a Max-PoSSo problem is converting it into a Max-SAT problem and then solving it with a Max-SAT solver. However, this method has a disadvantages that the original algebraic structure is destroyed after the conversion. In [1], the authors proposed a method to convert Max-PoSSo problems into mixed integer programming (**MIP**) problems, and then solved it with a **MIP** solver **SCIP**. Essentially, these two kinds of methods are all based on the idea of searching all the possible values of variables, and their differences are the techniques of pruning redundant branches of the search tree.

Since Max-PoSSo is a NP-hard problem, it is natural to consider whether there is a polynomial-time approximate algorithm with a large approximate ratio r. In [13], Zhao and Gao gave a negative result that when the degree of polynomials

[*] Corresponding author at: SKLOIS, Institute of Information Engineering, CAS, Beijing 100093, China.
E-mail addresses: huangzhenyu@iie.ac.cn (Z. Huang), ddlin@iie.ac.cn (D. Lin).

http://dx.doi.org/10.1016/j.tcs.2017.03.005
0304-3975/© 2017 Elsevier B.V. All rights reserved.

are 2 and the number of variables is large, the maximal approximation ratio that can be achieved in polynomial time for Max-PoSSo over $\mathbb{F}_2$ is $1/2$. In practical computation, since Max-PoSSo problems can be converted to Max-SAT or MIP problems, approximate algorithms for solving these problems can be used to solve Max-PoSSo. Like the above deterministic algorithms, these approximate algorithm also focus on the possible values of variables, not the possible values of polynomials.

In [11], a new method called **ISBS** for solving Max-PoSSo problems over $\mathbb{F}_2$ was proposed. The basic idea of **ISBS** is searching the values of polynomials, which is equivalent to searching all the possible noises. Precisely speaking, given a noisy polynomial system $\{f_1, f_2, \ldots, f_m\}$, one tries to solve polynomial systems $\{f_1 + e_2, f_2 + e_2, \ldots, f_m + e_m\}$, where $(e_1, e_2, \ldots, e_m)$ can be equal to $(0, 0, \ldots, 0), (1, 0, \ldots, 0), \ldots, (1, 1, \ldots, 1)$. Then, the solution of a system $\{f_1 + e_2, f_2 + e_2, \ldots, f_m + e_m\}$ with $(e_1, e_2, \ldots, e_m)$ having the smallest Hamming weight is the solution of the Max-PoSSo problem. In the **ISBS** method, the above basic idea is combined with the ideas of incrementally solving $\{f_1 + e_1, f_2 + e_2, \ldots, f_m + e_m\}$ and searching all possible $(e_1, e_2, \ldots, e_m)$ with backtracking. By this way, one can prune a lot of search branches. The experimental results of [11] showed that compared with **SCIP**, **ISBS** has better performances in solving the Max-PoSSo problems generated from the Cold Boot Key recovery problems of block ciphers AES [12] and Serpent [2].

In **ISBS**, the incrementally solving process can be executed by running some polynomial system solving algorithm, such as the Characteristic Set algorithms [4,7] and the Gröbner Basis algorithms [5,6]. We can see this solving process as a black-box process, and in this case, a major factor that influences the complexity of **ISBS** is the number of branches in the search tree with respect to the noises $(e_1, e_2, \ldots, e_m)$. The motivation of this paper is to theoretically analyze the number of branches in this tree, then show the relation of the basic properties of the input system to this number of branches. Furthermore, based on the theoretical analysis, we want to develop some techniques which can further decrease the number of branches, hence improve the efficiency of **ISBS**. The main contributions of this paper are as follows.

We give some theoretical analysis of the structure of the search trees of **ISBS**. We prove that the search tree of **ISBS** is equivalent to the intersection of two binary tree $\mathbb{T}_{quasi}$ and $\mathbb{T}_{nw}$, where the number of branches in $\mathbb{T}_{quasi}$ is determined by the randomness of the input system, while the number of branches in $\mathbb{T}_{nw}$ is determined by the number of the input polynomials and the bound of the Hamming weight of the possible noises.

For general polynomial systems, we propose the artificial noise-bound strategy to decrease the size of $\mathbb{T}_{nw}$. In this strategy, we artificially bound the Hamming weight of the possible noises, and gradually increase this bound until we find the optimal solution. Moreover, when the input polynomial system satisfies $m = sn$, where m is the number of polynomials, n is the number of variables and $s \geq 2$ is an integer, we propose the s-direction approach, whose idea is dividing the input polynomials into s parts and searching the possible noises of different parts with smaller bounds, to decrease the number of branches in the search tree. We prove that this approach is more efficient than the original **ISBS**, and show that in which way of setting the smaller bounds and dividing the polynomial system, this approach can be more efficient.

Finally, we implement a modified **ISBS** algorithm by applying the above improvements, and test it by solving some Cold Boot key recovery problems of the block cipher Serpent. We compare our experimental results with those in [1,11], and the experimental results demonstrate that the efficiency of **ISBS** is improved significantly by our modification.

The rest of this paper is organized as follows. In Section 2, we introduce the Max-PoSSo problem and the **ISBS** method. In Section 3, the number of branches in the search tree of **ISBS** is analyzed. In Section 4, the s-direction approach for solving the problems with $m = sn$ is proposed. In Section 5, we show some experimental results. In Section 6, the conclusions are presented.

2. Solving Max-PoSSo problems over $\mathbb{F}_2$

Let $\mathbb{F}_2$ be the finite field with two elements $\{0, 1\}$, and $\{x_1, x_2, \ldots, x_n\}$ be a variable set. Consider the Boolean polynomial ring $\mathbb{R}_2 = \mathbb{F}_2[x_1, x_2, \ldots, x_n]/\langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n\rangle$. An element in $\mathbb{R}_2$ is called a Boolean polynomial.

Given a Boolean polynomial system $\mathbf{P} = \{f_1, \ldots, f_m\} \subset \mathbb{R}_2[x_1, \ldots, x_n]$. The polynomial system solving (PoSSo) problem over $\mathbb{F}_2$ is finding a solution $(x_1, ..., x_n) \in \mathbb{F}_2^n$ such that $\forall f_i \in \mathbf{P}$, we have $f_i(x_1, ..., x_n) = 0$. The set consisting of all the solutions of the PoSSo problem is called the *zero set* of $\mathbf{P}$.

The Max-PoSSo problem over $\mathbb{F}_2$ is defined as below.

Max-PoSSo: *Let* $\mathbf{P} = \{f_1, \ldots, f_m\} \subset \mathbb{R}_2[x_1, \ldots, x_n]$ *be a Boolean polynomial system. Find a point* $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$ *such that* $\{i \in N | f_i(x_1, \ldots, x_n) = 0, f_i \in \mathbf{P}\}$ *has maximal cardinality.*

The name "Max-PoSSo" was first proposed in [1]. In the computational complexity field, this problem is sometimes called the maximum equation satisfying problem [8,13]. Obviously, Max-PoSSo is at least as hard as PoSSo. Moreover, whether the polynomials in $\mathbf{P}$ are linear or not, Max-PoSSo is an NP-hard problem. Besides Max-PoSSo, in [1], the authors introduced another variant problem: Partial Weighted Max-PoSSo, in which the solution is constrained by another polynomial system and it has to maximize a cost function. In this paper we focus on the Max-PoSSo problem, since it is a more fundamental problem and study this problem can help us illustrate the major properties of these similar problems. Moreover, the techniques used in solving Max-PoSSo can be simply applied into solving Partial Weighted Max-PoSSo and other similar problems.

In the following paragraphs of this paper, unless otherwise stated, the problems we discuss are all over $\mathbb{F}_2$, and we use n to denote the number of variables and m to denote the number of input polynomials.

Before introducing **ISBS** method, we first discuss the problem of recovering the true solutions of Max-PoSSo problems originated from cryptanalysis in the following subsection.

2.1. The success rate of recovering the true solution

In cryptanalysis, the input polynomial system $\mathbf{P}$ of a Max-PoSSo problem is always originated from another system $\mathbf{P}_0$ which always has a solution. Moreover, the difference between $\mathbf{P}_0$ and $\mathbf{P}$ is that t polynomials of them have different constant terms. If $|\mathbf{P}_0| = |\mathbf{P}| = m$, the ratio $r = t/m$ is called the error rate of $\mathbf{P}$. The solution of $\mathbf{P}_0$ is called *the true solution* of $\mathbf{P}$.

Actually, in cryptanalysis, the major motivation of solving a Max-PoSSo problem is to recover the true solution which is the sensitive information in most cases. Sometimes when the error rate r is big, the solution of the Max-PoSSo problem may not be the true solution of the input polynomial system. Therefore, we want to know in what cases we have big probability of recovering the true solution, and how this probability changes when m, n and r change. Let $\mathcal{P}$ be the probability of recovering the true solution by solving a Max-PoSSo problem, and call it the *success rate*. Now we present some results about $\mathcal{P}$ under the following assumption.

Given a polynomial system, $\mathbf{P} = \{f_1(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n)\}$ with $m \geq n$, we define a map $\mathcal{S}_m : \mathbb{F}_2^n \to \mathbb{F}_2^m$, with $\mathcal{S}_m(x) = (f_1(x), f_2(x), \dots, f_m(x))$. Assume for any x, the probability of $\mathcal{S}_m(x)$ being equal to each element in $\mathbb{F}_2^m$ is the same. Moreover, for any $x_1 \neq x_2$, $\mathcal{S}_m(x_1)$ and $\mathcal{S}_m(x_2)$ have independent distribution. Obviously, this assumption is reasonable when each f_i is random and balanced. Then we have the following proposition about the success rate.

Proposition 1. *Let* $\mathbf{P}$ *be the input system of a Max-PoSSo problem with error rate r. Suppose* $\mathbf{P}$ *has n variables and m polynomials, and the above assumption about the distribution of* $\mathcal{S}_m(x)$ *is valid. Then the success rate* $\mathcal{P}$ *is equal to* $(1 - \frac{\sum_{i=0}^{rm}\binom{m}{i}-1}{2^m})^{2^n}$.

Proof. Suppose $\mathbf{P}$ is generated from $\mathbf{P}_0$. Then t polynomials of $\mathbf{P}$ and $\mathbf{P}_0$ have different constant terms. We denote the differences between the elements of $\mathbf{P}$ and $\mathbf{P}_0$ by $e'_1, e'_2, \dots, e'_m$, then $e'_i = 0$ or 1 for $1 \leq i \leq m$. The solution of the Max-PoSSo problem about $\mathbf{P}$ is the true solution when the following condition is satisfied:

- For any vector $(e_1, e_2, \dots, e_m) \in \mathbb{F}_2^m$ such that its Hamming weight $w \leq t = mr$ and $(e_1, e_2, \dots, e_m) \neq (e'_1, e'_2, \dots, e'_m)$, polynomials $f_1 + e_1, f_2 + e_2, \dots, f_m + e_m$ don't have common solutions, which means for any point $x_0 \in \mathbb{F}_2^n$, $(f_1(x_0), f_2(x_0) \dots, f_m(x_0)) \neq (e_1, e_2, \dots, e_m)$.

Therefore, the number of vectors that satisfy the above condition, is $\sum_{i=0}^{t}\binom{m}{i} - 1$. By the assumption about the distribution of $\mathcal{S}_m$, we can deduce that for any point x_0, the probability of $\mathcal{S}_m(x_0)$ being not equal to these vectors is $1 - \frac{\sum_{i=0}^{t}\binom{m}{i}-1}{2^m}$. Since there are 2^n points in $\mathbb{F}_2^n$, then the success rate $\mathcal{P}$, which is equal to the probability of the above condition being satisfied, is $(1 - \frac{\sum_{i=0}^{mr}\binom{m}{i}-1}{2^m})^{2^n}$. □

Proposition 2. *Let* $r \leq 1/2$ *be a fixed real number and* $1/r$ *be a positive integer. For any integers* $m_2 > m_1$, *we have*

$$\frac{\sum_{i=0}^{m_2 r}\binom{m_2}{i}}{2^{m_2}} \leq \frac{\sum_{i=0}^{m_1 r}\binom{m_1}{i}}{2^{m_1}}.$$

Moreover, the equality holds when $r = 1/2$.

Proof. The completely strict proof of this proposition is complicated and is given in Appendix. Here we present an approximate proof by the normal distribution theory.

By the central limit theorem, we know that when m is large enough, $\frac{\sum_{i=0}^{mr}\binom{m}{i}}{2^m}$ is approximately equal to

$$\frac{1}{\sqrt{m\pi/2}} \int_{-\infty}^{mr} e^{-\frac{1}{2}(\frac{t-m/2}{\sqrt{m/4}})^2} dt = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{mr-m/2}{\sqrt{m/4}}} e^{-\frac{1}{2}s^2} ds$$

Thus, it is sufficient to prove $\frac{m_1 r - m_1/2}{\sqrt{m_1/4}} \geq \frac{m_2 r - m_2/2}{\sqrt{m_2/4}} \Leftrightarrow (\sqrt{m_1} - \sqrt{m_2})(2r - 1) \geq 0$. Obviously, when $r \leq 1/2$, this inequality holds, and when $r = 1/2$, the equality holds. □

From the above proposition, we can deduce that when n, r are fixed, $\frac{\sum_{i=0}^{mr}\binom{m}{i}-1}{2^m}$ decreases with m increasing, which means $\mathcal{P}$ increase with m increasing. Similarly, we can prove that when n, m are fixed, $\mathcal{P}$ decreases with r increasing.

Actually, for most problems in cryptanalysis, r and n are always fixed, such as the Cold Boot key recovery problem introduced in Section 5, and the probabilistic algebraic attacks on LFSR-based stream ciphers [3]. It means that by increasing the number of input polynomials, we can increase the success rate of recovering the true solution.

2.2. The Incremental Solving and Backtracking Search (ISBS) method

Now we introduce the **ISBS** method for solving Max-PoSSo problems [11]. As we introduced in Section 1, the principle idea of **ISBS** is searching the values of polynomials. Let's show it more specifically.

Given a noisy polynomial set $\mathbf{P} = \{f_1, f_2, \ldots, f_m\}$, for every vector $E = (e_1, e_2, \ldots, e_m) \in \mathbb{F}_2^m$, we can solve the polynomial system $\{f_1 + e_1, f_2 + e_2, \ldots, f_m + e_m\}$ by some algebraic methods. Hence, we can exhaustively search all such E in the order of increasing Hamming weight and solve the PoSSo problem of the corresponding polynomial system for each E. For some E, if the corresponding PoSSo problem has a solution, then this solution is the solution of the Max-PoSSo problem.

The above approach uses the most common way to search E, and obviously there are a lot of redundant computations. For example, if $\{f_1 + e'_1, f_2 + e'_2, \ldots, f_k + e'_k\}$ has no solution for some fixed $(e'_1, e'_2, \ldots, e'_k)$, we don't need to solve any system with the form $\{f_1 + e'_1, f_2 + e'_2, \ldots, f_k + e'_k, f_{k+1} + e_{k+1}, \ldots, f_m + e_m\}$, since it will always be a contradiction system. Therefore, in order to avoid this kind of redundant computations, **ISBS** combines the incremental solving method and the backtracking search method with the above idea.

First let's introduce the incremental solving method. Given a polynomial system $\mathbf{P}$, we can solve it by some algebraic methods, such as the Characteristic Set (CS) method [4,7] and the Gröbner Basis method [5,6].[1] In the following, we see this polynomial system solving process as a black-box process. If the input of this black-box is a polynomial system $\mathbf{P}$, we denote its output results as $\texttt{Result}(\mathbf{P})$, which is called the *result set* of $\mathbf{P}$. When the polynomial system $\mathbf{P}$ has no solution, we set $\texttt{Result}(\mathbf{P})$ to be $\{1\}$. According to the theories of the CS method and the Gröbner Basis, all the solutions of $\mathbf{P}$ can be derived easily from $\texttt{Result}(\mathbf{P})$. We remind the reader that, for different methods, $\texttt{Result}(\mathbf{P})$ can be different. For example, if we use the CS method to solve $\mathbf{P}$, $\texttt{Result}(\mathbf{P}) = \cup_i \mathcal{A}_i$ is the union of a group of triangular sets (a triangular set $\mathcal{A}_i$ is a polynomial set whose solutions can be easily achieved, and its precise definition will be found in [7]). If we use the Gröbner Basis method to solve $\mathbf{P}$, $\texttt{Result}(\mathbf{P})$ is the Gröbner Basis of ideal $\langle \mathbf{P} \rangle$.

Now we show that given $\texttt{Result}(\mathbf{P})$ and a polynomial g, $\texttt{Result}(\{\texttt{Result}(\mathbf{P}), g\})$ can be achieved. For example, for the CS method, we need to compute each $\texttt{Result}(\{\mathcal{A}_i, g\})$ and output the union of them. For the Gröbner Basis method, we need to compute the Gröbner Basis of the ideal generated by $\{\mathcal{A}_i, g\}$. Therefore, given a polynomial system $\mathbf{P} = \{f_1, f_2, \ldots, f_m\}$, $\texttt{Result}(\mathbf{P})$ can be achieved by incrementally computing

$$\texttt{Result}(\{f_1\}), \texttt{Result}(\{\texttt{Result}(\{f_1\}), f_2\}), \ldots,$$

and this is the incremental solving method.

Now let's present the major processes of **ISBS**.

(i) We incrementally solve $\{f_1 + e_1, f_2 + e_2, \ldots, f_i + e_i\}$ for i from 1 to m with each $e_i = 0$. If $\texttt{Result}(\{f_1, f_2, \ldots, f_i\}) = \{1\}$ for some i, we flip e_i to 1 and continue solving the remaining polynomials based on $\texttt{Result}(\{f_1, f_2, \ldots, f_i + 1\})$. At last, we will obtain a candidate $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_m + e_m\})$ where $(e_1, \ldots, e_m)$ is equal to some fixed $(e'_1, \ldots, e'_m)$. We set u_{bound} to be the $u_0 - 1$, where u_0 is the Hamming weight of $(e'_1, \ldots, e'_m)$, and set the backtracking index k to be m.

(ii) In order to obtain a better candidate, we search all the possible values of $(e_1, \ldots, e_m)$ with backtracking based on the value $(e'_1, \ldots, e'_m)$. That is for i from k to 1 we find the first e'_i such that $e_i = 0$ and the Hamming weight of $(e_1, \ldots, e_{i-1}, 1)$ is less than u_{bound}, then similarly as step (i) we try to incrementally solve $f_{i+1}, \ldots, f_m$ based on $\texttt{Result}(\{f_1 + e'_1, \ldots, f_i + 1\})$. If we find a better candidate $\texttt{Result}(\{f_1 + e'_1, f_2 + e'_2, \ldots, f_i + 1, f_{i+1} + e''_{i+1}, \ldots, f_m + e''_m\})$, such that u, the Hamming weight of $(e'_1, e'_2, \ldots, 1, e''_{i+1}, \ldots, e''_m)$, is not bigger than u_{bound}, then we set k to be m, replace $(e'_1, \ldots, e'_m)$ with $(e'_1, \ldots, e'_{i-1}, 1, e''_{i+1}, \ldots, e''_m)$, replace u_{bound} with u, and do step (ii) again. Otherwise, set $k = i - 1$ and do step (ii) again if $k > 0$.

(iii) Finally, we have searched all the possible $(e_1, \ldots, e_m)$ and obtain the optimal solution.

In the Algorithm 1, we present the steps of **ISBS** method specifically. In this algorithm, a vector $(e_1, e_2, \ldots, e_s) \in \mathbb{F}_2^s$ with $s \leq m$ is called a *noise vector*. Moreover, we call the Hamming weight of a noise vector a *noise weight* and a bound of the noise weight a *noise-bound*. For the proof of the correctness and termination of **ISBS**, the reader is referred to [11].

[1] Note that SAT-solvers are not suitable for incremental solving, since it cannot represent a lot of solutions with a simple form.

Algorithm 1: ISBS algorithm.

```
input  : A polynomial system $\mathbf{P} = \{f_1, f_2, \dots, f_m\}$.
output: $(x_1, \dots, x_n) \in \mathbb{F}_2^n$ s.t. $\{i \in N | f_i(x_1, \dots, x_n) = 0, f_i \in \mathbf{P}\}$ has maximal cardinality.
1  Compute Candidate($\mathbf{P}, \emptyset, m, 0$);
2  Let $t$, $u$, $\{\mathbf{Q}_0, \mathbf{Q}_1, \dots, \mathbf{Q}_m\}$ and $E = (e_1, e_2, \dots, e_m)$ be the corresponding outputs
   of Candidate($\mathbf{P}, \emptyset, m, 0$);                          /* Here $t$ is always equal to $m$ */
3  $u_{bound} \leftarrow u - 1$, $\mathbf{S} \leftarrow \mathbf{Q}_m$;
4  Let $\mathbf{S}_1$ be the output of Backtracking($\mathbf{P}, E, \{\mathbf{Q}_0, \mathbf{Q}_1, \dots, \mathbf{Q}_m\}, u_{bound}, u$);
5  if $\mathbf{S}_1 \neq \emptyset$ then $\mathbf{S} \leftarrow \mathbf{S}_1$;
6  Get $(x_1, \dots, x_n)$ from $\mathbf{S}$ and return $(x_1, \dots, x_n)$;
```

Function: Candidate.

```
input  : $\mathbf{P} = \{f_{s+1}, f_{s+2}, \dots, f_m\}$: a polynomial set,
         Result($\mathbf{R}$) : the result set of a polynomial set $\mathbf{R}$,
         $u_{bound}$: a noise-bound,
         $u$: the noise weight of $(e_1, \dots, e_s)$.
output: $t$: an index number,
         $u$: the noise weight of $(e_1, \dots, e_{s+t})$,
         $\mathcal{QS} = \{\mathbf{Q}_s, \mathbf{Q}_{s+1}, \dots, \mathbf{Q}_{s+t}\}$: a sequence of result sets,
         $E = (e_{s+1}, e_{s+2}, \dots, e_{s+t})$.
1  $\mathbf{Q}_s \leftarrow$ Result($\mathbf{R}$), $t \leftarrow m - s$;
2  for $i$ from $s + 1$ to $m$ do
3      $\mathbf{P}_i \leftarrow \{\mathbf{Q}_{i-1}, f_i\}$, and compute Result($\mathbf{P}_i$);
4      if Result($\mathbf{P}_i$) $= \{1\}$ then        /* In this case, Result($\mathbf{Q}_{i-1}, f_i + 1$) $=$ Result($\mathbf{Q}_{i-1}$) */
5          $\mathbf{Q}_i \leftarrow \mathbf{Q}_{i-1}$, $e_i \leftarrow 1$, $u \leftarrow u + 1$;
6          if $u > u_{bound}$ then
7              $t \leftarrow i - s$, and break;
8      else if Result($\mathbf{P}_i$) and $\mathbf{Q}_{i-1}$ have the same zero set then        /* Result($\mathbf{Q}_{i-1}, f_i + 1$) $= \{1\}$ */
9          $\mathbf{Q}_i \leftarrow \mathbf{Q}_{i-1}$, $e_i \leftarrow 0$;
10     else
11         $\mathbf{Q}_i \leftarrow$ Result($\mathbf{P}_i$), $e_i \leftarrow 0$;
12 return $t$, $u$, $\{\mathbf{Q}_s, \mathbf{Q}_{s+1}, \dots, \mathbf{Q}_{s+t}\}$, $(e_{s+1}, e_{s+2}, \dots, e_{s+t})$;
```

Function: Backtracking.

```
input  : $\mathbf{P} = \{f_1, f_2, \dots, f_m\}$: a polynomial set,
         $E = (e_1, e_2, \dots, e_m)$: a noise vector,
         $\{\mathbf{Q}_0, \mathbf{Q}_1, \dots, \mathbf{Q}_m\}$: a sequence of result sets
         $u_{bound}$: a noise-bound;
         $u$: the noise weight of $E$;
output: A result set $\mathbf{S}$.
1  $k \leftarrow m$, $\mathbf{S} \leftarrow \emptyset$;                          /* $k$ is the backtracking index */
2  while $k \geq 1$ do
3      if $e_k = 0$, $\mathbf{Q}_k \neq \mathbf{Q}_{k-1}$ and $u + 1 \leq u_{bound}$ then
4          $\mathbf{P}_k \leftarrow \{\mathbf{Q}_{k-1}, f_k + 1\}$ and compute Result($\mathbf{P}_k$);
5          $e_k \leftarrow 1$, $u \leftarrow u + 1$;
6          $\mathbf{Q}_k \leftarrow$ Result($\mathbf{P}_k$);
7          Set $t$, $u$, $\{\mathbf{Q}_k, \mathbf{Q}_{k+1}, \dots, \mathbf{Q}_{k+t}\}$, $(e_{k+1}, \dots, e_{k+t})$ to be the output of Candidate($\{f_{k+1}, f_{k+2}, \dots, f_m\}, \mathbf{Q}_k, u_{bound}, u$);
8          $k \leftarrow k + t$;
9          if $u \leq u_{bound}$ then                          /* In this case, $k = m$ */
10             $\mathbf{S} \leftarrow \mathbf{Q}_m$, $u_{bound} \leftarrow u - 1$;
11     else
12         $u \leftarrow u - e_k$, $k \leftarrow k - 1$;
13 return $\mathbf{S}$.
```

Remark 1. This version of **ISBS** is slightly different from the version in [11]. We add the comparisons of the zero sets of Result($\mathbf{P}_i$) and $\mathbf{Q}_{i-1}$ (Step 8) in Candidate. When this condition is true, we always have Result($\mathbf{Q}_{i-1}, f_i + 1$) $= \{1\}$. Thus, in this case we don't need to check whether Result($\mathbf{Q}_{i-1}, f_i + 1$) is equal to $\{1\}$ in Backtracking (that is what the algorithm in [11] did). Therefore, we add a condition $\mathbf{Q}_k \neq \mathbf{Q}_{k-1}$ in Step 3 of `Backtracking` to verify whether this case happens.

Note that checking whether two result sets `Result`($\mathbf{P}_i$) and $\mathbf{Q}_{i-1}$ have the same zero set is very easy, thus the cost is much less than that of solving $\{\mathbf{Q}_{i-1}, f_i + 1\}$. For example, when we use the characteristic set method as the incremental solving tool, the number of solutions in the result set can be counted easily, thus we only need to check whether `Result`($\mathbf{P}_i$) and $\mathbf{Q}_{i-1}$ have the same number of solutions.

3. The search tree of ISBS

It is easy to see that, **ISBS** can be efficient only when the cost of computing `Result` is small for the input system, which means incremental solving process can be executed fast. In this case, a major factor that determines the complexity of the algorithm is the size of the backtracking search tree. In this section, we will present theoretical analysis of the number of the branches in this tree, and show how the properties of the input systems effect this number.

First of all, we introduce some notations and terminologies about the binary trees used in this paper. For a binary tree, when we say a path in this tree, we mean a node sequence $N_0, N_1, \dots, N_k$, where N_0 is the root node and N_i is a child node of N_{i-1}. Moreover, we denote this path by $N_0 \to N_1 \to \cdots \to N_k$. A path from the root node to a leaf node is called a *branch* of the tree. For a binary tree $\mathbb{T}$, the number of its branches is denoted by $|\mathbb{T}|$. For two binary trees $\mathbb{T}_1$ and $\mathbb{T}_2$, their intersection, denoted by $\mathbb{T}_1 \cap \mathbb{T}_2$, is the tree whose nodes are the common nodes of these two trees.

Now we strictly define *the search tree* of **ISBS**. This binary tree can be generated as the following procedures. First, let the root node of the tree be the empty set, and use a pointer $\mathcal{M}$ pointing to the root node. Then run the algorithm and generate the new nodes by the following operations.

- In `Candidate`, after each time we set the value of some e_i, we generate a new node $f_i + e_i$, and draw an edge from the node pointed by $\mathcal{M}$ to this new node, then let $\mathcal{M}$ point to this new node.
- In `Backtracking`, after each time we decrease the backtracking index k by 1, we let $\mathcal{M}$ point to the parent node of the node pointed by $\mathcal{M}$ currently. Besides, after we set the value of some e_i, we generate a new node $f_i + e_i$, and draw an edge from the node pointed by $\mathcal{M}$ to this new node, then let $\mathcal{M}$ point to this new node.

In this way, we can generate a binary tree, denoted by $\mathbb{T}_{ISBS}$, with depth m (the depth of the root node is set to be 0) after running **ISBS**. Note that, in **ISBS**, e_i is evaluated if and only if `Result` has been computed for one time. Therefore, the number of nodes in $\mathbb{T}_{ISBS}$ is equal to the number of computing `Result`. It is easy to see that a path $f_1 + e_1 \to f_2 + e_2 \to \cdots \to f_k + e_k$ with length k in $\mathbb{T}_{ISBS}$ is one-to-one corresponding to the process of incrementally solving the polynomial system $\{f_1 + e_1, f_2 + e_2, \dots, f_k + e_k\}$ in **ISBS**.

Since the depth of $\mathbb{T}_{ISBS}$ is m, $|\mathbb{T}_{ISBS}|$ is roughly bounded by 2^m. However, this search tree is not a perfect binary tree, since a lot of subtrees are pruned in the following four cases:

(1) `Result`($\mathbf{P}_i$) $= \{1\}$ in Step 4 of `Candidate`.
(2) `Result`($\mathbf{P}_i$) and $\mathbf{Q}_{i-1}$ have the same zero set in Step 8 of `Candidate`.
(3) $u > u_{bound}$ in Step 6 of `Candidate`.
(4) $u + 1 > u_{bound}$ in Step 3 of `Backtracking`.

From the above four cases, we can conclude two major factors which influence the size of the search tree. Cases (1) and (2) are corresponding to the first one, which is the randomness of the input system. Cases (3) and (4) are corresponding to the second one, which is the value of the noise-bound.

First, we analyze the effect of the first factor. To this end, we will consider a bigger tree, which is generated from **ISBS** with the following modifications. That is we don't compare the noise weight with u_{bound} in `Candidate` and `Backtracking`. Instead, we achieve the solutions of the polynomial systems with all possible noise weight, and finally output the solution of the system whose noise weight is lowest. We denote this modified algorithm by **ISBS**$_1$. We can generate a search tree of **ISBS**$_1$ similar as $\mathbb{T}_{ISBS}$, and we call this tree the *quasi search tree* of **ISBS**, and denote it as $\mathbb{T}_{quasi}$. It is easy to see that for each path p in $\mathbb{T}_{ISBS}$ with length k, we can find a path p' in $\mathbb{T}_{quasi}$ with length at least k, such that the first k nodes of p and p' are the same. This means $\mathbb{T}_{ISBS}$ is a truncated subtree of $\mathbb{T}_{quasi}$. We have the following proposition about $\mathbb{T}_{quasi}$.

Proposition 3. *Let $f_1, f_2, \dots, f_m$ be the input of* **ISBS**. *We denote the number of paths with length k in $\mathbb{T}_{quasi}$ by $\mathcal{N}_k$, for $1 \le k \le m$. Moreover, we define a map $\mathcal{S}_k : \mathbb{F}_2^n \to \mathbb{F}_2^k$, with $\mathcal{S}_k(x) = (f_1(x), f_2(x), \dots, f_k(x))$. Then, we have*

(a) *All the branches in $\mathbb{T}_{quasi}$ have depth m.*
(b) *$\mathcal{N}_k = |Im(\mathcal{S}_k)|$, for any $1 \le k \le m$.*

Proof. (a) Note that a branch ends after `Candidate` terminated, which is equivalent to the loop of Step 2 terminated. This loop terminates before $i = m$, only when $u > u_{bound}$ in Step 6. However, for $\mathbb{T}_{quasi}$, the noise-bound is not used, hence `Candidate` terminates after it has dealt with f_m, which implies all branches in $\mathbb{T}_{quasi}$ have length m.

(b) For a path $p : f_1 + e_1 \to f_2 + e_2 \to \cdots \to f_k + e_k$, suppose we have $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k\}) = \{1\}$. Then we can find a constant $s \leq k$ such that $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_s + e_s\}) = \{1\}$ and $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_{s-1} + e_{s-1}\}) \neq \{1\}$.

- If $e_s = 0$, then the condition in Step 4 of `candidate` holds, we will set e_i to be 1, which means we will generate the node $f_s + 1$ instead of the node f_s, then the path p is pruned at the node f_s, a contradiction.
- If $e_s = 1$, then $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_{s-1} + e_{s-1}\}) = \texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_{s-1} + e_{s-1}, f_s\})$. Obviously, node $f_s + 1$ can only be generated in Step 5 of `Backtracking`. However, since $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_{s-1} + e_{s-1}) = \texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_{s-1} + e_{s-1}, f_s\})$, the condition in Step 3 of `Backtracking` doesn't hold, which means Step 5 will not be executed, a contradiction.

Therefore, we have $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k\}) \neq \{1\}$ which means $(e_1, e_2, \ldots, e_k) \in \mathrm{Im}(\mathcal{S}_k)$. Then, we can build a map $\mathbb{M}$ which maps a path $f_1 + e_1 \to f_2 + e_2 \to \cdots \to f_k + e_k$ to $(e_1, e_2, \ldots, e_k) \in \mathrm{Im}(\mathcal{S}_k)$. Obviously, $\mathbb{M}$ is an injection. For any $(e_1, e_2, \ldots, e_k) \in \mathrm{Im}(\mathcal{S}_k)$, from the definition of $\mathcal{S}_k$, we know that $f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k$ have common solutions, which means $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k\}) \neq \{1\}$. It is easy to see that the path $f_1 + e_1 \to f_2 + e_2 \to \cdots \to f_k + e_k$ is in $\mathbb{T}_{quasi}$, hence $\mathbb{M}$ is a surjection. In summary, $\mathbb{M}$ is a bijection, thus $\mathcal{N}_k = |\mathrm{Im}(\mathcal{S}_k)|$. □

The above proposition shows that $|\mathbb{T}_{quasi}|$ is equal to $|\mathrm{Im}(\mathcal{S}_m)|$, which is bounded by 2^n. $|\mathrm{Im}(\mathcal{S}_m)|$ is an essential property of a polynomial system, and changing the order of f_i will not change the value of $|\mathrm{Im}(\mathcal{S}_m)|$, hence not change $|\mathbb{T}_{quasi}|$. The maximal value of $|\mathrm{Im}(\mathcal{S}_m)|$ is 2^n, which can be reached when the randomness of the input systems is good enough. The above analysis implies that in **ISBS**, we can prune all branches which are redundant because of the bad randomness of the input system.

Remark 2. For **ISBS**$_1$, from Proposition 3, we can deduce that the number of computing `Result` is equal to $\sum_{i=1}^{m} |\mathrm{Im}(\mathcal{S}_i)|$, where $|\mathrm{Im}(\mathcal{S}_1)| \leq |\mathrm{Im}(\mathcal{S}_2)| \leq \cdots \leq |\mathrm{Im}(\mathcal{S}_m)|$. Obviously, changing the order of f_i will change the value of $|\mathrm{Im}(\mathcal{S}_i)|$, for $1 \leq i \leq m - 1$. Unfortunately, it is too hard to estimate the value of $|\mathrm{Im}(\mathcal{S}_i)|$, hence we cannot theoretically find a way of sorting f_i such that $\sum_{i=1}^{m} |\mathrm{Im}(\mathcal{S}_i)|$ reaches its minimal value. However, the complexity of incrementally solving a polynomial system is highly relevant to the order of the polynomials, which means the order of f_i can effect the complexity of computing `Result`, hence we can sort f_i such that the incremental solving processes are more efficient. A natural idea of sorting f_i is putting the "easy" ones before the "hard" ones. Here the "hardness" of a polynomial can be defined by different indexes for different polynomial system solving methods.

Now we consider the influence of the noise-bound. Given a noise-bound u_{bound}, we construct a binary tree $\mathbb{T}_{nw}$ from a perfect binary tree $\mathbb{T}$ with depth m. That is in $\mathbb{T}$ we keep the branches $f_1 + e_1 \to f_2 + e_2 \to \cdots \to f_m + e_m$, such that $e_1 + e_2 + \cdots + e_m \leq u_{bound}$, and pruned other branches. Obviously, we have $|\mathbb{T}_{nw}| = \sum_{i=0}^{u} \binom{m}{i}$. Then, we have the following theorem.

Theorem 4. *Assume* **ISBS** *is executed under noise-bound u_{bound}, and this bound is not changed after* **ISBS** *terminated. Suppose $\mathbb{T}_0 = \mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$, and $\mathbb{T}_1$ is a binary tree generated from $\mathbb{T}_0$ by appending a child node $f_{k+1} + 1$ to the end of each branch with depth $k < m$ in $\mathbb{T}_0$. Then we have:*

(1) $\mathbb{T}_{ISBS} = \mathbb{T}_1$.
(2) $|\mathbb{T}_{ISBS}| = |\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}|$.

Proof. (1) First we prove $\mathbb{T}_{ISBS} \subset \mathbb{T}_1$. Note that in `Candidate`, in any one of the three cases corresponding to Step 4, Step 8 and Step 10, e_{s+1} will always be evaluated, which means at least one new node will be generated in $\mathbb{T}_{ISBS}$.

- Consider a branch $p : f_1 + e_1 \to f_2 + e_2 \to \cdots \to f_k + e_k$ with depth $k < m$ in $\mathbb{T}_{ISBS}$. If $e_k = 0$, the value of e_k must be set in `Candidate` at Step 9 or 11. It means that `Candidate` doesn't end after evaluating e_k, and either node $f_{k+1} + 1$ or node f_k will be generated in the following steps, which contradicts to the assumption that $f_k + e_k$ is a leaf node. Therefore we have $e_k = 1$. There are two cases that e_k is set to be 1. The first case is that in `Backtracking` we backtrack to index k and flip the value of e_k from 0 to 1. In this case, the condition of Step 3 holds, which means $e_1 + e_2 + \cdots + e_k \leq u_{bound}$, then `Candidate` will be executed and a child node will be generated, a contradiction. Hence, e_k is set to be 1 in `Candidate` when $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_k\})$ is equal to $\{1\}$. Then, we have $e_1 + e_2 + \cdots + e_k > u_{bound}$, otherwise, `Candidate` will not terminate and a child node will be generated. Obviously, $e_1 + e_2 + \cdots + e_{k-1} \leq u_{bound}$. We can deduce $e_1 + e_2 + \cdots + e_{k-1} = u_{bound}$. Hence path $p' : f_1 + e_1 \to f_2 + e_2 \to \cdots \to f_{k-1} + e_{k-1}$ is in $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$. Note that $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_{k-1} + e_{k-1}, f_k\}) = \{1\}$, which means path $f_1 + e_1 \to f_2 + e_2 \to$

$\cdots \to f_{k-1}+e_{k-1} \to f_k$ is not in $\mathbb{T}_{quasi}$, and path $f_1+e_1 \to f_2+e_2 \to \cdots \to f_{k-1}+e_{k-1} \to f_k+1$ is in $\mathbb{T}_{quasi}$. Therefore, in $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$, node $f_{k-1}+e_{k=1}$ of p' is a leaf node is a branch which implies p' is a branch with depth less than m. Therefore, p is in $\mathbb{T}_1$.

- Consider a branch $p: f_1+e_1 \to f_2+e_2 \to \ldots \to f_m+e_m$ in $\mathbb{T}_{ISBS}$. If $e_1+e_2+\cdots+e_m \le u_{bound}$, then it is obvious that $p \in \mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$. Since the depth of p is m, we have $p \in \mathbb{T}_1$. Note that, there is a case that $e_1+e_2+\cdots+e_m > u_{bound}$. That is e_m is set to be 1 in `Candidate`. Similarly as the proof above, we have $e_1+e_2+\cdots+e_{m-1} = u_{bound}$ and `Result`$(\{f_1+e_1, f_2+e_2, \ldots, f_{m-1}+e_{m-1}, f_m\}) = \{1\}$, which means $f_1+e_1 \to f_2+e_2 \to \ldots \to f_{m-1}+e_{m-1}$ is a branch of $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$, thus $p \in \mathbb{T}_1$.

Now we prove $\mathbb{T}_1 \subset \mathbb{T}_{ISBS}$. Let $p: f_1+e_1 \to f_2+e_2 \to \cdots \to f_k+e_k$ be a branch in $\mathbb{T}_1$.

- If $e_k = 0$, we have $k = m$, since the branches in $\mathbb{T}_1$ with depth less than m have a leaf node with the form f_k+1. Then $p \in \mathbb{T}_{quasi}$ and $e_1+e_2+\cdots+e_k \le u_{bound}$, hence $p \in \mathbb{T}_{ISBS}$.
- Consider the case of $e_k = 1$. If $f_1+e_1 \to f_2+e_2 \to \cdots \to f_{k-1}+e_{k-1}$ is not a branch in $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$, then p is in $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$, hence in $\mathbb{T}_{ISBS}$. Assume $p': f_1+e_1 \to f_2+e_2 \to \cdots \to f_{k-1}+e_{k-1}$ is a branch in $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$, hence is a path in $\mathbb{T}_{ISBS}$. If $e_1+e_2+\cdots+e_{k-1} < u_{bound}$, then $e_1+e_2+\cdots+e_{k-1}+1 \le u_{bound}$. Since either `Result`$(\{f_1+e_1, f_2+e_2, \ldots, f_{k-1}+e_{k-1}, f_k\}) \ne \{1\}$ or `Result`$(\{f_1+e_1, f_2+e_2, \ldots, f_{k-1}+e_{k-1}, f_k+1\}) \ne \{1\}$, we have either path $f_1+e_1 \to f_2+e_2 \to \cdots \to f_{k-1}+e_{k-1} \to f_k$ or path $f_1+e_1 \to f_2+e_2 \to \cdots \to f_{k-1}+e_{k-1} \to f_k+1$ is in $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$, which contradicts to the assumption that $f_{k-1}+e_{k-1}$ is a leaf node in $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$. This implies $e_1+e_2+\cdots+e_{k-1} = u_{bound}$. Note that path $p': f_1+e_1 \to f_2+e_2 \to \cdots \to f_{k-1}+e_k \to f_k$ is not in $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$. From $e_1+e_2+\cdots+e_k+0 = u_{bound}$, we can deduce that p' is not in $\mathbb{T}_{quasi}$, which means `Result`$(\{f_1+e_1, f_2+e_2, \ldots, f_{k-1}+e_{k-1}, f_k\}) = \{1\}$. Thus, in `Candidate` of **ISBS**, after computing `Result`$(\{f_1+e_1, f_2+e_2, \ldots, f_{k-1}+e_{k-1}, f_k\})$, e_k will be set to be 1, and a leaf node f_k+1 is generated and appended to the end of p' in $\mathbb{T}_{ISBS}$. Therefore, p is in $\mathbb{T}_{ISBS}$.

(2) It is easy to see that $|\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}| = |\mathbb{T}_1|$. Thus, from (1), we have $|\mathbb{T}_{ISBS}| = |\mathbb{T}_1| = |\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}|$. □

The above proof can be illustrated by the following figures. We see $\mathbb{T}_{ISBS}$ as a tree generated from $\mathbb{T}_{quasi}$ by the following operations. Consider a path $p: f_1+e_1 \to f_2+e_2 \to \cdots \to f_{k-1}+e_{k-1} \to f_k+1$ in $\mathbb{T}_{quasi}$, such that $\sum_{i=1}^{k-1} e_i = u_{bound}$.

(a) If $f_{k-1}+e_{k-1}$ has two child nodes in $\mathbb{T}_{quasi}$, we pruned the subtree with f_k+1 as the root node. This case can be illustrated in Fig. 1, where $\bar{f}_i$ means f_i+1.

(b) If $f_{k-1}+e_{k-1}$ has a unique child node in $\mathbb{T}_{quasi}$, we pruned the subtrees whose root nodes are the child nodes of f_k+1. This case can be illustrated in Fig. 2.

In comparison, $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$ can also be seen as a binary tree generated from $\mathbb{T}_{quasi}$ by the pruning operations when the above cases (a) and (b) occur. For case (a), the pruning operations are the same as $\mathbb{T}_{ISBS}$. For the case (b), we prune the subtree whose root node is f_k+1. These two kinds of operations can be illustrated in Figs. 3 and 4.

From Theorem 4, we can know that since $|\mathbb{T}_{quasi}|$ is fixed for a given input system, a principle way to decrease $|\mathbb{T}_{ISBS}|$, is decreasing $|\mathbb{T}_{nw}|$. Moreover, when m is fixed, the only way to decrease $|\mathbb{T}_{nw}|$ is decreasing u_{bound}. If the noise weight of the optimal solution is u_0, then **ISBS** can obtain this solution under noise-bound u_0, and in this case $|\mathbb{T}_{nw}|$ reaches its minimal value.

Therefore, a natural way to improve **ISBS** is constructing a small artificial noise-bound and then executing **ISBS** under this bound. If there is no solution under this noise-bound, we can gradually increase the artificial bound by a step size s until we find the optimal solution. In this way, we can keep the noise-bound and $|\mathbb{T}_{nw}|$ as small as possible. According to this idea, we present a modified algorithm **ISBS**$_b$.

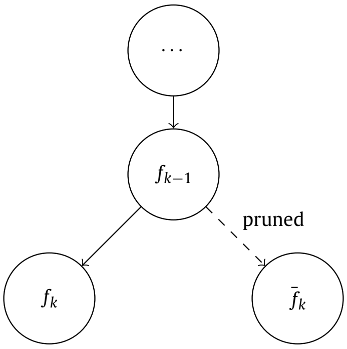

Fig. 1. Case (a) of $\mathbb{T}_{ISBS}$.

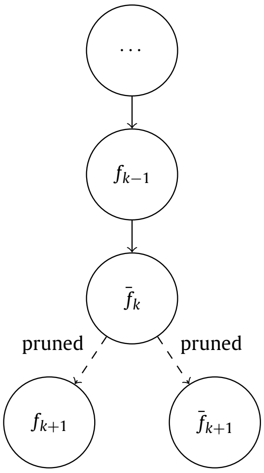

Fig. 2. Case (b) of $\mathbb{T}_{ISBS}$.

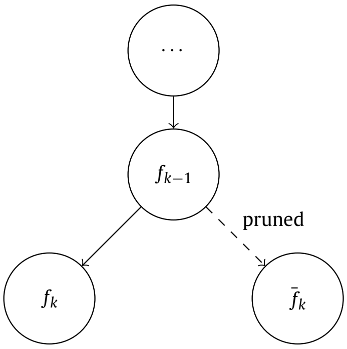

Fig. 3. Case (a) of $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$.

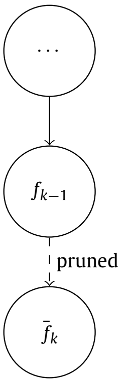

Fig. 4. Case (b) of $\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}$.

Algorithm 2: ISBS$_b$ algorithm.

```
input  : A polynomial system $\mathbf{P} = \{f_1, f_2, \dots, f_m\}$;
         A step size $s$.
output: $(x_1, \dots, x_n) \in \mathbb{F}_2^n$ s.t. $\{i \in N | f_i(x_1, \dots, x_n) = 0, f_i \in \mathbf{P}\}$ has maximal cardinality.

Compute Candidate($\mathbf{P}, \emptyset, m, 0$);
Let $t$, $u_m$, $\{\mathbf{Q}_0, \mathbf{Q}_1, \dots, \mathbf{Q}_m\}$ and $E = (e_1, e_2, \dots, e_m)$ be the corresponding output
of Candidate($\mathbf{P}, \emptyset, m, 0$);                    /* Here $t$ is always equal to $m$ */
$u_{bound} \leftarrow 0$, $\mathbf{S} \leftarrow \mathbf{Q}_m$;
while $u_{bound} < u_m - 1$ do
    $u_{bound} = \min(u_{bound} + s, u_m - 1)$;
    Let $\mathbf{S}_1$ be the output of Backtracking($\mathbf{P}, E, \{\mathbf{Q}_0, \mathbf{Q}_1, \dots, \mathbf{Q}_m\}, u_{bound}, u_m$);
    if $\mathbf{S}_1 \neq \emptyset$ then $\mathbf{S} \leftarrow \mathbf{S}_1$ and break
Get $(x_1, \dots, x_n)$ from $\mathbf{S}$ and return $(x_1, \dots, x_n)$;
```

4. Solving Max-PoSSo when $m = sn$

In this section, we will discuss to solve the Max-PoSSo problems when the input polynomial systems satisfying $m = sn$, where $s > 1$ is an integer. This kind of polynomial systems frequently occur in cryptanalysis problems. For example, for the Cold Boot key recovery problem introduced in Section 5, we can generate sn polynomials for any integer s. According to the results in Section 2.1, in most cases, in order to keep a high success rate we need to set $s > 1$.

The original idea of the approach proposed in this section was roughly introduced in [11] for the case $s = 2$. However, why it works and how to optimize the idea were not shown. Thus, in this section we will discuss these problems by comparing the number of branches in the search trees of different approaches.

From Proposition 3, we know that $|\mathbb{T}_{quasi}| = |\mathrm{Im}(\mathcal{S}_m)|$. From our observation, for most polynomial systems generated from cryptanalysis, we have $|\mathrm{Im}(\mathcal{S}_n)| \approx |\mathrm{Im}(\mathcal{S}_m)|$. Hence $|\mathbb{T}_{quasi}| \approx |\mathrm{Im}(\mathcal{S}_n)|$. For a binary tree $\mathbb{T}$, denote its truncated subtree with depth n by $\bar{\mathbb{T}}$. Then, from $|\mathbb{T}_{quasi}| \approx |\mathrm{Im}(\mathcal{S}_n)| = |\bar{\mathbb{T}}_{quasi}|$, we can deduce that $|\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}| \approx |\bar{\mathbb{T}}_{quasi} \cap \mathbb{T}_{nw}|$. Moreover, it is obvious that $\bar{\mathbb{T}}_{quasi} \cap \mathbb{T}_{nw} = \bar{\mathbb{T}}_{quasi} \cap \bar{\mathbb{T}}_{nw}$, thus $|\mathbb{T}_{ISBS}| = |\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}| \approx |\bar{\mathbb{T}}_{quasi} \cap \bar{\mathbb{T}}_{nw}|$. Consequently, if we can decrease $|\bar{\mathbb{T}}_{nw}|$, $|\mathbb{T}_{ISBS}|$ will decrease. In the following, we will discuss how to decrease $|\bar{\mathbb{T}}_{nw}|$.

4.1. Case $m = 2n$

In this subsection, we suppose the input system is a polynomial system $\{f_1, f_2, \ldots, f_m\}$ with $m = 2n$. First we introduce the following lemma.

Lemma 5. *Let u_1, u_2, u be three non-negative integers, and $u_1 + u_2 \leq u$. For any non-negative integers a, b, such that $a + b = u - 1$, we have either $u_1 \leq a$ or $u_2 \leq b$.*

Proof. Suppose $u_1 > a$ and $u_2 > b$. Since u_1, u_2 are integers, we have $u_1 \geq a + 1$ and $u_2 \geq b + 1$. Then $u \geq u_1 + u_2 \geq a + b + 2 = u + 1$, which is a contradiction. □

This lemma shows the following fact. We can divide a noise vector $E = (e_1, \ldots, e_m)$ into two parts $E_1 = (e_1, \ldots, e_n)$, $E_2 = (e_{n+1}, \ldots, e_m)$. Suppose E has a noise-bound u. Let a, b be two non-negative integers such that $a + b = u - 1$, then either E_1 has a noise-bound a or E_2 has a noise-bound b.

We call the noise-bound of E_1 (or E_2) a partial noise-bound of E. For a noise vector $(e_1, \ldots, e_s)$, we define its *partial noise weight* to be the noise weight of $(e_1, \ldots, e_n)$ when $s > n$, or the noise weight of itself when $s \leq n$. Then we can build the following *two-direction approach*.

First, we solve the system from the forward direction, which means we find the optimal solution of the system $\mathbf{P}_1 : \{f_1, f_2, \ldots, f_n, f_{n+1}, \ldots, f_m\}$ under the conditions:

- The partial noise weight of $(e_1, e_2, \ldots, e_m)$ is bounded by the partial noise-bound a;
- The noise weight of $(e_1, e_2, \ldots, e_m)$ is bounded by the noise-bound u.

This can be done by a modified **ISBS** algorithm in which the comparison of the partial noise-bound and the partial noise-weight is added in Step 6 of `Candidate`, and Step 3 and Step 9 of `Backtracking`.

Then, we solve the system from the backward direction, which means we find the optimal solution of the system $\mathbf{P}_2 : \{f_{n+1}, f_{n+2}, \ldots, f_m, f_1, \ldots, f_n\}$ under the conditions:

- The partial noise weight of $(e_1, e_2, \ldots, e_m)$ is bounded by the partial noise-bound b;
- The noise weight of $(e_1, e_2, \ldots, e_m)$ is bounded by the noise-bound u.

Finally, the better one of the solutions of the above two systems is the solution of the original Max-PoSSo problem.

Similarly as the original **ISBS**, for $\mathbf{P}_1$ or $\mathbf{P}_2$, we can generate the search tree $\mathbb{T}_{ISBS}$ after the above solving processes ended. Moreover, $\mathbb{T}_{quasi}$ is the tree generated when the noise-bound and the partial noise-bound are not used. $\mathbb{T}_{nw}$ is the tree generated from a perfect binary tree by pruning the branches which don't satisfy the constraints about the noise-bound and the partial noise-bound. It is easy to see that when solving from the forward direction, we have $|\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{a} \binom{n}{i} < \sum_{i=0}^{u} \binom{n}{i}$, while when solving from the backward direction, we have $|\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{b} \binom{n}{i} < \sum_{i=0}^{u} \binom{n}{i}$. Note that, for the original approach, we have $|\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{u} \binom{n}{i}$. It means that, by the two-direction approach, we convert the original Max-PoSSo problem into two easier subproblems.

There are two natural questions about the two-direction approach:

i) Whether the total number of branches in the two search trees of the two directions is always less than the number of branches in the search tree of the original approach.

ii) How to choose a and b such that the total number of branches in the two search trees of the two-direction approach is minimal.

In the following paragraphs of this subsection, we will show that these two questions can be perfectly answered when the input systems $\{f_1, f_2, \ldots, f_m\}$ satisfying the condition that $|\mathrm{Im}(\mathcal{S}_n)| = |\mathrm{Im}(\mathcal{S}'_n)| = 2^n$, where $\mathcal{S}_n$ is the map which maps $x \in \mathbb{F}_2^n$ to $(f_1(x), f_2(x), \ldots, f_n(x)) \in \mathbb{F}_2^n$, and $\mathcal{S}'_n$ is the map which maps $x \in \mathbb{F}_2^n$ to $(f_{n+1}(x), f_{n+2}(x), \ldots, f_m(x)) \in \mathbb{F}_2^n$.

When $|\mathrm{Im}(\mathcal{S}_n)| = 2^n$, for $\mathbf{P}_1$, we have $|\bar{\mathbb{T}}_{quasi}| = 2^n$, thus $\bar{\mathbb{T}}_{quasi}$ is a perfect binary tree. Therefore, $\bar{\mathbb{T}}_{quasi} \cap \bar{\mathbb{T}}_{nw} = \bar{\mathbb{T}}_{nw}$, and $|\bar{\mathbb{T}}_{quasi} \cap \bar{\mathbb{T}}_{nw}| = |\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{a} \binom{n}{i}$. Moreover, $|\bar{\mathbb{T}}_{quasi}| \le |\mathbb{T}_{quasi}| \le 2^n$. Thus, for any leaf node N of $|\bar{\mathbb{T}}_{quasi}|$, if it is the root node of a subtree in $\mathbb{T}_{quasi}$, then this subtree has a unique branch. Hence, for the nodes in $\mathbb{T}_{quasi}$ with depth bigger than n, the pruning case (a) in Fig. 1 of Section 3 will not occur. This means $|\mathbb{T}_{ISBS}| = |\bar{\mathbb{T}}_{ISBS}| = |\bar{\mathbb{T}}_{quasi} \cap \bar{\mathbb{T}}_{nw}| = |\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{a} \binom{n}{i}$. Similarly, for $\mathbf{P}_2$, we have $|\mathbb{T}_{ISBS}| = |\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{b} \binom{n}{i}$.

Proposition 6. *Let a, b, u, n be non-negative integers, such that $u \le n/2$ and $a + b = u - 1$. Then we have:*

$$\sum_{i=0}^{u} \binom{n}{i} > \sum_{i=0}^{a} \binom{n}{i} + \sum_{i=0}^{b} \binom{n}{i} \ge \sum_{i=0}^{\lfloor \frac{a+b}{2} \rfloor} \binom{n}{i} + \sum_{i=0}^{\lceil \frac{a+b}{2} \rceil} \binom{n}{i}.$$

Proof. Without loss of generality, we can assume $a \le b$. First we prove $\sum\limits_{i=0}^{u} \binom{n}{i} > \sum\limits_{i=0}^{a} \binom{n}{i} + \sum\limits_{i=0}^{b} \binom{n}{i}$. It is equivalent to prove $\sum\limits_{i=a+1}^{u} \binom{n}{i} > \sum\limits_{i=0}^{b} \binom{n}{i}$. Note that the numbers of terms in two sides of the inequality are both $b + 1$. Moreover, since $n/2 \ge u \ge a + 1 > 0$, we have $\binom{n}{a+1} > \binom{n}{0}$, $\binom{n}{a+2} > \binom{n}{1}$, $\binom{n}{u} > \binom{n}{b}$. By summing up all these inequalities, we have $\sum\limits_{i=a+1}^{u} \binom{n}{i} > \sum\limits_{i=0}^{b} \binom{n}{i}$.

Now we prove the second inequality. Obviously, when $a = \lfloor \frac{a+b}{2} \rfloor$, which means that either u is odd and $a = b$ or u is even and $a = b - 1$, the equality holds. Without loss of generality, we can assume $a < \lfloor \frac{a+b}{2} \rfloor$, which implies $a < \lfloor \frac{a+b}{2} \rfloor \le \lceil \frac{a+b}{2} \rceil < b$. Then it is sufficient to show that

$$\binom{n}{a+1} + \binom{n}{a+2} + \cdots + \binom{n}{\lfloor \frac{a+b}{2} \rfloor} < \binom{n}{\lceil \frac{a+b}{2} \rceil + 1} + \binom{n}{\lceil \frac{a+b}{2} \rceil + 2} + \cdots + \binom{n}{b}.$$

Since $n/2 \ge u \ge b$, we have the numbers of terms in both sides of the above inequality are same, and $\binom{n}{a+1} < \binom{n}{\lceil \frac{a+b}{2} \rceil + 1}$, $\binom{n}{a+2} < \binom{n}{\lceil \frac{a+b}{2} \rceil + 2}$, $\ldots$, $\binom{n}{\lfloor \frac{a+b}{2} \rfloor} < \binom{n}{b}$. Therefore, we can deduce the conclusion by summing up all these inequalities. □

Given a noise-bound $u \le n/2$.[2] Proposition 6 shows that if $|\mathrm{Im}(\mathcal{S}_n)| = |\mathrm{Im}(\mathcal{S}'_n)| = 2^n$, the branches we need to solve in the two-direction approach is strictly less than those in the original approach. Moreover, we can conclude that the optimal strategy of the two-direction approach is using the partial noise-bound $\lfloor \frac{a+b}{2} \rfloor$ for one system and $\lceil \frac{a+b}{2} \rceil$ for another system. It is easy to see that when n is big, the number of branches solved in the optimal two-direction approach is much less than that in the original approach. For example, let $n = 128$, $u = 10$. If $|\mathrm{Im}(\mathcal{S}_n)| = |\mathrm{Im}(\mathcal{S}'_n)| = 2^n$, for the original approach, we need to solve $\sum_{i=0}^{10} \binom{128}{i} \approx 2^{47.8}$ branches. For the optimal two-direction approach, we need to solve $\sum_{i=0}^{4} \binom{128}{i} + \sum_{i=0}^{5} \binom{128}{i} \approx 2^{28.1}$ branches. Obviously, this is a significant improvement.

By combining the two-direction approach and $\mathbf{ISBS}_b$, we propose an improved algorithm called $\mathbf{ISBS}_2$, whose mainly steps are as follows.

1. Given an input system $\mathbf{P} : \{f_1, \ldots, f_n, f_{n+1}, \ldots, f_{2n}\}$.
2. Generate two systems $\mathbf{P}_1 : \{f_1, \ldots, f_n, f_{n+1}, \ldots, f_{2n}\}$ and $\mathbf{P}_2 : \{f_{n+1}, \ldots, f_{2n}, f_1, \ldots, f_n\}$.
3. Set a total noise-bound u.
4. Set the partial noise-bound of $\mathbf{P}_1$ to be $\lfloor \frac{u-1}{2} \rfloor$ and the partial noise-bound of $\mathbf{P}_2$ to be $\lceil \frac{u-1}{2} \rceil$.
5. For i from 1 to 2, solve $\mathbf{P}_i$ with **ISBS** method under noise-bound u and its partial noise-bound.
   - If for some $\mathbf{P}_i$, we achieve a solution x, set x_0 to be x and u to be $w(x) - 1$, where $w(x)$ is the noise weight of the noise vector corresponding to x. If $i = 1$, update the partial noise-bounds of $\mathbf{P}_2$.
6. If x_0 is not empty, then it is the solution of the Max-PoSSo problem. Otherwise, increase u by a step size, and repeat Step 4–5.

For the practical Max-PoSSo problems, the condition $|\mathrm{Im}(\mathcal{S}_n)| = |\mathrm{Im}(\mathcal{S}'_n)| = 2^n$ may not hold, and theoretically finding the optimal values of a and b is too hard. From the experiments, we verified that respectively setting a, b to be $\lfloor \frac{u-1}{2} \rfloor$ and $\lceil \frac{u-1}{2} \rceil$ is still a good strategy, by which the timing results are much better than those of other strategies.

[2] For practical Max-PoSSo problems, $u \le n/2$ always holds.

4.2. The optimal division strategy

In the above two-direction approach, we divide the input system $\mathbf{P}$ with $2n$ elements into two sub-systems $\mathbf{P}_1$ and $\mathbf{P}_2$ with n elements, and then respectively solve $\{\mathbf{P}_1, \mathbf{P}_2\}$ and $\{\mathbf{P}_2, \mathbf{P}_1\}$. A natural question is that if we divide the input system into one system with $n-k$ elements and another system with $n+k$ elements, whether the number of the total branches will be less. Hence, in this subsection, we will discuss this question.

For convenience, in this subsection, when we say a k-partial noise-bound, we mean the noise-bound about the noise vector $(e_1, e_2, \ldots, e_k)$. Then, the above division strategy is:

(A) Given a noise-bound u, we find the optimal solution from $\mathbf{P}_1 = \{f_1, f_2, \ldots, f_{n-k}, f_{n-k+1}, \ldots, f_{2n}\}$ under an $(n-k)$-partial noise-bound a and $\mathbf{P}_2 = \{f_{n-k+1}, f_{n-k+2}, \ldots, f_{2n}, f_1, f_2, \ldots, f_{n-k}\}$ under an $(n+k)$-partial noise-bound b, where $a + b = u - 1$.

The division strategy proposed in last subsection is:

(B) Given a noise-bound u, we find the optimal solution from $\mathbf{P}_1 = \{f_1, f_2, \ldots, f_n, f_{n+1}, \ldots, f_{2n}\}$ under an n-partial noise-bound a and $\mathbf{P}_3 = \{f_{n+1}, f_{n+2}, \ldots, f_{2n}, f_1, f_2, \ldots, f_n\}$ under an n-partial noise-bound b, where $a + b = u - 1$.

In the following we will prove that the number of branches solved in Strategy B is always smaller than that in Strategy A, when $k > 0$, and $|\mathrm{Im}(\mathcal{S}_n)| = 2^n$ for $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$.

Firstly, let's consider the forward direction. We will show that the number of branches in the search tree of Strategy B is less than that of Strategy A. For $\mathbf{P}_1$, since $|\mathrm{Im}(\mathcal{S}_n)| = 2^n$, as shown in last subsection, we can deduce that $|\mathbb{T}_{ISBS}| = |\bar{\mathbb{T}}_{ISBS}| = |\bar{\mathbb{T}}_{quasi} \cap \bar{\mathbb{T}}_{nw}| = |\bar{\mathbb{T}}_{nw}|$ for Strategy A, where $\mathbb{T}_{ISBS}, \mathbb{T}_{quasi}, \mathbb{T}_{nw}$ are defined as before. Note that, $\mathbb{T}_{nw}$ is generated from a perfect tree by pruning the branches which don't satisfy the noise-bound and the partial noise-bound. Hence, $|\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{a} \binom{n-k}{i} \sum_{j=0}^{u-i} \binom{k}{j}$. We have the following inequality about this value.

Lemma 7. $\sum_{i=0}^{a} \binom{n-k}{i} \sum_{j=0}^{u-i} \binom{k}{j} \geq \sum_{i=0}^{a} \binom{n}{i}$.

Proof. By Vandermonde's identity, we have

$$\begin{aligned}
\binom{n}{a} &= \binom{n-k}{a}\binom{k}{0} + \binom{n-k}{a-1}\binom{k}{1} + \cdots + \binom{n-k}{0}\binom{k}{a} \\
\binom{n}{a-1} &= \binom{n-k}{a-1}\binom{k}{0} + \binom{n-k}{a-2}\binom{k}{1} + \cdots + \binom{n-k}{0}\binom{k}{a-1} \\
&\vdots \\
\binom{n}{0} &= \binom{n-k}{0}\binom{k}{0}.
\end{aligned}$$

Since $u - i + i = u \geq a$, the terms in the right of the above equalities are all in $\sum_{i=0}^{a} \binom{n-k}{i} \sum_{j=0}^{u-i} \binom{k}{j}$, and these terms are distinct. Thus, the inequality is valid. □

Note that, for $\mathbf{P}_1$ in Strategy B, we have $|\mathbb{T}_{ISBS}| = |\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{a} \binom{n}{i}$. It proves the above conclusion about the forward direction.

Secondly, let's consider the backward direction. That is we solve $\mathbf{P}_2$ by Strategy A, and $\mathbf{P}_3$ by Strategy B. Similarly as above, we only need to compare the value of $|\mathbb{T}_{nw}|$. For $\mathbb{T}_{nw}$ of $\mathbf{P}_2$ in Strategy A, the only constraint that the paths with depth n should satisfy is $e_1 + e_2 + \cdots + e_n \leq b$. Hence $|\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{b} \binom{n}{i}$. Obviously, for $\mathbb{T}_{nw}$ of $\mathbf{P}_3$ in Strategy B, we also have $|\bar{\mathbb{T}}_{nw}| = \sum_{i=0}^{b} \binom{n}{i}$. This implies that the number of branches in the search trees of these two strategies are the same.

By combining the conclusions of the two directions, we can conclude that Strategy B is always the optimal strategy when the assumption $|\mathrm{Im}(\mathcal{S}_n)| = 2^n$ is valid.

4.3. The case $m = sn$

In this section, we extend the idea of the two-direction approach to the problems with $s > 2$, and present the s-direction approach. Similarly as Lemma 5, we have the following lemma.

Lemma 8. *Let $u_1, u_2, \ldots, u_s, u$ be $s+1$ non-negative integers, and $u_1 + u_2 + \cdots + u_s \leq u$. For any non-negative integers $a_1, a_2, \ldots, a_s$, such that $a_1 + a_2 + \cdots + a_s = u - s + 1$, at least one of the following inequalities hold: $u_1 \leq a_1$, $u_2 \leq a_2$,…, $u_s \leq a_s$.*

This lemma shows the following fact. We can divide a noise vector $E = (e_1, \ldots, e_m)$ into s parts

$$E_1 = (e_1, \ldots, e_n), E_2 = (e_{n+1}, \ldots, e_{2n}), \ldots, E_s = (e_{(s-1)n+1}, \ldots, e_{sn})$$

If E has a noise-bound u, then E_i has a noise-bound a_i, where $\sum_{i=1}^{s} a_i = u - 1$. Thus, we can build the following s-direction approach. That is we generate s polynomial systems:

$$\begin{aligned}
&\mathbf{P}_1 : \{f_1, \dots, f_n, f_{n+1}, , \dots, f_{2n}, \dots, f_{(s-1)n+1}, \dots, f_{sn}\}\\
&\mathbf{P}_2 : \{f_{n+1}, \dots, f_{2n}, f_1, \dots, f_n, f_{2n+1}, f_{2n+1}, \dots, f_{3n}, \dots, f_{(s-1)n+1}, \dots, f_{sn}\}\\
&\dots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (1)\\
&\mathbf{P}_s : \{f_{(s-1)n+1}, \dots, f_{sn}, f_1, \dots, f_n, f_{n+1}, \dots, f_{2n}, \dots, f_{(s-2)n+1}, \dots, f_{(s-1)n}\}.
\end{aligned}$$

Then, solve each $\mathbf{P}_i$ under the partial noise-bound a_i and the total noise-bound u.

Similarly as the case of $m = 2n$, when $|\mathrm{Im}(\mathcal{S}_n)| = 2^n$ for each $\mathbf{P}_i$, we can derive the best strategy of setting the values of these a_i from the following proposition.

Proposition 9. *Let u, n be two non-negative integers with $n/2 \geq u$. $s \geq 2$ is an integer. Suppose $u - s + 1 \equiv r$ mod s, and $p = (u - s + 1 - r)/s$. $a_1, a_2, \dots, a_s$ are s non-negative integers, s.t. $a_1 + a_2 + \dots + a_s = u - s + 1$. We have*

$$\sum_{i=0}^{u} \binom{n}{i} > \sum_{j=1}^{s}\sum_{i=0}^{a_j} \binom{n}{i} \tag{2}$$

$$\sum_{j=1}^{s}\sum_{i=0}^{a_j} \binom{n}{i} \geq \sum_{j=1}^{s-r}\sum_{i=0}^{p} \binom{n}{i} + \sum_{j=1}^{r}\sum_{i=0}^{p+1} \binom{n}{i}. \tag{3}$$

Proof. First, let's prove inequality (2). Since $u + 1 = (a_1 + 1) + (a_2 + 1) + \dots + (a_s + 1)$, we can divide $\sum_{i=0}^{u} \binom{n}{i}$ into s parts:

$$\binom{n}{0} + \ldots + \binom{n}{a_1}, \binom{n}{a_1 + 1} + \ldots + \binom{n}{a_1 + a_2 + 1}, \dots, \binom{n}{u - a_s} + \ldots + \binom{n}{u}.$$

Note that the j-th part is the sum of $a_j + 1$ elements. In the following, when we say a left-part, we mean one of these parts. We can write $\sum_{j=0}^{s}\sum_{i=0}^{a_j}\binom{n}{i}$ as the sum of s parts $\sum_{i=0}^{a_1}\binom{n}{i}, \dots, \sum_{i=0}^{a_s}\binom{n}{i}$. In the following, when we say a right-part, we mean one of such parts. Obviously the first left-part $\sum_{i=0}^{a_1}\binom{n}{i}$ is equal to the first right-part. For the j-th left-part with $j > 1$, since $n/2 \geq u$, we can check that each element in this left-part is bigger than the corresponding element in the j-th right-part. Hence the j-th left-part is bigger than the j-th right-part. Since $s \geq 2$, we at least have two parts. Thus, we can derive the first inequality.

Now, let's prove inequality (3). Without loss of generality, we can assume $a_1 \leq a_2 \leq \dots \leq a_s$. Note that $a_1 + a_2 + \dots + a_s = u - s + 1 = (s - r)p + r(p + 1)$. Suppose among these a_i, there are s_1 elements being smaller than p, b elements being equal to p, c elements being equal to $p + 1$ and s_2 elements being bigger than $p + 1$. Then $a_1, a_2, \dots, a_s$ can be written as

$$a_1, a_2, \dots, a_{s_1}, \underbrace{p, p, \dots, p}_{b}, \underbrace{p + 1, p + 1, \dots, p + 1}_{c}, a_{s-s_2+1}, a_{s-s_2+2}, \dots, a_s,$$

where $s_1 + b + c + s_2 = s$.

Now we consider the following three cases:

1. $s_1 + b = s - r$ and $c + s_2 = r$. In this case, the left side of (3) minus the right side of (3) is equal to

$$\sum_{j=1}^{s_2}\sum_{i=p+2}^{a_{s-s_2+j}} \binom{n}{i} - \sum_{j=1}^{s_1}\sum_{i=a_j+1}^{p} \binom{n}{i} \tag{4}$$

The first part of (4) has $T_1 = a_{s-s_2+1} + a_{s-s_2+2} + \dots + a_s - s_2(p + 1)$ terms. The second part of (4) has $T_2 = s_1 p - (a_1 + a_2 + \dots + a_{s_1})$ terms. $T_1 - T_2 = (u - s + 1) - bp - c(p + 1) - s_1 p - s_2(p + 1) = (u - s + 1) - (s - r)p - r(p + 1) = 0$, which means the first and second parts of (4) have the same number of terms. Moreover, $a_{s-s_2+j} \geq p + 2 > p \geq a_i + 1$, which means every term in the left part of (4) is bigger than that in the right part of (4). Then we have (4) is not smaller than 0, which implies that the second inequality of (3) is valid, and the equality holds when $b = s - r$ and $c = r$.

2. $s_1 + b > s - r$ and $c + s_2 < r$. In this case, the left side of (3) minus the right side of (3) is equal to

$$\sum_{j=1}^{s_2}\sum_{i=p+2}^{a_{s-s_2+j}} \binom{n}{i} - \left(\sum_{j=1}^{b+s_1-s+r} \binom{n}{p+1} + \sum_{j=1}^{s_1}\sum_{i=a_j+1}^{p} \binom{n}{i}\right) \tag{5}$$

Similarly as case 1, the first and second parts of (5) have the same number of terms, and every term in the left is bigger than that in the right, which implies the correctness of the second inequality of (3).

3. $s_1 + b < s - r$ and $c + s_2 > r$. In this case, the left side of (3) minus the right side of (3) is equal to

$$\left(\sum_{j=1}^{s_2}\sum_{i=p+2}^{a_{s-s_2+j}}\binom{n}{i}+\sum_{j=1}^{c+s_2-r}\binom{n}{p+1}\right)-\sum_{j=1}^{s_1}\sum_{i=a_j+1}^{p}\binom{n}{i} \tag{6}$$

   Similarly as the above cases, the first and second parts of (6) have the same number of terms, and every term in the left is bigger than that in the right, which implies the correctness of the second inequality of (3).

In summary, inequality (2) is valid in any cases. □

This proposition shows that when $|\text{Im}(\mathcal{S}_n)| = 2^n$ for all these $\mathbf{P}_i$, the number of branches solved in the s-direction approach is always less than that in the original approach, and the best strategy is setting the partial noise-bounds of $s - r$ systems to be p and those of the rest r systems to be $p + 1$.

Based on the above theoretical results, similarly as algorithm **ISBS**$_2$, we can implement an algorithm by using the s-direction approach. The main steps of the algorithm are as follows:

1. Given an input system $\mathbf{P} : \{f_1, \dots, f_n, f_{n+1}, \dots, f_{2n}, \dots, f_{(s-1)n+1}, \dots, f_{sn}\}$.
2. Generate s systems $\mathbf{P}_1, \mathbf{P}_2, \dots, \mathbf{P}_s$ as (1).
3. Set a noise-bound u. Suppose $u - s + 1 \equiv r \bmod s$, and let $p = (u - s + 1 - r)/s$.
4. For $\mathbf{P}_1, \dots, \mathbf{P}_{s-r}$, set their partial noise-bounds to be p. For $\mathbf{P}_{s-r+1}, \dots, \mathbf{P}_s$, set their partial noise-bounds to be $p + 1$.
5. For i from 1 to s, solve $\mathbf{P}_i$ with the **ISBS** method under noise-bound u and its partial noise-bound.
   - If for some $\mathbf{P}_k$, we achieve a solution x, then set x_0 to be x and u to be $w(x) - 1$, where $w(x)$ is the noise weight of the noise vector corresponding to x. Update the partial noise-bounds of $\mathbf{P}_{k+1}, \mathbf{P}_{k+2}, \dots, \mathbf{P}_s$ by the strategy used in Step 4.
6. If x_0 is not empty, then it is the solution of the Max-PoSSo problem. Otherwise, increase u by a step size, and repeat Step 4–5.

Remark 3. Based on the above results about $|\mathbb{T}_{ISBS}|$, we can deduce some complexity bounds of **ISBS**. First, suppose the complexity of solving one branch of $\mathbb{T}_{ISBS}$ is C. Then, for the original version of **ISBS**, we have $|\mathbb{T}_{ISBS}| \le |\mathbb{T}_{quasi}| \le 2^n$, hence its complexity is bounded by $C \cdot 2^n$. To the best of the author's knowledge, the best worst-case complexity of incrementally solving a polynomial system with n variables and m polynomials is $O(m2^n)$, thus the complexity of the original **ISBS** is bounded by $O(m2^{2n})$. For **ISBS**$_b$, u, the noise-bound of the optimal solution, can be approached by the artificial noise-bound, hence we have $|\mathbb{T}_{ISBS}| = |\mathbb{T}_{quasi} \cap \mathbb{T}_{nw}| \le min(2^n, \sum_{i=0}^{u}\binom{m}{i})$. In most times, $2^n \ge \sum_{i=0}^{u}\binom{m}{i}$, hence the worst-case complexity of **ISBS**$_b$ is $O(m\sum_{i=0}^{u}\binom{m}{i}2^n) = O(m^{u+1}2^n)$. For the s-direction version of **ISBS**, the partial noise-bound is nearly u/s, where u is the noise weight of the optimal solution and $s = m/n$, hence $|\mathbb{T}_{ISBS}| \approx s\sum_{i=1}^{u/s}\binom{n}{i}$, and the worst-case complexity of **ISBS** is $O(sn^{u/s}2^n)$. These complexity bounds show that **ISBS** is a single exponential algorithm. In practical computation, for some polynomial systems, the complexity of solving one branch of $\mathbb{T}_{ISBS}$ is much lower than $O(m2^n)$ and sometime is polynomial, then the practical complexity of **ISBS** can be much lower.

5. Experimental results

In order to test the proposed improvements, we generated some benchmarks from the Cold Boot key recovery problem of Serpent. The Cold Boot key recovery problem is originated from the Cold Boot attack, which was first proposed and discussed in the seminal work of [9]. The Cold Boot attack relies on the data remanence property of DRAM to retrieve memory contents after power off. In the Cold Boot attack to a block cipher, the attacker is able to retrieve the round keys, but some bits of the round keys is flipped since the decay of the memory data. Thus, the Cold Boot key recovery problem for a block cipher is to recover the initial key for these decayed round keys.

In [1], Cid and Albrecht proposed a mathematical model, by which one can convert Cold Boot key recovery problems into Partial Weighted Max-PoSSo problems, then they solved some Cold Boot key recovery problems of AES and Serpent by mixed integer programming solver **SCIP**. In [11], the same problems were solved by the original **ISBS** algorithm and some better experimental results were presented.

Note that in this paper we focus on solving the Max-PoSSo problem, hence in our experiments, unlike the general Cold Boot key recovery model, we assume that the bit decay in DRAM is symmetric: bit flips 0 → 1 and 1 → 0 occur with same probabilities δ. Under this assumption, the Cold Boot key recovery problem of a block cipher can be described as follows.

Let $\mathcal{KS} : \mathbb{F}_2^n \to \mathbb{F}_2^N$ be the key schedule function of a block cipher, where $N > n$. Let $G_{\mathcal{K}}$ be an equation system corresponding to $\mathcal{KS}$ such that the only pairs (k, K) that satisfy $G_{\mathcal{K}}$ are any initial key $k \in \mathbb{F}_2^n$ and round key $K = \mathcal{KS}(k)$. Moreover, each $g_i \in G_{\mathcal{K}}$ has the form $h_i + K_i$ where h_i is some polynomial and K_i is the i-th bit of K. Let K' be the decayed round keys, and set $f_i = h_i + K'_i$ for any $1 \le i \le N$. Then, $\mathbf{P} = \{f_1, f_2, \dots, f_N\}$ is the input of the Max-PoSSo problem we need to solve.

Table 1
Serpent considering $32 \cdot N$ bits of key schedule output (symmetric noise).

$\delta_0 = \delta_1$	Method	N	Limit t	r	min t	avg. t	max t
0.01	**ISBS$_2$**	8	3600.0 s	100%	0.60 s	2.46 s	30.62 s
	ISBS$_0$	8	3600.0 s	100%	0.78 s	9.87 s	138.19 s
	SCIP	12	3600.0 s	96%	4.60 s	256.46 s	–
0.02	**ISBS$_2$**	8	3600.0 s	96(99)%	0.82 s	55.67 s	996.65 s
	ISBS$_0$	8	3600.0 s	96(99)%	0.80 s	163.56 s	2001.59 s
	SCIP	12	3600.0 s	79%	8.20 s	1139.72 s	–
0.03	**ISBS$_2$**	8	3600.0 s	91(95)%	0.58 s	171.17 s	2138.77 s
	ISBS$_0$	8	3600.0 s	90(92)%	1.74 s	314.78 s	3463.00 s
	SCIP	12	7200.0 s	53%	24.57 s	4205.34 s	–
0.05	**ISBS$_2$**	8	3600.0 s	40(98)%	3.67 s	382.61 s	1916.91 s
	ISBS$_0$	8	3600.0 s	38(94)%	12.37 s	745.80 s	2993.81 s
	SCIP	12	3600.0 s	18%	5.84 s	1921.89 s	–

In [1] and [11], benchmarks with symmetric noise generated from the 128-bit version of Serpent were tested, and in the experiments of this paper, we solved the same benchmarks. Since we use 256-bit round keys which means $m = 2n$, the improved **ISBS** algorithm which was implemented and tested in our experiments is **ISBS$_2$**, and in our implementation the polynomial system solving process is executed by running the Characteristic Set algorithm **MFCS** proposed in [7]. We compared the experimental results of **ISBS$_2$** with those of **ISBS$_0$** and **SCIP**. Here, **ISBS$_0$** denotes the **ISBS** algorithm implemented and tested in [11], and **SCIP** denotes the mixed integer programming method used in [1].

Our experimental platform is a PC with i7 2.8 Ghz CPU (only one core is used), and 4 GB Memory, which is same as the one used in [11]. In our experiments, for each δ we generated 100 instances with random initial keys and random noises. As in [1,11], we interrupted the solver when the running time exceeded the time limit 3600 seconds.

In Table 1, the column "r" gives the success rate, which is the percentage of the instances we recovered the correct initial key, while the values in the brackets are the percentage of the instances we achieved the optimal solution within the time limit. Note that, there are two cases in which we cannot recover the correct initial key.

(1) The solver was interrupted after the time limit.
(2) The solution achieved from the Max-PoSSo problem is not the true solution.

The column "avg. time" gives the average running time of the instances which are solved within the time limit, and the column "max t" gives the maximal running time for the instances which are solved within the time limit.

From the experimental results, we can see that when $\delta = 0.01$, **ISBS$_2$** is about 4 times faster than **ISBS$_0$**. When $\delta = 0.02, 0.03, 0.05$, **ISBS$_2$** is about 2 times faster than **ISBS$_0$**. When $\delta = 0.05$, as in [11], we interrupted the solver after we have searched all the possible noise vectors under the noise-bound 12, thus although 98 instances ended within the time limit, only 40 of them returned the true solutions. In summary, these experimental results show that with our modification we significantly improve the efficiency of **ISBS**.

Remark 4. The above experimental results imply that **ISBS** can be an efficient algorithm when the error rate δ is small. On our experimental platform, we can solve the above Serpent problems with $\delta = 0.08$ by **ISBS$_2$** in about 6 hours. It seems that problems with higher error rate cannot be solved in reasonable time by our method.

6. Conclusions

In this paper, we revisit the Max-PoSSo problem and the **ISBS** method. For the basic of Max-PoSSo, we show some results about the behavior of the success rate of recovering the true solution. For **ISBS**, we present some theoretical results about the number of branches in the search tree, and propose some ways to decrease the number of branches for general polynomial systems and overdetermined polynomial systems. We implement a new algorithm based on these improvements and test it by solving the Cold Boot Key recovery problem of Serpent with symmetric noise. The experimental results demonstrate that compared with the **ISBS** algorithm implemented in [11], the new algorithm is about 2–4 times faster for different benchmarks.

There is an idea of further improving **ISBS** which can be applied in the future. In the improved algorithm, when the artificial noise-bound increases gradually, there are some repeated computations which can be avoided. We know that $\mathbb{T}_{quasi}$ doesn't change after the noise-bound increased. It means that after we increase the artificial noise-bound we only need to continue searching the paths in $\mathbb{T}_{quasi}$ which are pruned because of the former noise-bound. Therefore, if we can store the information of all these paths efficiently, a lot of repeated computations can be avoided.

Acknowledgements

This work was in part supported by the National Key Basic Research Program of China under Grant No. 2013CB834203, the National Key Research and Development Plan of China under Grant No. 2016YFB0200504, and the National Natural Science Foundation of China under Grant No. 61502485.

Appendix A. The strict proof of Proposition 2

In the following, we assume $1/r$ is an integer. First, we need the following two lemmas by which we can present the binomial cumulative function by the incomplete beta function. The two lemmas can be easily proved by integration by parts.

Lemma 10. [10] *Let n, c be positive integer, and $0 \leq p \leq 1$ is a real number. We have*

$$\sum_{k=0}^{c} \binom{n}{k} p^k (1-p)^{n-k} = \frac{\int_0^{1-p} t^{n-c-1}(1-t)^2 dt}{B(n-c, c+1)},$$

where $B(a, b)$ is the Beta function.

Lemma 11. [10] *Let $I_x(a, b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1}dt}{B(a,b)}$. This $I_x(a, b)$ is called the incomplete beta function. Then, we have*

(1) $I_x(a, b) = 1 - I_{1-x}(b, a)$

(2) $I_x(a+1, b) = I_x(a, b) - \dfrac{x^a(1-x)^b}{aB(a, b)}$.

Proof of Proposition 2.

$$\sum_{i=0}^{a} (\binom{n-k}{i} \sum_{j=0}^{a+b-i} \binom{k}{j}) + \sum_{i=0}^{b} \binom{n}{i} \geq \sum_{i=0}^{a} \binom{n}{i} + \sum_{i=0}^{b} \binom{n}{i}.$$

Let $k = mr$, and $s = 1/r$ be an integer. From the above two lemmas, we have $f(k) = \frac{\sum_{i=0}^{k} \binom{m}{i}}{2^m} = I_{1/2}(k(s-1), k+1)$. Then, the conclusion is equivalent to $f(k+t) \leq f(k)$ for any $k, t \in N$. Thus, it is sufficient to prove $f(k+1) \leq f(k)$, $\forall k \in N$.

Note that by Lemma 11, we have

$$\begin{aligned} f(k+1) &= I_{1/2}((k+1)(s-1), k+2) = 1 - I_{1/2}(k+2, (k+1)(s-1)) \\ &= 1 - I_{1/2}(k+1, (k+1)(s-1)) + \frac{1}{2^{(k+1)s}(k+1)B(k+1, (k+1)(s-1))} \\ &= I_{1/2}((k+1)(s-1), k+1) + \frac{1}{2^{(k+1)s}(k+1)B(k+1, (k+1)(s-1))}. \end{aligned}$$

For simplicity, we set $a = (k+1)(s-1)$, $b = k+1$. By applying (2) of Lemma 11 $s-1$ times, we have

$$I_{1/2}(a, b) = I_{1/2}(a-(s-1), b) - \sum_{i=1}^{s-1} \frac{1}{2^{a+b-i}(a-i)B(a-i, b)}.$$

Note that $I_{1/2}(a-(s-1), b) = I_{1/2}(k(s-1), k+1) = f(k)$. Thus,

$$f(k+1) - f(k) = \frac{1}{2^{a+b}bB(b, a)} - \sum_{i=1}^{s-1} \frac{1}{2^{a+b-i}(a-i)B(a-i, b)}. \tag{7}$$

Since $1 \leq i \leq s$, we have

$$\begin{aligned} &\frac{1}{(a-i)B(a-i, b)} = \frac{1}{(a+b-i)B(a-i+1, b)} \\ &= \frac{a-i+1}{(a+b-i)(a+b-i+1)B(a-i+2, b)} \\ &= \cdots = \frac{(a-i+1)(a-i+2)\cdots(a-1)}{(a+b-i)(a+b-i+1)\cdots(a+b-1)B(a, b)} \\ &\geq \frac{(a-s+1)^{i-1}}{(a+b-1)^i B(a, b)}. \end{aligned}$$

Table 2
Incrementally solving the serpent problems by different methods.

Characteristic set	Gröbner basis	Cryptominisat
0.67 s	3.47 s	33.59 s

Therefore, by applying the above inequality to (7), we have

$$f(k+1) - f(k) \leq \frac{1}{2^{a+b} b B(b,a)} - \frac{1}{2^{a+b}(a-s+1)B(a,b)} \sum_{i=1}^{s-1} (\frac{2(a-s+1)}{a+b-1})^i.$$

Let $q = \frac{2(a-s+1)}{a+b-1}$, then $f(k+1) - f(k) = \frac{1}{2^{a+b}B(b,a)}(\frac{1}{b} - \frac{2}{a+b-1}(\frac{1-q^{s-1}}{1-q}))$.

Now, it is sufficient to show $\frac{1}{b} - \frac{2}{a+b-1}(\frac{1-q^{s-1}}{1-q}) \leq 0$. If $s = 2$, the conclusion is correct obviously. Now we consider the case $s \geq 3$. In this case, $q - 1 = \frac{2(a-s+1)}{a+b-1} = \frac{(k-1)(s-2)-1}{a+b-1} > 0$. Thus, $\frac{1-q^{s-1}}{1-q} = 1 + q + \cdots + q^{s-2} > s - 1$. Then $\frac{1}{b} - \frac{2}{a+b-1}(\frac{1-q^{s-1}}{1-q}) < \frac{1}{k+1} - \frac{2(s-1)}{(k+1)s-1} < \frac{1}{k+1} - \frac{2(s-1)}{(k+1)s} < \frac{2-s}{(k+1)s} < 0$. □

Appendix B. The incremental solving process

In this part, we investigate the efficiency of the method used in the incremental solving process. Here, we consider three methods, which are the Characteristic set method, the Gröbner basis method, and the SAT-solver. We test these methods by incrementally solving the polynomial systems generated from the Cold boot key recovery problems of Serpent in Section 5. The input polynomial system has 128 variables, and 256 polynomials. In the experiments, for the Characteristic Set method, we used our implementation of **MFCS** as Section 5. For the Gröbner Basis method, we used the Boolean Gröbner Basis function in Magma (Version 2.20) under graded reverse lexicographical order. For the SAT-solver, we used the Cryptominisat 5.0.0, which is the most efficient SAT-solver for solving SAT problems converted from Boolean polynomial systems. Table 2 lists the average running times for solving 100 instances by these methods.

From the results, we can conclude that the Characteristic Set method is the most efficient one for incrementally solving this kind of polynomial systems. We think the reason is as the following. In the incremental solving process, the SAT-solver cannot represent and store the former results by an easy way, since at the beginning of incremental solving, a lot of points can satisfy the first several polynomials. Then one cannot use the former results to reduce repeating computations. Thus, the SAT-solver is not suitable for incremental solving. For the Gröbenr Basis method, although it can represent and store the former results well, since the number of polynomials is small at the beginning of the incremental solving process, the degree of regularity will be high and the fast Gaussian elimination techniques cannot be well used. Hence, its efficiency is not high.

References

[1] M.R. Albrecht, C. Cid, Cold boot key recovery by solving polynomial systems with noise, in: ACNS, 2011, pp. 57–72.
[2] E. Biham, R. Anderson, L. Knudsen, Serpent: a new block cipher proposal, in: International Workshop on Fast Software Encryption, Springer, Berlin, Heidelberg, 1998, pp. 222–238.
[3] A. Braeken, B. Preneel, Probabilistic algebraic attacks, in: Cryptography and Coding, Springer, Berlin, Heidelberg, 2005, pp. 290–303.
[4] F. Chai, X.S. Gao, C. Yuan, A characteristic set method for solving Boolean equations and applications in cryptanalysis of stream ciphers, J. Syst. Sci. Complex. 21 (2) (2008) 191–208.
[5] J.C. Faugère, A new efficient algorithm for computing Gröbner bases (F4), J. Pure Appl. Algebra 139 (1–3) (1999) 61–88.
[6] J.C. Faugère, A new efficient algorithm for computing Gröner bases without reduction to zero (F5), in: Proc. ISSAC, 2002, pp. 75–83.
[7] X.S. Gao, Z. Huang, Characteristic set algorithms for equation solving in finite fields, J. Symbolic Comput. 47 (6) (2012) 655–679.
[8] J. Håstad, Satisfying degree-d equations over GF$[2]^n$, in: APPROX-RANDOM, 2011, pp. 242–253.
[9] J.A. Halderman, S.D. Schoen, N. Heninger, W. Clarkson, W. Paul, J.A. Calandrino, A.J. Feldman, J. Appelbaum, E.W. Felten, Lest we remember: cold boot attacks on encryption keys, in: USENIX Security Symposium, USENIX Association, 2009, pp. 45–60.
[10] H.O. Hartley, E.R. Fitch, A chart for the incomplete beta-function and the cumulative binomial distribution, Biometrika 38 (3/4) (1951) 423–426.
[11] Z. Huang, D. Lin, A new method for solving polynomial systems with noise over $\mathbb{F}_2$ and its applications in cold boot key recovery, in: Selected Areas in Cryptography, Windsor, Canada, in: Lecture Notes in Comput. Sci., vol. 7707, 2013, pp. 16–33.
[12] D. Joan, V. Rijmen, The Design of Rijndael: AES-the Advanced Encryption Standard, Springer Science & Business Media, 2013.
[13] S.W. Zhao, X.S. Gao, Minimal achievable approximation ratio for MAX-MQ in finite fields, Theoret. Comput. Sci. 410 (21–23) (2009) 2285–2290.