An Enhanced Binary Characteristic Set Algorithm and Its Applications to Algebraic Cryptanalysis

Sze Ling Yeo[1], Zhen Li[1(✉)], Khoongming Khoo[2], and Yu Bin Low[2]

[1] Infocomm Security Department, Institute for Infocomm Research,
1 Fusionopolis Way, #21-01 Connexis, Singapore 138632, Singapore
slyeo@i2r.a-star.edu.sg, lizh0019@gmail.com
[2] DSO National Laboratories, 20 Science Park Drive, Singapore 118230, Singapore
{kkhoongm,lyubin}@dso.org.sg

Abstract. Efficient methods to solve boolean polynomial systems underly the effectiveness of algebraic attacks on cryptographic ciphers and the security of multi-variate cryptosystems. Amongst various polynomial solving algorithms, the binary characteristic set algorithm was recently proposed to solve boolean polynomial systems including those arising from ciphers. In this paper, we propose some novel techniques to enhance the existing characteristic set solver. Specifically, we incorporate the ElimLin procedure and apply basic statistical learning techniques to improve the performance of the characteristic set algorithm. Our experiments show that our enhanced solver EBCSA performs better than existing algebraic methods on some ciphers, including CANFIL and PRESENT ciphers. We also perform the first algebraic cryptanalysis on the PRINCE cipher and an algebraic attack on Toyocrypt in a more practical/realistic setting as compared to previous attacks.

Keywords: Characteristic set algorithm · Algebraic cryptanalysis · ElimLin · Statistical learning

1 Introduction

1.1 Background

In the search for possible cryptographic algorithms in the post-quantum era, multi-variate cryptography has emerged as one of the potential candidates. This branch of cryptography comprises schemes based on the difficulty to solve multivariate polynomials over finite fields. While solving generic multi-variate polynomial systems is known to be NP-hard, a number of polynomial systems arising from cryptographic constructions have been efficiently solved. A well-known example is the polynomial system from the HFE cryptosystem [1]. As such, it is important to have a better understanding of the existing methods to solve multi-variate polynomial systems constructed from cryptosystems.

D. Gollmann et al. (Eds.): ACNS 2017, LNCS 10355, pp. 518–536, 2017.
DOI: 10.1007/978-3-319-61204-1_26

Various methods to solve multivariate polynomial systems exist. The most common approach to solve generic polynomial systems over finite fields is Gröbner basis algorithms [2–4]. Typically, a Gröbner basis with respect to the degree reverse lexicographical ordering is first computed via algorithms F_4 or F_5 [3,4]. It is then converted to a Gröbner basis with respect to the lexicographical ordering by algorithms such as the FGLM algorithm [5] which contains equations where variables are eliminated. This enables the variables to be solved one at a time. See [6] for more details.

Another approach to solve multivariate polynomial systems is the XL algorithm and its variants [7–10]. This class of algorithms performs well when the system under consideration is overdetermined, that is, the number of equations far exceeds the number of variables. Briefly, this method works by considering multiples of the generating polynomials bounded by some degree for which the number of independent equations exceeds the number of monomials. Common linear algebra techniques can then be applied to solve this resulting system.

In both the Gröbner basis and XL type approaches, new polynomials are gradually added with degrees larger than the degree of the generating polynomials. This often causes the number of monomials to increase very rapidly, thereby resulting in excessive memory requirements. The ElimLin algorithm was first proposed by Courtois to attack DES [11,12]. Essentially, the ElimLin algorithm seeks to successively find linear equations in the vector space generated by the original equations and subsequently, eliminating variables using these linear equations. Observe that his process constructs new polynomial systems with fewer variables without an increase in the degree. In fact, the "ElimLin" subroutine is incorporated into most Gröbner basis implementations.

All the above methods have been exploited in algebraic cryptanalysis of ciphers to solve for the unknown key variables in the polynomial system representing the ciphers. In general, block ciphers are less vulnerable to algebraic cryptanalysis as the polynomials usually have very high degrees or involve a large number of variables. Nonetheless, some block ciphers had been broken by algebraic attacks, including Keeloq [13]. As another example, ElimLin was used in [14] to solve 5 rounds of Present with half of the keybits fixed and 5 known plaintext/ciphertext pairs.

1.2 The Characteristic Set Algorithm

The characteristic set algorithm (or CSA for short) had been well-established to analyze algebraic properties of polynomial systems over algebraically-closed fields of characteristic 0, see [15,16]. In [17,18], the authors adapted this approach to solve polynomial systems over any finite field. Specifically, CSA seeks to decompose a polynomial system into monic triangular sets of polynomials with disjoint solution sets. Thus, the problem now boils down to solving a monic triangular set of polynomials which can be easily accomplished. In the case of a binary field, they further refined the decomposition algorithm so that the degrees of newly generated polynomials do not increase. The authors of [17] provided some experimental evidence which demonstrated that CSA seems to be more

effective on sparse polynomial systems, particularly those exhibiting a block triangular structure. Examples of such systems include polynomials arising from linear feedback shift registers with nonlinear combining functions used in stream ciphers.

1.3 Our Contributions

In this paper, we propose some novel techniques to further enhance the binary CSA. The main features can be summarized as follows:

- we incorporate ElimLin into the binary CSA.
- we apply some statistical learning techniques to choose the next "splitting polynomial".
- we also adaptively choose the next polynomial set to process.
- we propose a preprocessing phase where the variables can be sorted.
- we allow for some flexibility in choosing the combination of features suitable for each polynomial system.

We call our enhanced version the enhanced binary characteristic set algorithm (EBCSA for short). In this paper, we present our experimental results on the block ciphers Present and Prince, as well as the stream ciphers Canfil and Toyocrypt. First, we benchmark EBCSA against prior versions of the characteristic set algorithms on Canfil ciphers. For Present cipher, we show that EBCSA outperforms previous algebraic methods as 5 rounds can be solved with fewer fixed keybits and known plaintext/ciphertext pairs. Moreover, we provide the first algebraic attack on 6 rounds of Present. As for the block cipher Prince, we carried out the first algebraic attack and our results showed that EBCSA is more effective than brute force search for 6 rounds of Prince-core (i.e. without whitening). Finally, we show that the stream cipher Toyocrypt can be solved by EBCSA in a more realistic setting compared to previous attacks, namely, using re-synchronization with sufficient random IV's for a small number of keystream bits.

1.4 Organization of This Paper

This paper is organized as follows. In Sect. 2, we set out the notations and terminologies. We then provide a general description of the binary characteristic set algorithm [17] and its variant [18]. In Sect. 3, we describe our new techniques and present our enhanced binary characteristic set algorithm. Our experimental results are then provided in Sect. 4. Finally, we wrap up with some concluding remarks and suggestions for future research.

2 A Description of the Characteristic Set Algorithm

2.1 Notations and the Set-Up

Let $F = \mathcal{F}_2$ denote the binary field. For a positive integer n, let $R = F[x_0, x_1, \ldots, x_{n-1}]$ be the multivariate polynomial ring in n variables. Since

$x^2 = x$ for all $x \in F$, we are interested in the quotient ring $\overline{R} = R/\{x_i^2 + x_i : i = 0, 1, \ldots, n-1\}$. In particular, every polynomial in R is equivalent to a representative $p \in \overline{R}$ such that the degree of each variable in p is at most 1. In the following, we will represent all the polynomials in this form.

Definition 1. *Fix σ to be a permutation on $\{0, 1, \ldots, n-1\}$. Let P be a polynomial in $\overline{R}$. We define the class of P with respect to σ to be the smallest i such that $x_{\sigma(i)}$ occurs in P. We denote the class of P by $\mathrm{cls}_\sigma(P)$. If σ is clear in the context, we will simply denote it by $\mathrm{cls}(P)$.*

Remark 1. – *For a constant polynomial $P = 0$ or 1, we define $\mathrm{cls}(P) = -\infty$.*
– *Let σ be the identity permutation, that is, $\sigma(i) = i$ for $i = 0, 1, \ldots, n-1$. Then $\mathrm{cls}_\sigma(P)$ is the smallest i such that x_i occurs in P. On the other hand, suppose that $\sigma(i) = n-1-i$ for $i = 0, 1, \ldots, n-1$. Then $\mathrm{cls}_\sigma(P)$ is the largest i such that x_i occurs in P. This coincides with the definition in* [17].

For a polynomial $P \in \overline{R}$, let c be its class with respect to a permutation σ. Then P can be expressed as $P = Ix_{\sigma(c)} + U$, where $x_{\sigma(c)}$ does not occur in both I and U. We term I and U as the left child and right child of P, respectively. By the definition of c, it follows that both $\mathrm{cls}_\sigma(I)$ and $\mathrm{cls}_\sigma(U)$ are strictly greater than c. Note that a monic polynomial is one in which $I = 1$.

Definition 2. *Let $\mathcal{A}$ be a finite set of polynomials in $\overline{R}$. $\mathcal{A}$ is called triangular if the classes of the polynomials in $\mathcal{A}$ are all distinct. Moreover, $\mathcal{A}$ is monic triangular if every polynomial in $\mathcal{A}$ is monic.*

The Main Problem: Let $\mathcal{P}$ be a finite set of boolean polynomials from $\overline{R}$. We wish to find the zero set of $\mathcal{P}$, that is, to find $Z(\mathcal{P}) = \{(x_0, x_1, \ldots, x_{n-1}) \in F^n : P(x_0, x_1, \ldots, x_{n-1}) = 0 \text{ for all } P \in \mathcal{P}\}$.

2.2 The Basic Structure of Binary CSA

In this section, we describe the main principles underlying binary CSA to help solve the main problem stated above. Our description is more generic than that in [17] in the sense that we allow for a variable ordering on the variables. Note that we will focus on the multiplication-free CSA or MFCS presented in [17] as it was already shown to result in the best performance for the binary case. As such, CSA will refer to MFCS from now on. Throughout this section, we fix a permutation σ and simply write $\mathrm{cls}(P)$ for the class of P. Essentially, the approach of binary CSA is a result of the following lemma.

Lemma 1. *Let P be a polynomial of $\overline{R}$ with class c and write $P = Ix_{\sigma(c)} + U$. Then $Z(\{P\})$ can be split into $Z(\{P\}) = Z(\mathcal{P}_1) \cup Z(\mathcal{P}_2)$ such that: (1) $Z(\mathcal{P}_1)$ and $Z(\mathcal{P}_2)$ are disjoint; (2) One of $\mathcal{P}_1$ or $\mathcal{P}_2$ contains a monic polynomial with class c.*

Proof. As we are in the binary setting, the left child of P, namely I, can only take values 0 or 1 and $Z(\{I\})$ and $Z(\{I+1\})$ are disjoint. When $I = 0$, we

have $U = P + Ix_{\sigma(c)} = 0$. On the other hand, if $I = 1$, we have $x_{\sigma(c)} + U = Ix_{\sigma(c)} + U = P = 0$. Let $\mathcal{P}_1 = \{I, U\}$ and $\mathcal{P}_2 = \{I + 1, x_{\sigma(c)} + U\}$. The lemma now follows.

Suppose that $\mathcal{A}$ is a monic triangular set of polynomials. Clearly, $\mathcal{A}$ cannot contain more than n polynomials. Suppose that $0 \leq c_1 < c_2 < \ldots < c_t \leq n-1$ are the classes of the polynomials in $\mathcal{A}$. Then, $\mathcal{A}$ contains the polynomials $P_i = x_{\sigma(c_i)} + U_i$, $i = 1, 2, \cdots, t$, where $U_1, U_2, \ldots, U_t$ represents the right child of $P_1, \ldots, P_t$, respectively. Observe that since $c_1 < c_2 < \ldots < c_t$, U_t does not contain the variables $x_{\sigma(c_1)}, \ldots, x_{\sigma(c_t)}$. It follows that as long as the variables of U_t are fixed to some values, $x_{\sigma(c_t)}$ is determined. Similarly, U_{t-1} does not contain the variables $x_{\sigma(c_1)}, \ldots x_{\sigma(c_{t-1})}$ which allows one to compute $x_{\sigma(c_{t-1})}$. In this way, we can determine the zeroes of $\mathcal{A}$ by letting x_i, $i \notin \{\sigma(c_1), \ldots, \sigma(c_t)\}$ take values 0 or 1 and computing the corresponding values of $x_{\sigma(c_1)}, \ldots, x_{\sigma(c_t)}$ from the polynomials in $\mathcal{A}$.

In view of the above, the main goal of binary CSA is to decompose $Z(\mathcal{P})$ into disjoint sets based on Lemma 1. Specifically, we have the following theorem:

Theorem 1. [17, Theorems 4.1, 4.2] *Let $\mathcal{P}$ be a finite set of boolean polynomials of $\overline{R}$. The binary CSA decomposes $Z(\mathcal{P})$, in a finite number of steps, into a disjoint union of zero sets, namely,*

$$Z(\mathcal{P}) = \bigcup_{j=1}^{s} Z(\mathcal{A}_j),$$

where $\mathcal{A}_1, \ldots, \mathcal{A}_s$ are all monic triangular sets of boolean polynomials.

The main structure of CSA is described in Algorithm 1 in the Appendix. In order to achieve the goal of Theorem 1, the main function of binary CSA is the triset algorithm which takes in as input a finite set of boolean polynomials and outputs a monic triangular set $\mathcal{A}$ (possibly empty) as well as a set $\mathcal{P}^*$ of sets of boolean polynomials.

Remark 2. Beginning with the set $\mathcal{P}$, the whole CSA can be thought of as building a binary tree of sets ($\mathcal{P}^*$) by adding $\mathcal{Q}^*$ as a branch to the corresponding input set to the triset function. The process stops when all the sets in the tree are processed and the result is a set $\mathcal{A}^*$ of monic triangular sets. Since the polynomial splits by letting $I = 0$ or 1, the binary CSA is essentially a generalization of brute force search where we split by letting the variables take 0 or 1.

Remark 3. Notice that we iteratively split polynomials to obtain sets with disjoint solutions. In this basic version of the triset function, we work on polynomials according to their class, namely, we pick the polynomial with the smallest class. In addition, the next input set to be fed into the triset function is generally taken to be the last generated set in $\mathcal{P}^*$.

2.3 BCSA

In [18], the authors proposed some additional features to improve the efficiency of the binary CSA in [17]. We call it BCSA and briefly describe the techniques here.

Linearization. The authors claimed that this technique was already implemented in the experiments of [17] but it was not explicitly mentioned. This technique applies to the triset function. Suppose that at a certain step, the polynomial $x_{\sigma(c)} + L$ is generated, where $\deg(L) \leq 1$. Then we substitute $x_{\sigma(c)}$ with L in the remaining polynomials and add $x_{\sigma(c)} + L$ to $\mathcal{A}$. Observe that this forms part of the ElimLin process (only the substitution part).

The Choose Function. As discussed in Remark 2, we continue to pick polynomials to split until a monic triangular set is obtained or an inconsistency is reached. In the triset function of CSA, we simply pick the next polynomial to be one with the smallest class in the set. To improve efficiency, it was suggested in [18] to sort the polynomials according to some complexity metric and to choose the "simplest" polynomial among the remaining polynomials to split. The rationale is to seek for more linear polynomials which will help to simplify other polynomials by linearization described above. The complexity metric proposed in BCSA orders the polynomials as follows: A polynomial P is simpler than a polynomial P' if $(\deg(P), \#I(P), \#U(P), n - 1 - \mathrm{cls}_\sigma(P)) < (\deg(P'), \#I(P'), \#U(P'), n - 1 - \mathrm{cls}_\sigma(P'))$ in a lexicographical manner, where $I(P)$ (resp. $U(P)$) and $I(P')$ (resp. $U(P')$) represent the left child (resp. right child) of P and P' respectively. Note that the description of the choose function in [18] with respect to the class of the polynomials is in a reverse order due to the difference in the definition of the class of a polynomial.

The Threshold Value. By Lemma (1), whenever we split a polynomial, we obtain a monic polynomial with the same class. In the triset function of CSA, we first split all the polynomials of the same class c to obtain all the monic polynomials with class c. We choose one of the monic polynomials P_0 to be in $\mathcal{A}$ and then add P_0 to all the monic polynomials to eliminate the class c from the remaining polynomials (the adding step).

Since BCSA splits polynomials in order of their complexity instead of their classes, it is not efficient to first obtain all the monic polynomials before adding them. As such, the authors of [18] introduced a threshold value t such that whenever t monic polynomials of the same class are obtained, the adding process will be executed to eliminate the class from $t - 1$ of these polynomials.

3 The Enhanced Binary CSA

In this paper, we propose additional techniques to enhance BCSA. Our version will be called EBCSA. Like BCSA, we keep the core of the structure, namely, EBCSA comprises the EBCSA-triset function and the main EBCS function, as shown in Fig. 1, where the new features introduced in EBCSA are marked in bold. Its basic structure is described in Algorithm 2 in the appendix. In the subsequent sections, we will describe the main features of EBCSA.

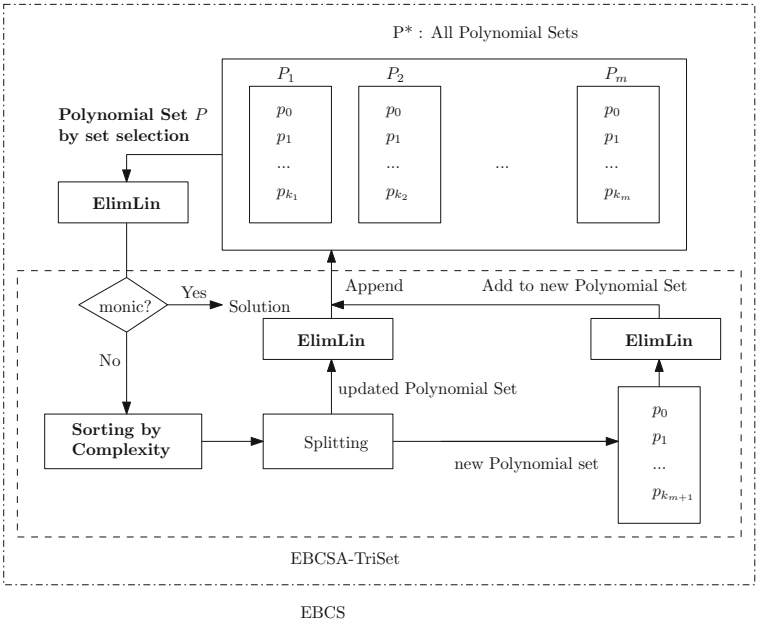


Fig. 1. Principles of the enhanced binary characteristic set algorithm

3.1 The ElimLin Technique

The first main feature of EBCSA is to integrate ElimLin into EBCSA-triset [11,12]. Specifically, whenever a new polynomial is generated in $\mathcal{P}$, we perform the following as long as the generated set contains a linear polynomial. First, order the monomials in decreasing order of degrees (for example, the Grevlex ordering). With respect to this monomial ordering, represent each polynomial as a vector of coefficients. Next, perform Gaussian elimination on the matrix of these vectors. If 1 is obtained, one can terminate EBCSA-triset with $\mathcal{A} = \emptyset$ and $\mathcal{P}^*$. Otherwise, obtain all the linear polynomials $x_i + L_i$ and substitute x_i with L_i in the remaining polynomials of $\mathcal{P}$ for all such i's.

A key merit of this process is that it allows us to detect all the linear polynomials in the vector space spanned by the polynomials [12], thereby simplifying the remaining polynomials more quickly. At the same time, it detects an inconsistent set more rapidly. However, the main drawback is that it generally requires more memory to construct the matrix and may not be feasible when our polynomial set contains too many polynomials.

3.2 An Improved Complexity Metric

In BCSA, a complexity metric was introduced in order to sort the polynomials. With respect to this metric, the next simplest polynomial will be picked to perform the splitting. Their metric compares the degree of the polynomials, the

number of terms in the left child and the right child of the polynomials as well as the classes of the polynomials in a linear manner. We propose a better metric based on the Sigmoid function.

For a child $\mathbf{p}$ of a polynomial $\mathbf{P}$, we define its complexity as

$$complex\,(x_1, x_2) = x_1 + Sigmoid\,(x_2, K) = x_1 + \frac{1 - e^{\frac{x_2}{K}}}{1 + e^{\frac{x_2}{K}}} \tag{1}$$

where $x_1 = \deg(\mathbf{p})$ determines the overall complexity and $x_2 = \#(\mathbf{p})$ counts the number of terms in $\mathbf{p}$ and it is normalized by the Sigmoid function to the interval $(0, 1)$. Here $K = -100$ is a scaling parameter which is optimized experimentally. Then the complexity of the polynomial $\mathbf{P} = I \cdot x_{\sigma(c)} + U$ is defined as

$$Complex(\mathbf{P}) \;=\; Complex(\mathbf{I}) + \alpha\,(\mathbf{P}) \cdot Complex(\mathbf{U}) \tag{2}$$

and

$$\alpha\,(\mathbf{P}) = c^{a \cdot \deg(\mathbf{P}) - b} = 10^{3 \cdot \deg(\mathbf{P}) - 8} \tag{3}$$

where $Complex(\mathbf{p}) = complex(\deg(\mathbf{p}), \#(\mathbf{p}))$, and a, b and c are set to be 3, 8 and 10, respectively. $\mathbf{p}$ is taken as either $\mathbf{I}$ or $\mathbf{U}$. Note that as the degree of the polynomial system increases, the complexity of the child $\mathbf{U}$ tends to be more emphasized. Thus the coefficient $\alpha(\mathbf{P})$ is defined as an exponential function of the degree. For instance, when $\deg(\mathbf{P})$ is 2, 3 and 4, the weighting coefficient $\alpha(\mathbf{P})$ is 0.01, 10 and 10000, respectively.

In our model, the Sigmoid function is chosen as a candidate because it has good properties, namely, it is a bounded differentiable real-valued function in $(0, 1)$ that is defined for all real input values and has a positive derivative at each value. The standard Sigmoid model is $Sigmoid\,(x) = \frac{1}{1+e^{\frac{x-x_0}{K}}}$, and we are using a modified version $Sigmoid\,(x) = \frac{1-e^{\frac{x}{K}}}{1+e^{\frac{x}{K}}}$. The complexity model in Eq. 2 is a combination of linear model and the modified Sigmoid model, and it preserves the good properties of the standard model. Similar logistic functions have been effectively used in population growth modelling, artificial neural networks and distance measure.

To select the best model mentioned above, we have tried many possible parameters for each model. Preliminary experimental results show that the modified Sigmoid model $Sigmoid\,(x) = \frac{1-e^{\frac{x}{K}}}{1+e^{\frac{x}{K}}}$ exhibits the best overall performance on our training sets. The training sets include 120 randomly selected polynomial sets from more than 300 available polynomial sets generated from many different ciphers with degrees ranging from 2 to 4, and the rest of the sets are the testing sets. To determine the optimal parameters of each model, a 10-fold cross-validation [19] is applied. Hence, the parameters are dependent on the specific cipher as well as the selected features (degree and number of terms).

In order to reduce the computational cost when selecting the optimal parameters, the parameters are confined to a limited range by preliminary observations.

Then the parameter setting (K, a, b, c) in Eqs. 1, 2 and 3 is determined by the cross-validation procedure as follows: we split the training sets into 10 equal sized parts; using 9 parts we fit the parameters, that is, record the parameter setting that produces the best performance; calculate its performance on the remaining 1 part as the validation set. We repeat these steps by using every part as the validation set. We repeat the whole procedure 3 times. Finally we get an average of the 3 solving times which corresponds to each parameter setting. We choose the setting which corresponds to the minimum solving time. For each of the models mentioned above, we repeat this procedure. As a result, the optimal parameter setting is determined as $(K, a, b, c) = (-100, 3, 8, 10)$. The parameters are not very sensitive, that is, when they are modified by around 10%, the solving time remains within a 10% change. This parameter setting is not guaranteed to be effective for all individual polynomial sets, but is optimal in the sense of overall performance.

By contrast, the choose function in BCSA uses a layered linear model with respect to the essential parameters. This model suffers from the following drawbacks. (1) There are no parameters to control, limiting the possibility to optimize the performance; (2) The "layered" comparison sometimes cannot get reasonable results, e.g. a polynomial with a higher degree but very few terms is not necessarily more complex than a polynomial with a lower degree but the number of terms is huge. As such, the proposed EBCSA complexity model has more flexibility to produce better performance by optimization over a number of polynomial sets. Indeed, we have carried out extensive experiments using different functions and the above function works well in general. We present our experimental results for solving Canfil functions as comparison between using the modified Sigmoid function and the choose function of BCSA in Subsect. 4.1.

3.3 The Set Metric

We introduce a set metric on the sets in the binary tree $\mathcal{P}^*$ to help us find a set with a solution more quickly. Hence, this feature is particularly useful if the system admits a known number of solutions, as well as the case where only a fixed number of solutions is required.

Observe that polynomial sets in the same branch may contain many identical polynomials. Roughly speaking, once many sets in a certain branch terminate without a solution, we like to choose sets from a different branch to increase the chances of a solution. In other words, we will look for a set which is most dissimilar to the previous terminated sets.

For the purpose of practical implementations, we propose the following statistical feature for a set $\mathcal{P}$ of polynomials: $S_{\mathcal{P}} = (m_1, s_1, m_2, s_2)$, where m_1, m_2 (respectively s_1, s_2) represent the mean (respectively the standard deviation) of the degrees and number of terms of the polynomials in $\mathcal{P}$ respectively. Suppose that $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_l$ are inconsistent, that is, terminate without any solution. Let $a(\mathcal{P}_1, \ldots, \mathcal{P}_l) = \frac{s_{\mathcal{P}_1} + \ldots + s_{\mathcal{P}_l}}{l}$, the average of the statistical features of the terminated sets. Then we choose a set $\mathcal{P}$ such that the Euclidean distance between $s_{\mathcal{P}}$ and $a(\mathcal{P}_1, \ldots, \mathcal{P}_l)$ is the largest on the fly.

3.4 Sorting the Variables

In our description of binary CSA, we have introduced the permutation σ to allow for the variables to be rearranged. In general, CSA tends to be more efficient if σ is chosen so that the variables $x_{\sigma(0)}, \ldots, x_{\sigma(x_{n-1})}$ occur in an increasing order of frequency in the polynomial system. The underlying reasons are as follows: recall that when performing linearization, we substitute a variable x_i with a linear polynomial L_i. While this reduces the number of variables, it may result in the polynomial system becoming more complex, particularly when L contains many terms. Hence, it is more advisable to substitute those x_i which do not occur too frequently. In terms of implementation, we simply rename the variables so that $x_0, x_1, \ldots, x_{n-1}$ occur in an ascending order of frequency in the polynomial system.

3.5 Combining the Various Features

We observe from our experiments that not all polynomial systems behave equally well with the same set of features. According to experimental results, some sets may be more efficient with linearization alone while others perform much better with the ElimLin technique. In addition, the set selection may not be necessary if the system has many solutions and we seek to find all of them. In view of the above, EBCSA allows for different combinations of features to yield the best performance. In our current implementation of EBCSA, the following choices are provided: (1) Linearization or ElimLin; (2) Set selection based on statistical feature or set index; and (3) Terminate with one solution or all solutions.

Remark 4. Observe that unlike BCSA, we do not need a threshold value here as whenever we obtain two monic polynomials with the same class, we add them and choose the simpler polynomial according to our complexity metric to be in the set $\mathcal{A}$. In addition, in case the ElimLin option is turned on, polynomials with the same leading term will automatically be added.

4 Experimental Results

We tested EBCSA with many different polynomial systems, specifically those from block and stream ciphers. Our experiments show significant improvements over existing algebraic methods. In this paper, we present three such experiments, namely, the block ciphers Present and Prince as well as the stream cipher Toyocrypt. Experiments on Canfil ciphers were also carried out to compare the efficiency of EBCSA against CSA and BCSA. Since our goal is to find the unknown key which is likely unique, we use the following configurations of EBCSA for all the experiments: Linearization/ElimLin, set selection, and termination when one solution is obtained.

The platform for testing is a personal computer (Ubuntu 14.04, Intel Xeon CPU E5640 2.67 GHz, 24 GB RAM). Only single-core CPU is used. In the following, we further classify EBCSA into EBCSA and EBCSA-GE (short for EBCSA with Gaussian Elimination), where the former refers to EBCSA with linearization and the latter representing EBCSA with the ElimLin feature switched on.

4.1 Experiments on Our Complexity and Set Metrics

In order to benchmark EBCSA against BCSA and CSA, we performed experiments on the Canfil functions similar to those carried out in [17,18]. The Canfil polynomial sets are boolean polynomial systems arising from linear feedback functions with nonlinear Canfil functions. In [17, Table 4], comparisons were made between CSA and the Gröbner basis algorithm for 8 such functions and the results showed that CSA outperformed the Gröbner basis method for these polynomial sets. In [18, Table 2], the authors recorded the number of splits (or branches) resulting from both CSA and BCSA for these same functions and concluded that BCSA was more effective on these sets.

In this paper, we carry out the same experiments using EBCSA and the number of splits (number of new non-terminating sets produced) for each of the 8 Canfil functions is recorded in Table 1. In addition, the timing on the Canfil functions using the approach of EBCSA with set selection is also listed.

Table 1. Comparison between EBCSA and BCSA in terms of the number of splits

#Splits\system	Canfil1	Canfil2	Canfil3	Canfil4	Canfil5	Canfil6	Canfil7	Canfil8
CSA	13719	23881	7251	1657	1086	3331	1551	180710
BCSA	11958	16074	3267	1316	574	671	1852	35547
EBCSA w.o set selection	5218	24616	5402	782	408	400	3970	90040
EBCSA w. set selection	5772	17299	3865	343	100	209	1046	35565
EBCSA time (s)	82.9	629.5	669.7	8.8	56.4	41.7	57.5	501.3

Observe that EBCSA performs better than BCSA in 5 out of the 8 Canfil experiments with significant improvements for Canfil1, Canfil4, Canfil5 and Canfil6. For Canfil7, CSA outperforms BCSA but is less effective than EBCSA (with set selection). Note that we have used a random key in our experiments and it was not stated if a random or a specific key was used in the experiments in [17,18].

The improvements of EBCSA suggest that our choice of a combined continuous Sigmoid function for polynomial complexity is more effective than the layered linear model used in BCSA. In addition, the use of set selection results in finding the unique solution more efficiently.

4.2 Experiments on Present Cipher

Present is a 31-round lightweight block cipher designed by Bogdanov et al. announced in CHES 2007 [20]. It has a block length of 64-bits and key lengths of either 80-bits or 128-bits. It is based on a permutation-substitution network with 16 4-bit S-boxes in each round. In [14], algebraic cryptanalysis was applied

to Present with the 80-bit master key. This involves representing the intermediate text by variables and each S-box by 21 quadratic equations. The authors claimed that with 5 known plaintext/ciphertext (KP) pairs, 40 key bits can be recovered by the method of ElimLin.

In the report of [21], results on algebraic attacks on Present with ElimLin and the Polybori Gröbner basis implementation were presented. Even though ElimLin was implemented as a sub-routine in Polybori implementation, it was demonstrated that the stand-alone ElimLin implementation was more effective to solve Present. In addition, it was reported that 6 rounds of Present remained resistant to algebraic cryptanalysis using either ElimLin or Polybori.

As EBCSA-GE incorporates ElimLin into EBCSA, we perform experiments on round-reduced Present with EBCSA-GE. Once again, we consider the implicit representation of each s-box with 21 quadratic equations. Note that we place all the text variables as the front variables while the 80 key variables are placed behind. We perform our experiments with either 1 or 2 KP. We considered r rounds of Present for $r = 5$ and $r = 6$. For experiments with one KP, we did some pre-processing on the polynomial systems before running them on EBCSA-GE. Specifically, we rename the variables so that the text variables and the key variables are sorted in an ascending order of frequency. In each of the experiments, we fix $x_{n-c-1}, x_{n-c}, \ldots, x_{n-1}$, (that is, c most frequent key variables) and record the time taken as well as the number of splits as c varies.

Table 2. 5-round and 6-round present with 1 KP on EBCSA-GE

r	#FixedBits	64	60	56	54	52	50	48	44	40	38	36	34
5	Time (s)	<1	<1	<1	<1	<1	3.89	5.46	58.6	237.3	161.8	2148.7	>4 h
5	#Splits	0	0	0	0	0	3	16	174	710	556	6200	–
6	Time (s)	7.1	81.0	378.9	1979.2	1489.0	>4 h	>4 h	>4 h	>4 h	>4 h	>4 h	>4 h
6	#Splits	24	216	1234	6163	2172	–	–	–	–	–	–	–

Table 3. 5-round present with 2 KP on EBCSA-GE

#FixedBits	40	36	34	32	30	28	26
Time (s)	9.5	300.1	282.9	1562.9	888.4	2120.8	>4 h
#Splits	6	191	172	974	563	1362	–

One sees that EBCSA-GE outperforms both ElimLin and the Polybori implementation of [21] in the following sense. From Table 4 with 40 keybits fixed, EBCSA-GE requires only 1 KP to solve in 4 min, which is of comparable magnitude to the timings of ElimLin and Polybori based on 5 KP's. With just 2 KP's, EBCSA-GE already outperforms ElimLin and Polybori, with a solving time of 9.5s. Next with 36 keybits fixed, EBCSA-GE can solve the system in a few minutes based on just 2 KP's while ElimLin and Polybori both take longer to solve (3 to 4 h) and need more plaintexts (16 KP's). From Table 3 with 2

Table 4. A comparison of algebraic attacks on 5-round present

	Polybori [22]		Elimlin [22]		EBCSA-GE			
#FixedBits	40	36	40	36	40	40	36	36
#KP	5	16	5	16	1	2	1	2
Time (s)	241.2	>4 h	154.8	12.2	237.3	9.5	2148.7	300.1

KP, EBCSA-GE can solve up to 52 keybits (28 fixed keybits) for 5 rounds of Present, which is more keybits than what ElimLin and Polybori can solve from [21]. Finally from Table 2, EBCSA-GE can solve 6 rounds of Present with 1 KP and 52 fixed keybits, while the ElimLin and Polybori experiments from [21] are only conducted up to 5 rounds.

Recall that ElimLin essentially seeks to find linear polynomials in the space spanned by the polynomials and then eliminating their leading variables from the remaining polynomials via substitution. This method can only be successful if there are sufficient linear polynomials in the spanning set. This implies that more polynomials involving the key variables in the original set increases the chance of finding enough linear polynomials and hence, leading to a greater number of KPs as observed. In case there are insufficient linear polynomials, Gröbner basis may not be effective as an increase in the degree of the polynomials with a large number of variables will explode the memory. On the other hand, EBCSA-GE executes the splitting process without a rapid increase in memory and continues to find linear polynomials via ElimLin. By alternating between splitting and ElimLin processes, one sees that EBCSA-GE is more effective for Present cipher.

4.3 Experiments on Prince Cipher

Prince is another lightweight block cipher published in Asiacrypt 2012 [22]. It is a 12-round block cipher with a 64-bit block length and a 128-bit key length. It is optimized with respect to latency for hardware implementations. Its construction satisfies the reflection property and is symmetric about the middle layer. The cryptanalysis of Prince has gained much attention, especially after a challenge was posed by Ruhr-Universität Bochum [23] to find a practical attack for round-reduced Prince. To date, it was shown that round-reduced Prince is vulnerable to various attacks including Differential and Integral cryptanalysis as well as time-memory trade-off and meet-in-the-middle attacks [24–28]. To the best of our knowledge, no algebraic attack has been published on Prince.

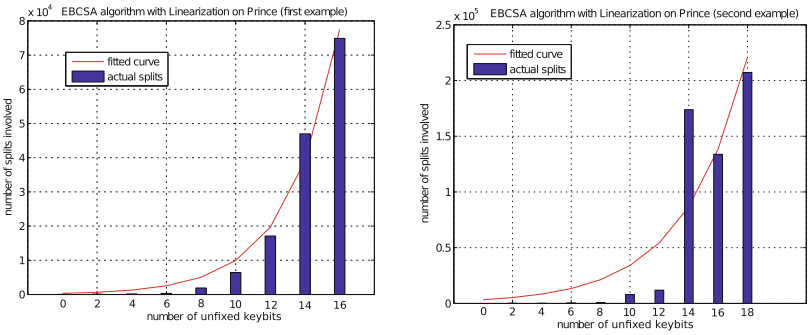
We carry out the first algebraic attack on Prince. Specifically, we consider Prince-core, that is, without whitening. Hence, only the key K_1 is used. As in all block ciphers, we consider the implicit representation of the s-boxes, thereby resulting in 21 quadratic equations for each s-box. In addition, we introduce 64 intermediate variables for each round. Since whitening is not used, the variables for the first round and the last two rounds can be omitted.

Now, consider the 6-round Prince-core. This system has 384 variables (320 text variables and 64 key variables) and 2016 quadratic equations. We place the key variables behind $(x_{320}, \ldots, x_{383})$. We generate 2 different systems using different random keys. For each key, we fix some key bits and run the system with our EBCSA solver. First, Table 5 records the results for Prince with both EBCSA and EBCSA-GE.

Table 5. 6-round Prince-core: EBCSA vs EBCSA-GE

#Fixed bits	64	56	52	48	40
#Splits of EBCSA	53	533	11415	53147	–
EBCSA time (s)	4.3	54	1400	**5900**	>4 h
EBCSA-GE time (s)	**2**	**266**	**784**	>4 h	>4 h

Observe that when more bits are fixed, both EBCSA and EBCSA-GE should be executed in parallel to find the key as it is not certain which of the two versions will be more efficient. However, when fewer bits are fixed, EBCSA-GE tends to require much more memory to generate the large matrices, thereby slowing down the process significantly. In the following tables, we show the results of EBCSA for two random keys by varying the number of fixed keybits. By fitting the results with a suitable graph, we extrapolate the results to estimate the complexity without fixing any key bits. We conclude that the number of splits of Prince-core is generally better than the brute force $f = 2^x$, where x denotes the number of unknown keybits, as shown in Fig. 2. The results are also listed in Table 6, where the complexity of EBCSA is expected to provide an improvement of more than half the key size over the brute force search. We remark that this approach can be applied to test the complexity of other block ciphers as well.



(a) (b)

Fig. 2. Splits of Prince-core. (a) $f = 324.19 \times 2^{0.494x}$ (b) $f = 3287.44 \times 2^{0.337x}$

Table 6. Splits of Prince-core (a) $f = 324.19 \times 2^{0.494x}$ (b) $f = 3287.44 \times 2^{0.337x}$

#Fixed bits	64	62	60	58	56	54	52	50	48	46
(a)										
#Splits	45	105	167	301	1908	6415	17101	46994	74887	–
Linearization	3.51	6.89	11.66	21.44	138.27	471.83	1393.60	4010.22	5699.43	>4 h
(b)										
#Splits	49	106	121	433	888	8012	11893	173878	133736	207162
Linearization	3.52	6.51	8.03	29.97	67.09	599.15	956.47	12309.5	10288.5	15156.6

The comparison with the existing attack on 6-round Prince core from [24] is provided in Table 7. It is obvious that the proposed EBCSA algorithm is more practical in terms of required number of chosen plaintext.

Table 7. Comparison of existing attacks on 6-round Prince-core

Source	Data	Time	$D \times T$	Memory	Attacks
FSE13 [24]	2^{16}	2^{30}	2^{46}	2^{16}	Integral
Proposed EBCSA	1	$T \times K \approx 2^{60}$	2^{60}	2^{15}	First algebraic attack by EBCSA

Remark 5. **D**ata: number of chosen plaintext; **T**ime: equivalent Prince operations; complexity of attacks on Prince is measured by $D \times T < 2^{128}$; Memory: Number of Prince 64-bit WORDs.

4.4 Experiments on the Stream Cipher Toyocrypt

The Toyocrypt [29] is a stream cipher comprising a 128-bit Galois linear feedback shift register (GLFSR) passing through a nonlinear function f of degree 63. Obviously, a naive application of algebraic cryptanalysis will not work due to the high degree of f. In [30], the authors discovered that in fact, $(x_{23} + 1)f$ and $(x_{42} + 1)f$ are cubic polynomials. Further, in [31], the authors showed that with re-synchronization and 4 IV's, the degrees of the equations can be further reduced to 1. Note that in this case, the 4 IV's must span a 2-D vector space.

Several algebraic attacks on Toyocrypt have been proposed. Table 8 summarizes the assumptions and requirements needed to carry out existing algebraic attacks on Toyocrypt.

The attack of [30] requires 2^{18} keystream bits for one session, while the attack of [31] requires 128 keystream bits from 4 chosen IV resynchronizations. These attack scenarios may not be easy to achieve in practice. Here we consider a more realistic scenario where only a few keystream bits can be deduced from some known information about the plaintext. Besides, we allow the IV in different sessions to be selected randomly. Finally, we apply our attack to Toyocrypt with GLFSR as in the original specifications of the cipher unlike previous attacks which only considered Toyocrypt with LFSR as a good approximation.

Table 8. Comparison of assumptions on algebraic attacks on Toyocrypt

Source	Resynchronization needed?	No. of keystream-bits	Remarks on IV
EUROCRYPT 2003 [30], CRYPTO 2004 [32]	No	2^{18}	1 session with random IV
SAC 2004 [33]	Yes	2^{13}	Chosen IVs
CANS 2009 [31]	Yes	128	Chosen IVs
Proposed EBCSA	Yes	≤ 6	Random IVs

To be more precise, we fix a GLFSR with feedback taps $0, 1, 2, 7, 128$. For any 2 IV's,it follows from [31, Theorem 1] that the sum of $(x_{23}+1)f$ (or $(x_{42}+1)f$) at these two IV's result in equations of degree 2. Instead of choosing some fixed IV's, we randomly choose k IV's and construct the corresponding equations for $(x_{23}+1)f + z(x_{23}+1)$ and $(x_{42}+1)f + z(x_{42}+1)$, where z represents the keystream bit. Notice that these are degree 3 equations but the sum of any two of them will give a degree 2 equation.

We perform our experiments with at most 6 keystream bits using EBCSA-GE and record the results of our experiments in Table 9. In the preprocessing stage, we once again arranged the 128 variables ascendingly of their frequencies.

Table 9. Performance of Toyocrypt with EBCSA-GE

#Key stream	2	2	2	2	2	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	6
#IV	128	120	112	104	96	128	120	112	104	96	92	128	120	112	104	96	92	88	84	80	80
#Poly	512	480	448	416	384	768	720	672	624	576	552	1024	960	896	832	768	736	704	672	640	960
#Splits	0	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	2	2	2
#Elim Lin	6	9	8	7	11	4	8	8	6	8	7	4	6	5	8	5	9	9	13	13	11
Time (s)	7.1	41.0	318.5	24.1	32.7	4.7	19.4	17.1	178.7	59.4	516.4	7.4	134.0	413.0	76.1	79.7	71.1	97.0	542.1	669.3	187.4

5 Conclusions and Future Work

In this paper, we enhanced the binary characteristic set algorithm to work more effectively on both block ciphers and stream ciphers. Using EBCSA, we improve algebraic attacks on various well-known ciphers including Prince, Present and Toyocrypt. EBCSA uses CSA as its core and incorporates ElimLin and some statistical features to improve its performance. In our current version, we can turn ElimLin on or off for each input polynomial system. For future research, we will seek to identify some characteristics of the polynomial system to choose between EBCSA-GE or EBCSA. We can further integrate the criteria into the main algorithm so that ElimLin can be turned on or off for each individual set produced by EBCSA.

Acknowledgements. We are grateful to Dr. Matt Henricksen, Dr. Yap Wun She, Dr. Lee Hian Kiat and Ms. Ivana Thng for their helpful contributions throughout the project.

Appendix: Pseudocodes for CSA and EBCSA

Next, we present the main structure of EBCSA in Algorithm 2. Note that pre-processing of the input set can be carried out to sort the variables in a desired order, that is, we fix a permutation σ beforehand.

Algorithm 1. The CSA (or MFCS) algorithm [**17**]

- Set $\mathcal{P}^* = \emptyset$ and $\mathcal{A}^* = \emptyset$.
- CSA-triset function – Feed $\mathcal{P}$ into triset function and let $\mathcal{Q}^*$ and $\mathcal{A}$ be outputs:
  - Set $\mathcal{Q}^* = \emptyset$ and $\mathcal{A} = \emptyset$.
  - For $c = 0$ to $n - 1$ do:
    - If $1 \in \mathcal{P}$, return $\mathcal{A} = \emptyset$ and $\mathcal{Q}^*$.
    - Let $\mathcal{Q} = \{P \in \mathcal{P} : \text{cls}(P) = c\}$, set $\mathcal{P} = \mathcal{P} \backslash \mathcal{Q}$, set $\mathcal{Q}_1 = \emptyset$.
    - While $\mathcal{Q} \neq \emptyset$ do
      - Pick $P \in \mathcal{Q}$ and set $\mathcal{Q} = \mathcal{Q} \backslash \{P\}$.

    Splitting step Write $P = Ix_{\sigma(c)} + U$. Add $\mathcal{P} \cup \mathcal{Q} \cup \{I, U\}$ to $\mathcal{Q}^*$.
      - Add $I + 1$ to $\mathcal{P}$ and add $x_{\sigma(c)} + U$ to $\mathcal{Q}_1$.
    - Pick $P \in \mathcal{Q}_1$ and add P to $\mathcal{A}$.

  Adding step For every $P' \in \mathcal{Q}_1, P' \neq P$, add $P + P'$ to $\mathcal{P}$.
- Set $\mathcal{P}^* = \mathcal{P}^* \cup \mathcal{Q}^*$ and add $\mathcal{A}$ to $\mathcal{A}^*$.
- Repeat the whole procedure till $\mathcal{P}^*$ is empty.

Algorithm 2. Algorithm EBCSA

- **Input**: a set of polynomials $\mathbb{P} = \{p_1, \cdots, p_k\}$.
- **Outputs**: A monic triangular set $\mathbb{A}$ and a super set $\mathbb{P}^*$ for other potential solutions.
- **EBCSA-triset** function
  - Begin with the simplest P, e.g. $P_k = Ix_{c_k} + U \in \mathbb{P}$.
  - Let $\mathbb{P}' = \mathbb{P} \backslash \{P_k = Ix_{c_k} + U\}$. Split P_k: $\mathbb{P} \leftarrow \mathbb{P}' \cup \{I + 1\}$, $\mathbb{A} \leftarrow \mathbb{A} \cup \{x_{c_k} + U\}$ and $\mathbb{P}_{new} \leftarrow \mathbb{P}' \cup I \cup U$, append $\mathbb{P}_{new}$ to $\mathbb{P}^*$.
  - Process each new generated $\mathbb{P}_{new}$ or updated polynomial set $\mathbb{P}$, and terminate them as early as possible.
- Continue with every P until $\mathbb{P}$ is terminated or becomes the monic set $\mathbb{A}$.
- Select a new set $\mathbb{P}$ from $\mathbb{P}^*$ to apply the above algorithm.

References

1. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996). doi:10.1007/3-540-68339-9_4
2. Buchberger, B.: Ein algorithmus zum auffinden der basiselemente des restklassenrings nach einem nulldimensionalen polynomideal. Universitat Innsbruck, Austria, Ph.D. thesis (1965)
3. Faugere, J.C.: A new efficient algorithm for computing gröbner bases (f4). J. Pure Appl. Algebra **139**(1), 61–88 (1999)
4. Faugere, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: Proceedings of ISSAC, ACM, pp. 75–83 (2002)
5. Faugere, J.C., Gianni, P., Lazard, D., Mora, T.: Efficient computation of zero-dimensional Gröbner bases by change of ordering. J. Symb. Comput. **16**(4), 329–344 (1993)
6. Cox, D., Little, J., O'shea, D.: Ideals, Varieties, and Algorithms, vol. 3. Springer, New York (1992)
7. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE public key cryptosystem by relinearization. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_2
8. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000). doi:10.1007/3-540-45539-6_27
9. Courtois, N.T., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002). doi:10.1007/3-540-36178-2_17
10. Courtois, N.T., Patarin, J.: About the XL algorithm over $GF(2)$. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 141–157. Springer, Heidelberg (2003). doi:10.1007/3-540-36563-X_10
11. Courtois, N.T., Bard, G.V.: Algebraic cryptanalysis of the data encryption standard. In: Galbraith, S.D. (ed.) Cryptography and Coding 2007. LNCS, vol. 4887, pp. 152–169. Springer, Heidelberg (2007). doi:10.1007/978-3-540-77272-9_10
12. Courtois, N.T., Sepehrdad, P., Sušil, P., Vaudenay, S.: ElimLin algorithm revisited. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 306–325. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34047-5_18
13. Indesteege, S., Keller, N., Dunkelman, O., Biham, E., Preneel, B.: A practical attack on KeeLoq. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 1–18. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78967-3_1
14. Nakahara Jr., J., Sepehrdad, P., Zhang, B., Wang, M.: Linear (Hull) and algebraic cryptanalysis of the block cipher PRESENT. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 58–75. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10433-6_5
15. Aubry, P., Lazard, D., Maza, M.M.: On the theories of triangular sets. J. Symb. Comput. **28**(1), 105–124 (1999)
16. Kalkbrener, M.: A generalized euclidean algorithm for computing triangular representations of algebraic varieties. J. Symb. Comput. **15**(2), 143–167 (1993)
17. Fengjuan, C., Xiao-Shan, G., Chunming, Y.: A characteristic set method for solving boolean equations and applications in cryptanalysis of stream ciphers. J. Syst. Sci. Complex. **21**(2), 191–208 (2008)

18. Huang, Z., Sun, Y., Lin, D.: On the efficiency of solving Boolean polynomial systems with the characteristic set method. arXiv preprint (2014). arXiv:1405.4596
19. Kohavi, R., et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. Int. Jt. Conf. Artif. Intell. **14**(2), 1137–1145 (1995)
20. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74735-2_31
21. Sepehrdad, P.: Statistical and algebraic cryptanalysis of lightweight and ultra-lightweight symmetric primitives. Ph.D. thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE (2012)
22. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE – a low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34961-4_14
23. Bochum, R.U.: The PRINCE Challenge. https://www.emsec.rub.de/research/research_startseite/prince-challenge/ (2014). Accessed 18 Jan 2017
24. Jean, J., Nikolić, I., Peyrin, T., Wang, L., Wu, S.: Security analysis of `PRINCE`. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 92–111. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43933-3_6
25. Dinur, I.: Cryptanalytic time-memory-data tradeoffs for FX-constructions with applications to PRINCE and PRIDE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 231–253. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46800-5_10
26. Derbez, P., Perrin, L.: Meet-in-the-middle attacks and structural analysis of round-reduced PRINCE. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 190–216. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48116-5_10
27. Canteaut, A., Fuhr, T., Gilbert, H., Naya-Plasencia, M., Reinhard, J.-R.: Multiple differential cryptanalysis of round-reduced PRINCE. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 591–610. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46706-0_30
28. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: improved MITM attacks. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 222–240. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40041-4_13
29. Mihaljevic, M.J.: Cryptanalysis of toyocrypt-HS1 stream cipher. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **85**(1), 66–73 (2002)
30. Courtois, N.T., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003). doi:10.1007/3-540-39200-9_21
31. Zhang, A., Lim, C.-W., Khoo, K., Wei, L., Pieprzyk, J.: Extensions of the cube attack based on low degree annihilators. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 87–102. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10433-6_7
32. Hawkes, P., Rose, G.G.: Rewriting variables: the complexity of fast algebraic attacks on stream ciphers. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 390–406. Springer, Heidelberg (2004). doi:10.1007/978-3-540-28628-8_24
33. Armknecht, F., Lano, J., Preneel, B.: Extending the resynchronization attack. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 19–38. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30564-4_2