Journal of Cryptographic Engineering manuscript No.
(will be inserted by the editor)

Improved Algebraic Attacks on Lightweight Block Ciphers

Sze Ling Yeo · Duc-Phong Le · Khoongming Khoo

the date of receipt and acceptance should be inserted later

Abstract This paper proposes improved algebraic attacks that are effective for lightweight block ciphers. Concretely, we propose a new framework that leverages on algebraic preprocessing as well as modern SAT solvers to perform algebraic cryptanalysis on block ciphers. By combining with chosen plaintext attacks, we show that our framework can be applied to lightweight block ciphers that exhibit a nice differential trail. In particular, we demonstrate our techniques by performing algebraic cryptanalysis on both the Present cipher and the Simon cipher. For the Present cipher, we successfully solved up to 9 rounds with at most 32 key bits fixed and 8 chosen plaintexts. On the other hand, for the Simon cipher, we tested our method on Simon-32/64 and Simon-64/128. For these two versions, our attack can solve up to 13 rounds with only 8 chosen plaintexts by fixing 4 and 6 key bits for Simon-32/64 and Simon-64/128, respectively. Further, by considering a class of weak keys, we can extend our attacks to 16 rounds. As far as we are aware, these are the best algebraic attacks on these ciphers in the literature.

Keywords Algebraic Attacks · SAT solvers · Lightweight block ciphers · Simon · Present

1 Introduction

With the proliferation of low-powered devices such as smart cards, sensors and a booming IoT industry, the need for lightweight cryptography to protect the data stored on such devices becomes increasingly important. As such, different proposals have been put forward in recent years to design efficient and lightweight cryptographic primitives. These primitives are typically built from simple operations resulting in low latency and gate counts. Examples of lightweight stream ciphers include Grain [29] and Trivium [10], while Present [7] and Simon [3] are some well-known lightweight block cipher designs.

Even though an important goal of lightweight cryptography is to offer efficient implementations with limited computational resources, its core objective to provide security needs to be maintained, that is, the scheme should, at the minimum, resist all known attacks. At present, there are two main classes of attacks on block ciphers, namely, statistical approaches like differential cryptanalysis [5],

S-L. Yeo
Institute for Infocomm Research, 1 Fusionopolis Way, Singapore, 138632.
E-mail: slyeo@i2r.a-star.edu.sg

D-P. Le
Canadian Institute for Cybersecurity, UNB, NRC building, 46 Dineen Drive, Fredericton, Canada.
E-mail: le.duc.phong@unb.ca

K.M. Khoo
DSO National Laboratories, 12 Science Park Drive, Singapore, 118225.
E-mail: kkhoongm@dso.org.sg

linear cryptanalysis [34], integral attacks [21], invariant subspace attacks [32], as well as algebraic-based attacks [15,35]. The former is much better understood and good block cipher designs will be built to resist against such attacks. Typically, one seeks for non-trivial relationships among the plaintexts and ciphertexts. Such relationships are then used to distinguish between a random key and a correct key. A large amount of data is often required for this distinguishing property to be feasible. As such, good cipher designs can resist such attacks by ensuring that the amount of data required to produce the distinguishing relationships is beyond the available data.

On the other hand, algebraic cryptanalysis is more deterministic in the sense that it does not depend on any statistical property. Moreover, it generally requires fewer data and the complexity relies on the complexity of algebraic solving techniques to solve such systems. Algebraic attacks have been shown to be useful to break some stream ciphers such as Toyocrypt [16] and Crypto-1 [17]. while block ciphers with a high degree of non-linearity seem to be more resistant to algebraic attacks, there have been some successful attacks on lightweight block cipher designs with a low degree of multiplicity. with better algebraic representations of the ciphers and new breakthroughs in algebraic solving methods, algebraic attacks on block ciphers are becoming more feasible [12, 27,43]. By considering the cipher representation over finite field extensions, interpolation attacks have been designed to break some ciphers, including a recent attack on the cipher MiMC [33]. Finally, hybrid attacks have been designed that combine both statistical and algebraic attacks such as algebraic differential cryptanalysis [1].

This paper focuses on improving the state-of-the-art of algebraic cryptanalysis on lightweight block ciphers with a low degree of multiplicity over the binary field. Briefly, algebraic cryptanalysis comprises the following steps. First, given a known plaintext/ciphertext pair, one constructs a system of polynomial equations to relate the ciphertext with the plaintext in terms of the unknown key variables and possibly some intermediate variables. In general, there is a trade-off between the degree of the polynomials involved and the number of intermediate variables required. Typically, one often works with quadratic equations. Next, in order to find the unknown key bits, one will need to solve this system of equations. There are various approaches to solve a system of boolean polynomial equations, including ElimLin [15,20], XL and its generalizations [13,19,18], Gröbner basis algorithms [9,25,26], the characteristic set algorithm (CSA for short) [27,30,43] as well as by SAT solvers [2,42]. Each of these methods has its own merits and shortcomings and its efficiency is dependent on the structure of the system involved. For instance, Gröbner basis algorithms work better for dense systems while CSA and SAT solvers are more effective on sparse systems.

In this work, we seek to convert an algebraic system of polynomials with certain nice characteristics into a system that facilitates the solving process via modern SAT solvers. First, we present an ElimLin-SAT framework to carry out algebraic attacks on lightweight block ciphers. Concretely, one first finds a set of short linear equations by the ElimLin approach. One then adds this set of linear equations to the original set, thereby creating more equations without increasing the number of variables. Finally, this set is then converted into CNF clauses and fed into a modern SAT solver. In terms of algebraic cryptanalysis, we show how one can construct good differential trails such that plaintexts satisfying such trails can generate a large number of linear equations via ElimLin.

To demonstrate our techniques, we tested on two well-known lightweight block ciphers, the Simon cipher and the Present cipher. In each case, we identified good differential trails to choose the plaintext/ciphertext pairs. We experimentally carried out the attacks using random keys. Without fixing any key bits, we were able to break 7 rounds of the Present cipher and 12 rounds of both Simon-32/64 and Simon-64/128 by using not more than 12 plaintext/ciphertext pairs. By fixing up to 32 key bits, we further improved our attack to break 9 rounds of Present cipher. Our attack on Simon can be extended to 13 rounds when not more than 6 key bits were fixed. Moreover, by considering a class of weak keys, our attacks improved to 15 and 16 rounds for Simon-32/64 and Simon-64/128, respectively.

Next, we proposed some enhancements to the ElimLin-SAT approach. To this end, we introduced higher-order differentials and show that more linear relations can be generated from such differentials. By adding in higher degree equations representing the differential equations among a subspace of plaintexts, we show that we can now break 16 rounds of Simon-32/64 for a class of weak keys and with only 8 chosen plaintexts. To our best knowledge, our attacks surpass the existing results of algebraic cryptanalysis on both Simon and Present ciphers.

This paper is structured in the following way. In the next section, we recall the overall framework of algebraic cryptanalysis and review two main approaches, namely ElimLin and SAT solver techniques. In addition, we discuss some existing approaches that are closely related to our work. In Section 3, we present our improved algebraic technique by combining ElimLin with SAT solvers. We then provide our experimental results for the Simon cipher and the Present cipher in the following two sections. In Section 6, we investigate the linear equations that arise from the ElimLin process on the Simon cipher. By introducing the differential representation of two plaintexts, we improve our attack on Simon-32/64 in Section 7. Finally, we wrap up our work with some concluding remarks and suggestions for future work.

2 Preliminaries and related work

2.1 Solving algebraic equations via ElimLin and SAT solvers

Let $S = \{f_1, \ldots, f_m\}$ be a set of m polynomials over the binary field $\mathbb{F} = GF(2)$ in n variables $x_1, x_2, \ldots, x_n$. Our goal is to find some $y_1, \ldots, y_n \in \mathbb{F}$ such that $f_i(y_1, \ldots, y_n) = 0$ for all $i = 1, 2, \ldots, m$. Here, recall that we have $x_i^2 = x_i$ for all $i = 1, \ldots, n$. For a polynomial f_i of degree that is greater than 2, one can often introduce more variables to convert it to a polynomial of degree at most 2. Thus, for simplicity and for applications to cryptography, we will assume that each polynomial f_i is quadratic.

2.1.1 The ElimLin process

The ElimLin algorithm was first formally introduced by Courtois in [15] to solve the Data Encryption Standard (DES) and subsequently revisited in [20]. This simple technique involves the following steps:

– Find all the linear polynomials in the vector space spanned by the f_i's. One way to do this is to represent the coefficients of the polynomials in a matrix with respect to some arrangement of the monomials such that the linear and constant terms are placed at the end. By finding the echelon form of this matrix via Gaussian elimination, one can then find all the linear polynomials in the vector space.
– For each of the linear polynomials L, let x be one of its variables and write $L = x + L_0$. Substitute x as L_0 into all the polynomials to eliminate x from the system.
– Repeat this procedure until the system is solved or no more linear polynomials can be found.

Essentially, the ElimLin algorithm successively eliminates variables from the system via linear equations. Thus, a sufficient number of linear polynomials must be generated in the whole process. Otherwise, one may need to use other solving methods to solve the resulting system of equations (with potentially less variables since some variables may have been eliminated). In fact, the ElimLin procedure has been incorporated into other polynomial solving techniques such as the F_4 and F_5 Gröbner basis algorithms [25,26], the characteristic set algorithm [43] as well as some SAT solvers (such as minisat in the preprocessing stage).

2.1.2 SAT solvers

Let us first briefly review the SAT problem and SAT solving techniques.

The satisfiability problem (or SAT for short) is one of the classical NP complete problems that seeks to determine if a boolean formula has a satisfiability assignment. With its wide and varied range of applications, such as in automated testing and artificial intelligence, many efficient SAT solvers have been implemented to solve SAT problems involving large numbers of variables and constraints. Typically, the input propositional statement is represented in its conjunctive normal form (CNF) or a conjunction of clauses. One therefore seeks for a variable assignment that results in all clauses being true or shows that such an assignment does not exist.

Most modern SAT solvers follow the DPLL approach [23,22] that performs a depth first search on the variables with unit propagation. Some innovative features that have been incorporated to help speed up the SAT solvers include good branching heuristics, lazy data structures such as the use of watch literals, non-chronological backtracking, random restarts as well as conflict-driven clause learning. In particular, various conflict driven clause learning techniques have resulted in marked improvements in the progress of SAT solvers. Essentially, these techniques seek to construct useful clauses from conflicts encountered in the solving process. These learned clauses are added to the system to avoid choosing branches that will lead to the same conflict.

Some well-known modern SAT solvers include Minisat [24], CryptoMiniSat [42] and MapleSAT [28]. For a comprehensive survey on modern SAT solvers and the various techniques, the reader may refer to [4].

In order to use SAT solvers to solve the set S of boolean polynomials, one must first convert the polynomials into CNF clauses. One way is to convert each polynomial into its equivalent CNF form by means of truth tables. This conversion strategy, so-called *sparse strategy*, uses the assumption that the input polynomials are rather sparse (i.e. they do not have many terms and variables). Thus we can perform a brute force search of all possible assignments for all logical variables in the polynomial to construct the sparse CNF. Note that this approach will not be feasible when the polynomial has high degree or many terms.

In [2], the authors presented a more general and efficient method to convert a boolean polynomial into CNF. Briefly, for each degree 2 term x_ix_j in f, introduce a new variable y_k to get $y_k = x_ix_j$. Convert $y_k = x_ix_j$ into the following CNF:

$$(y_k \vee \neg x_i \vee \neg x_j) \wedge (\neg y_k \vee x_i) \wedge (\neg y_k \vee x_j).$$

Replace the x_ix_j term in f by y_k to get a linear equation of the form

$$x_{i_1} + x_{i_2} + \ldots + x_{i_s} + y_{j_1} + \ldots + y_{j_t} + c = 0,$$

where $c \in \mathbb{F}$. Fix a cutting number $w < s + t$. Split the linear polynomial into a sum of linear polynomials, each with w terms (one of the polynomials may have fewer than w terms). Introduce new variables z_l for each of these short linear polynomials and continue the process of splitting. Finally, convert the equation of the form $z = x_1 + x_2 + \ldots + x_w$ into the CNF with 2^w clauses, each containing all the variables $z, x_1, \ldots, x_w$ and an odd number of the variables are negative literals. Note that one can combine the above two strategies into a hybrid (s, w) strategy, namely, one uses the sparse strategy when the number of terms of $f \leq s$ and the dense strategy with cutting number w, otherwise.

2.2 Algebraic cryptanalysis and related works

Broadly speaking, algebraic cryptanalysis seeks to represent a cipher algebraically so that the unknown key can be recovered by using one of the algebraic solving techniques. This can be achieved since ciphers are essentially composed of operations/components such as linear operations, logic gates, rotations, permutations, and S-boxes. Among these operations, finding an algebraic representation of S-boxes may be the least straightforward as most S-boxes used in ciphers are of high algebraic degree. However, one may represent the input and output bits of an S-box layer implicitly as polynomials of lower degree of the form $f(x, y) = 0$ [40]. In particular, for an n-bit S-box with $n \leq 6$, one can find a set of quadratic equations to represent the S-boxes.

Consider a block cipher E and a random master key K. For any plaintext P and corresponding ciphertext $C = E_K(P)$, one can now obtain a set of equations in a set of unknown variables representing the key K, as well as the input and output bits of each component in E. More precisely, suppose that E consists of r rounds and suppose that each round has k components. Thus, E can be viewed as a composition of rk functions consecutively. Let x_i denote the input bits of the i-th function in the entire chain of functions and let x_{rk} denote the final output bits. Thus, the block cipher E can be represented by the set S of equations: $S = \{f_i(x_i, x_{i+1}, K_j) = 0 : i = 0, 1, \ldots, rk-1, j = 0, 1, \ldots, r-1\}$, where f_i is the i-th function and K_j is the round key (which may or may not occur in the equation). For a given plaintext/ciphertext (P, C) pair, we substitute $x_0 = P$ and $x_{rk} = C$ and denote the set as $S(P; C)$. In most block cipher designs, one can add to $S(P; C)$ equations representing the key schedule as well. In this way, determining the unknown key boils down to solving the system $S(P; C)$.

Notice that as the number of rounds of E increases, the number of variables in this system will increase correspondingly. Thus, solving this system with any of the polynomial solving methods will be less feasible. For instance, this method was employed in [15,35] on DES and AES, respectively.

Remark 1 In the preceding description, we construct a separate equation for each component in the block cipher. Many of these equations may be simple linear equations (such as permutation layer or rotation layer). As such, some of these components may be combined to form a more compact system. However, combining too many layers together may cause the resulting equations to have a much higher degree or many more terms.

In order to construct more equations involving the unknown key K, one can consider multiple known plaintext/ciphertext pairs and take the union of the sets of polynomial equations. While this technique may not be useful for most polynomial solvers (since the number of variables is much larger), in some cases, this enlarged set may produce some linear equations in the corresponding vector spaces, thus allowing ElimLin to proceed. Such an approach was shown to be successful for the Present block cipher [36] where five rounds of the cipher can be broken with 5 known plaintext/ciphertext pairs and 40 fixed key bits. In [43], the result was improved to solve 5 rounds with 2 known plaintexts/ciphertexts and 28 fixed key bits as well as 6 rounds with 1 known plaintext/ciphertext pair and 52 fixed key bits using an enhanced version of CSA where ElimLin was incorporated into the solver among other added features.

A similar approach to use multiple plaintext/ciphertext pairs was adopted for the Simon cipher as well. In [14], experiments on Simon with a 64-bit block size and 128-bit key size were performed with both ElimLin and SAT solvers using multiple plaintext/ciphertext pairs. Here, the authors first performed Elimlin on the system of algebraic equations representing the ciphers and fed the resulting processed system into a SAT solver. The authors performed experiments with 3 different scenarios. First, random plaintext/ciphertext pairs (RP/RC) were used. In the other two cases, plaintexts with small Hamming distance were selected. In the SP/RC setting, no condition was placed on the ciphertexts while for the other SP/SC setting, the ciphertexts were further restricted to those satisfying some truncated differentials.

From their experimental results, up to 10 rounds of Simon-64/128 can be solved using 8 pairs of plaintext/ciphertext and 90 fixed key bits in the SP/RC setting. For the SP/SC setting, they can similarly break 10 rounds by fixing less number of key bits (i.e., 70 bits) and using 10 pairs of plaintext/ciphertext.

In [37], this method was improved to choose plaintext/ciphertext pairs that yield more linear polynomials. Such plaintexts can be found due to the simple structure of the Simon cipher. As a result, the author demonstrated that up to 11 rounds of Simon-32/64 can be solved when 12 pairs of chosen plaintexts/ciphertexts were used and up to 12 rounds of Simon-64/96 can be solved with 10 chosen plaintext/ciphertext pairs.

In a slightly different direction, the advantage resulting from adding more linear equations to the system of polynomials representing the cipher was exploited in [1]. In this work, algebraic cryptanalysis was combined with Differential cryptanalysis. Concretely, linear equations representing good differential characteristics with a probability p were utilized to solve the polynomial system, and a solution can be expected to be found after about $1/p$ such systems were tested. Using this approach, it was reported in [1] that 6 or 7 rounds of the Present cipher can be solved.

3 An ElimLin-SAT framework

Motivated by the works highlighted in the preceding section, we now present our improved algebraic attack. Essentially, our approach combines the strengths of both ElimLin and modern SAT solvers to launch the algebraic attack. More precisely, let E be a block cipher and let $(P_1, C_1), \ldots, (P_t, C_t)$ represent t plaintext/ciphertext pairs encrypted by E under some key K. The main ideas of our attack work as follows.

– For each $i = 1, 2, \ldots, t$, construct the algebraic system $S_i = S(P_i; C_i)$ for the pair (P_i, C_i).
– Let $S = \bigcup_{i=1}^{t} S_i$.
– Using ElimLin with input S, construct the set L of linear equations.
– Set $T = S \cup L$.
– Convert the equations in T into CNF clauses and solve the corresponding CNF clauses using a good SAT solver.

Recall that in the ElimLin process, the goal is primarily to find linear polynomials so that variables can be eliminated successively from the system. However, two main problems arise. First, while the number of variables keeps on decreasing, the equations in the remaining variables tend to grow much longer after each substitution. This is especially true if we use linear equations with many terms. When converting them into the CNF form, the number of clauses will exponentially rise to the length of these equations. Second, in most cases, the ElimLin procedure may not generate sufficient linear equations to eliminate all the variables to solve the system.

To overcome the latter problem, we will use SAT solvers to solve our system instead. SAT solvers provide a good alternative as they can typically work with many variables. However, notice that apart for the key variables which are common to all the sets S_i, the other variables occurring in different sets are all distinct. This tends to limit the effectiveness of SAT solvers as information on variables in one set will not provide any information on variables used in other sets. As such, we apply ElimLin to the set S to search for linear equations, preferably to relate variables in different sets. Instead of eliminating all the variables occurring in these linear equations from the equations in S, we merely append them to S to form the extended system T. Finally, we convert the equations in T into CNF clauses and feed into a modern SAT solver.

Remark 2 We observe that:

– By adding the linear equations to S, we are, in some sense, adding useful clauses to the system, just like what SAT solvers seek to achieve with conflict driven clause learning techniques.
– For lightweight ciphers, the operations used tend to be less complex, and thus, the equations in S will typically be shorter. From our experiments, we observed that the above approach tends to be more effective when the equations in the set S are kept short.
– A couple of techniques can be employed to improve the above approach. In particular, one can choose to eliminate variables occurring in very short linear equations. For our experiments in this paper, we eliminate variables occurring in very short linear equations of the form $a = b$ or $a = b + 1$ in the ElimLin step before the conversion into CNF clauses.
  In addition, one can find short linear equations from the space spanned by the set L and use these short linear equations instead of L.

It follows from our approach that one hopes to find as many linear equations as possible via ElimLin. To this end, one can select the pairs (P_i, C_i) appropriately so that more linear equations can result. The following lemma will be useful.

Lemma 1 *Let x_i, x'_i, x_j, x'_j be 4 distinct variables and let L and L' be two linear polynomials. Consider the set $A = \{x_ix_j + L, x'_ix'_j + L', x_i + x'_i + c, x_j + x'_j + c'\}$, where $c, c' \in \{0, 1\}$. Then applying ElimLin yields the linear equation*

$$f = L + L' + cx_j + c'x_i + cc'.$$

Proof The proof is straightforward. Simply substitute $x'_i = x_i + c$ and $x'_j = x_j + c'$ into $x'_ix'_j + L'$ and add the result to the equation $x_ix_j + L$ to obtain f. □

proof

Remark 3 We remark that in Lemma 1, we may replace the variables by some linear polynomials L_i, L_j, L'_i, L'_j, respectively. Once again, we obtain a linear equation if we have $L_i + L'_i = c$ and $L_j + L'_j = c'$.

Lemma 1 and Remark 3 suggest that when two pairs of input variables to a quadratic function differ by constants, the output difference is linear. As a block cipher comprises several rounds, it will be helpful to trace the behaviour of the differences across several rounds to find those differences that are linearly related. In particular, we seek to find good differential trails with many linear differences. We describe in Algorithm 1 the steps how to find good differential trails. Specifically, we will consider the non-linear components of the cipher, let each of these components be active with 1 bit difference, and compute the differential trails through Elimlin. We then choose the differential trail with the most number of linear differences.

Algorithm 1 Finding good differential trails

```
INPUT: A block cipher with a low-degree non-linear component
OUTPUT: Find as many linear equations as possible via ElimLin

Step 1. Let $r = 1$.
Step 2. Construct the sets $S(P, C)$ and $S(P', C')$ for $r$ rounds of cipher with $x_i$'s as the variables in $P$ and $x'_i$'s as
    the variables in $P'$. Set $S = S(P, C) \cup S(P', C')$. Here, the inputs and outputs of the cipher are represented by
    variables.
Step 3. Suppose that the nonlinear components of the cipher are of the form $L_iL_j + L$, where $L_i, L_j, L$ are all
    linear.
  For (each such a form) do
    For a variable $x_k$ in $L_i$ or $L_j$ do
      Add to $S$ the linear equations $x_i + x'_i + \delta_i$, where $\delta_i = 0$ for $i \neq k$ and $\delta_k = 1$.
      While new linear equations are obtained Do
        Perform ElimLin to obtain linear relations among the output variables of round $r$.
        Increase $r$ to $r + 1$ and continue to perform ElimLin.
      End While
    End For
  End For
Step 4. Choose a $k$ such that the total number of linear equations obtained in the above procedure is the largest.
```

Remark 4 Starting from this trail, one may work backwards, that is, perform decryption, to extend the number of rounds exhibiting this good differential trail for part of the cipher.

4 Algebraic attack on the Simon cipher

4.1 Description of the Simon cipher

The Simon [3] family of lightweight block ciphers was designed by the National Security Agency and was optimized for hardware implementations. It is based on a typical Feistel design and comprises

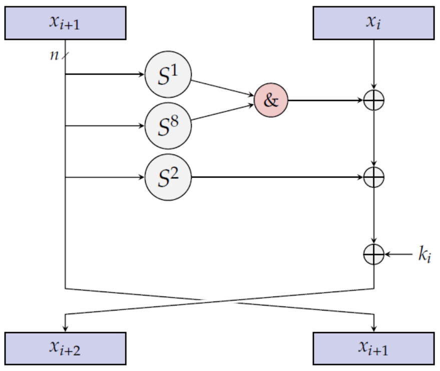

Fig. 1 One round of Simon cipher

3 simple operations, namely, the bitwise 'and', 'rotation' and 'xor' operations. The specifications allowed for a number of block sizes and key sizes with varying number of rounds in each case. Let n denote the word size. Then, Simon-$2n/mn$ will denote Simon with block size and key size $2n$ and mn, respectively, where $m = 2, 3$ or 4. In this paper, we will fix $m = 4$ and consider the cases where $n = 16$ and $n = 32$, that is, Simon-32/64 and Simon-64/128.

The master key K has mn bits and is partitioned into m words to be used as the round keys for the first m rounds of the cipher. Subsequent round keys are generated as linear combinations of the key bits in the preceding m rounds. Let $K^i = (K^i_{n-1}, K^i_{n-2}, \dots, K^i_0)$ denote the round key used in round i, where $0 \le i \le T-1$. For $i \ge m$ and $j = 0, 1, \dots, n-1$, we have

$$K^i_j = K^{i-1}_{(j+3) \bmod n} + K^{i-m+1}_j + K^{i-1}_{(j+4) \bmod n} + K^{i-m+1}_{(j+1) \bmod n} + K^{i-m}_j + c + (z_j)_{i-m} \tag{1}$$

where $c = 2^n - 4 =$ `0xff···fc` is a constant, and $(z_j)_i$ is the $i-th$ bit from the one of the five constant sequences $z_0, z_1, \dots, z_4$. The details of the key schedule can be found in [3].

The round function used for Simon is shown in Figure 1. Observe that the nonlinearity of the function is only provided by the 'and' operation. For a T-round Simon cipher, let $(x^i_{n-1}, \dots, x^i_0 || y^i_{n-1}, \dots, y^i_0))$ denote the input of round i for $i = 0, 1, \dots, T-1$ and $(x^T_{n-1}, \dots, x^T_0 || y^T_{n-1}, \dots, y^T_0)$ be the output after the last round. The following relations hold for any $j = 0, 1, \dots, n-1$:

$$\begin{aligned} x^{i+1}_j &= x^i_{j-1} x^i_{j-8} + x^i_{j-2} + y^i_j + K^i_j, \\ y^{i+1}_j &= x^i_j. \end{aligned} \tag{2}$$

We can now construct the ANF system for a plaintext/ciphertext pair of an T-round Simon cipher as follows. For $i = 0, 1, \dots, T-1$, let $(x^i_j || y^i_j)$ and K^i_j, $j = 0, 1, \dots, n-1$ denote the input and the round key of round i and let $(x^T_j || y^T_j)$ denote the output of the cipher. Using Equation (1), construct the linear equations for each of the round key variables for $i = m, m+1, \dots, T-1$. In addition, Equation (2) gives the equation for each of the x^{i+1}_j in terms of the x^i_j and K^i_j. Hence, S comprises $(2T-4)n$ equations of degree at most 2. As all indices are computed modulo n, in what follows, we omit mod n in indices, that is, x^i_j and K^i_j denotes for $x^i_{j \bmod n}$ and $K^i_{j \bmod n}$, respectively.

4.2 Plaintext pairs with many linear relations

Let us first work with 2 plaintext/ciphertext pairs. Our goal is to find plaintext/ciphertext pairs (P, C) and (P', C') that yield many linear relations via ElimLin. Observe from Equations (2)

Table 1 Trail for the left part of the cipher

Bit position	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Round 0	0	0	0	0	0	0	0	0
Round 1	1	0	0	0	0	0	0	0
Round 2	0	0	0	0	0	0	0	0
Round 3	$*$	0	0	0	0	0	$x_6^1 + x_8^2$	$*$
Round 4	0	0	0	0	$x_6^1 + x_8^2 + x_{10}^3$	$*$	$*$	0
Round 5	$*$	0	$x_6^1 + x_8^2 + x_{10}^3 + x_{12}^4$	$*$	$*$	$*$	$*$	$*$
Round 6	$x_6^1 + x_8^2 + x_{10}^3 + x_{12}^4 + x_{14}^5$	$*$	$*$	$*$	$*$	$*$	$*$	$x_8^1 + x_{10}^2 + x_{12}^3 + x_{14}^4 + x_0^5$

Bit position	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Round 0	0	0	0	0	0	0	0	0
Round 1	0	0	0	0	0	0	0	0
Round 2	x_6^1	0	0	0	0	0	1	x_8^1
Round 3	0	0	0	0	1	$x_8^1 + x_{10}^2$	$*$	0
Round 4	$*$	0	1	$x_8^1 + x_{10}^2 + x_{12}^3$	$*$	$*$	$*$	$*$
Round 5	1	$x_8^1 + x_{10}^2 + x_{12}^3 + x_{14}^4$	$*$	$*$	$*$	$*$	$*$	0
Round 6	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$

that the equations for a Simon round are of the form given in Lemma 1. More precisely, let x_j^i and x'^{i}_{j} denote the variables for the equations arising from the (P, C) pair and the (P', C') pair, respectively. Further, let $\delta_j^i = x_j^i + x'^{i}_{j}$. We have:

$$\delta_j^{i+1} = \delta_{j-1}^i x_{j-8}^i + \delta_{j-8}^i x_{j-1}^i + \delta_{j-1}^i \delta_{j-8}^i + \delta_{j-2}^i + \delta_j^{i-1}. \tag{3}$$

For example, if $\delta_{j-1}^i = \delta_{j-8}^i = 0$ and $\delta_{j-2}^i = \delta_j^{i-1}$, then $\delta_j^{i+1} = 0$. By applying Algorithm 1, one finds that the following differential trail generates many linear relations for Simon-32/64. This trail can be easily extended to Simon-64/128.

Let P and P' be two plaintexts with a non-zero difference at bit $n-1$ and identical everywhere else. In other words, $x_j^0 = x'^{0}_{j}$ for $j = n-1, \ldots, 0$, $y_j^0 = y'^{0}_{j}$ for $j \neq n-1$ and $y_{n-1}^0 = y'^{0}_{n-1} + 1$. In Table 1, we show the trail for the left half of Simon-32/64. The symbol $*$ denotes a difference with nonlinear terms and x_j^i denotes the j-th input bit of round i for the pair (P, C).

From Table 1, we can see that linear relations can be obtained up to the input of round 6 (for bits 15 and 8 in round 6). We next extend this differential trail backwards to generate more rounds.

Extending one round

Let P and P' be as in the preceding trail. Recall that we have $P = (x_{n-1}^0, \ldots, x_0^0 || y_{n-1}^0, \ldots, y_0^0)$ and $P' = (x'^{0}_{n-1}, \ldots, x'^{0}_{0} || y'^{0}_{n-1}, \ldots, y_0^0)$. We seek to find some plaintexts $u^0 = (u_{n-1}^0, \ldots, u_0^0 ||$ $v_{n-1}^0, \ldots, v_0^0)$ and $u'^{0} = (u'^{0}_{n-1}, \ldots, u'^{0}_{0} || v'^{0}_{n-1}, \ldots, v'^{0}_{0})$ such that $u_j^1 + u'^{1}_{j} = x_j^0 + x'^{0}_{j}$ and $v_j^1 + v'^{1}_{j} = y_j^0 + y'^{0}_{j}$. In other words, we have the following equations:

$$\begin{aligned} u_{n-1}^0 &= u'^{0}_{n-1} + 1 \\ u_j^0 &= u'^{0}_{j} \text{ for } j = 0, 1, \ldots, n-2 \\ u_j^1 &= u'^{1}_{j} \text{ for } j = 0, 1, \ldots, n-1. \end{aligned}$$

It follows that $v_j^0 = v'^{0}_{j}$ except when $j-1, j-2, j-8 = n-1$. We consider each of these cases in turn.

- $j = 0$: We have $v_0^0 + v_0^{'0} = u_{n-8}^0$.
- $j = 1$: We have $v_1^0 + v_1^{'0} = 1$.
- $j = 7$: We have $v_7^0 + v_7^{'0} = u_6^0$.

Thus, we have the following difference when $n = 16$:

$$D_1 : u^0 + u'^0 = (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 \,||\, 0,0,0,0,0,0,0,u_6^0,0,0,0,0,0,0,1,u_8^0).$$

Extending two rounds

Using the difference D_1, we once again trace backwards to extend by another round. From D_1, let $u_{n-8}^0 = u_6^0 = 0$. Let the bits in the new extended round be $(P_i^0||Q_i^0)$ and $(P_i^{'0}||Q_i^{'0})$, for $i = 0, 1, \ldots, n-1$. Following the same analysis as above, we obtain the following difference:

$$P_i^0 + P_i^{'0} = \begin{cases} 1 & \text{if } i = 1, \\ 0 & \text{otherwise} \end{cases}$$

$$Q_i^0 + Q_i^{'0} = \begin{cases} 1 & \text{if } i = n-1, 3, \\ P_{n-6}^0 & \text{if } i = 2, \\ P_8^0 & \text{if } i = 9, \\ 0 & \text{otherwise} \end{cases}$$

For $n = 16$, we have:

$$D_2 : (0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0||1,0,0,0,0,0,\ P_8^0,0,0,0,0,0,1,P_{10}^0,0,0).$$

Note that to have $u_8^0 = u_6^0 = 0$, one needs to guess the two key bits K_8 and K_6.

Extending by 3 rounds and beyond

In the same way, we can further extend the trail to more rounds. We illustrate this with an example. Suppose that the master key K is such that $K_{0,j} = 0$ for all even integers j. Let P be a plaintext such that $x_j^0 = 0$ for all even integers j. Then, it follows from Eq (2) that for $i = 1, \ldots, m$, we have $x_j^i = 0$ for all even integers j.

One can now extend the trail with difference D_2 further as shown in Table 2. Here, note that Round -1 refers to the right half of the plaintext (of Round 0). In other words, the whole plaintext difference is given by:

$$(D_5, D_4) : (1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0||1,0,0,0,0,0,\ P_8^0,0,0,0,0,0,1,P_{10}^0,0,0).$$

In our experiments, we begin with a plaintext difference of (D_i, D_{i+1}) and try to solve as many rounds of the cipher as possible.

From the above, we conclude that we can obtain a 8-round differential trail for all keys and a t-round ($t > 8$) differential trail by fixing some key bits. We use the linear expressions from the differential trails to give us more simple relations among the intermediate bits for the different plaintexts used. Together with the cipher equations, we can now apply a SAT solver to solve for the unknown key bits.

Remark 5 It is a common technique to guess some key bits when performing algebraic cryptanalysis [36,14,43]. One may fix a set of random key bits to be guessed. A common method in algebraic cryptanalysis is to choose the key bits which occur most frequently in the algebraic representations so as to simplify the algebraic expressions as much as possible. In our approach, we first fix key bits in order to extend our differential trail as this will lead to more linear relations. We can then fix additional key bits by using other techniques.

Table 2 Extended trail for the left part of Simon-32/64 under the weak key condition where all the even bits are 0

Bit position	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Round -1 (D_5)	1	0	0	0	0	0	0	0
Round 0 (D_4)	0	0	0	0	0	0	0	0
Round 1 (D_3)	1	0	0	0	0	0	0	0
Round 2 (D_2)	0	0	0	0	0	0	0	0
Round 3 (D_1)	1	0	0	0	0	0	0	0
Round 4	0	0	0	0	0	0	0	0
Round 5	1	0	0	0	0	0	0	0
Round 6	0	0	0	0	0	0	0	0
Round 7	$*$	0	0	0	0	0	$x_6^2 + x_8^3$	$*$
Round 8	0	0	0	0	$x_6^2 + x_8^3 + x_{10}^4$	$*$	$*$	0
Round 9	$*$	0	$x_6^2 + x_8^3 + x_{10}^4 + x_{12}^5$	$*$	$*$	$*$	$*$	$*$
Round 10	$x_6^2 + x_8^3 + x_{10}^4 + x_{12}^5 + x_{14}^6$	$*$	$*$	$*$	$*$	$*$	$*$	$x_8^2 + x_{10}^3 + x_{12}^4 + x_{14}^5 + x_0^6$

Bit position	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Round -1 (D_4)	1	0	0	0	1	0	0	0
Round 0 (D_4)	0	0	1	0	0	0	0	0
Round 1 (D_3)	0	0	0	0	1	0	0	0
Round 2 (D_2)	0	0	0	0	0	0	1	0
Round 3 (D_1)	0	0	0	0	0	0	0	0
Round 4	0	0	0	0	0	0	0	0
Round 5	0	0	0	0	0	0	0	0
Round 6	x_6^2	0	0	0	0	0	1	x_8^2
Round 7	0	0	0	0	1	$x_8^2 + x_{10}^3$	$*$	0
Round 8	$*$	0	1	$x_8^2 + x_{10}^3 + x_{12}^4$	$*$	$*$	$*$	$*$
Round 9	1	$x_8^2 + x_{10}^3 + x_{12}^4 + x_{14}^5$	$*$	$*$	$*$	$*$	$*$	0
Round 10	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$

Extending to more plaintext/ciphertext pairs

Often, two plaintext/ciphertext pairs may not generate sufficiently many linear equations. Observe that we can easily obtain a rotated trail by rotating the left half and the right half simultaneously by i bits for $i = 0, 1, \ldots, n-1$. In this way, we can get n differences. For a difference D, let $D^{(i)}$ denote the difference where the left and right halves are simultaneously rotated by i bits to the right. To obtain t different plaintext pairs that are likely to generate many linear equations, we pick a set U of s integers between 0 and $n-1$, where $t \leq 2^s$ and take the first t differences from the vector space spanned by $\{D^{(i)} : i \in U\}$. In the case where no key bit needs to be fixed, the set U can be arbitrarily fixed. However, in the case where some key bits need to be guessed or fixed, we need to choose the set U more carefully. More specifically, we will choose U so that the best differential trails involve the same fixed key positions/values. In the following, we list down the set U, the structure of the plaintext and the key bits we fix for our experiments.

In the following, let $(x_{n-1}^0, \ldots, x_0^0 || y_{n-1}^0, \ldots, y_0^0)$ represent the bits of one of the plaintexts P. Let the master key bits be $K_0 K_1 \ldots K_{mn-1}$.

- 12 rounds and fewer (Simon-32/64 and Simon-64/128): Choose P so that $x_{2i}^0 = 0$ for $i = 0, 1, \ldots, (n-2)/2$. Let $D = (D_1, D_2)$. Set $U = \{0, 1, 2, 3, \ldots, s-1\}$.
- 13 rounds (Simon-32/64): Guess the key bits K_i for $i \in \{10, 8, 6, 4, 2, 0\}$. Pick a plaintext P such that $x_{2i}^0 = 0$ for $i = 0, 1, \ldots, 7$ and $y_i^0 = K_i$ for $i \in \{8, 7, 6, 5, 4, 3\}$. Let the plaintext difference be $D = (D_2, D_3)$. Set $U = \{0, 1, 2, 3\}$ to obtain less than $16 (P, C)$ pairs.
- 13 rounds (Simon-64/128): Guess the key bits K_i for $i \in \{24, 22, 20, 6, 4, 2\}$. Pick a plaintext P such that $x_{2i}^0 = 0$ for $i = 0, 1, \ldots, 15$ and $y_i^0 = K_i$ for $i \in \{24, 22, 20, 6, 4, 2\}$. Let the plaintext difference be $D = (D_2, D_3)$. Set $U = \{0, 2, 4\}$ to obtain less than 8 (P, C) pairs.
- 14 rounds (Simon-32/64): Let I be all the even integers from 1 to 15. Suppose that the master key has bits $K_{16+i} = 0$ for $i \in I$. Guess the key bits K_i for $i \in I$. Pick a plaintext P such

that $x_i^0 = 0$ for $i \in I$ and $y_i^0 = K_i$ for $i \in I$. Let the plaintext difference be $D = (D_3, D_4)$. Set $U = \{0, 2, 4, 6\}$ to obtain less than $16(P, C)$ pairs.

– 14 rounds (Simon-64/128): Let $I_1 = \{24, 22, 20, 6, 4, 2\}$ and let I_2 be all the even integers from 1 to 31. Suppose that the master key has bits $K_{32+i} = 0$ for $i \in I_1$. Guess the key bits K_i for $i \in I_2$. Pick a plaintext P such that $x_i^0 = 0$ for $i \in I_2$ and $y_i^0 = K_i$ for $i \in I_2$. Let the plaintext difference be $D = (D_3, D_4)$. Set $U = \{0, 2, 4\}$ to obtain less than $8(P, C)$ pairs.
– 15 rounds and more (Simon-32/64): Let I be all the even integers from 1 to 15. Suppose that the master key has bits $K_{16+i} = 0$ and $K_{32+i} = 0$ for $i \in I$. Guess the key bits K_i for $i \in I$. Pick a plaintext P such that $x_i^0 = 0$ for $i \in I$ and $y_i^0 = K_i$ for $i \in I$. Let the plaintext difference be $D = (D_4, D_5)$. Set $U = \{0, 2, 4, 6\}$ to obtain less than $16(P, C)$ pairs. In the same way, this can be easily extended to 16 rounds and more.
– 15 rounds and more (Simon-64/128): Let $I_1 = \{24, 22, 20, 6, 4, 2\}$ and let I_2 be all the even integers from 1 to 31. Suppose that the master key has bits $K_{64+i} = 0$ for $i \in I_1$ and $K_{32+i} = 0$ for $i \in I_2$. Guess the key bits K_i for $i \in I_2$. Pick a plaintext P such that $x_i^0 = 0$ for $i \in I_2$ and $y_i^0 = K_i$ for $i \in I_2$. Let the plaintext difference be $D = (D_4, D_5)$. Set $U = \{0, 2, 4\}$ to obtain less than $8(P, C)$ pairs. In the same way, this can be easily extended to 16 rounds and more.

Remark 6 For the experiments in this paper, we have only considered a class of weak keys when the number of rounds is 14 or more, namely, we have considered master keys with some of their key bits equal to 0. This is to ensure that the intermediate bits x_j^i for even j will be 0 for the first few rounds in order to facilitate the extension of the differential trails beyond 13 rounds. In the general case for a T-round Simon cipher with $T > 12$, one can do the following. Construct the equations for the first $T - 12$ rounds with the output differences given by the plaintext differences for the 12-round experiments described above. Fix a subset of key bits and guess their values. Find a set of plaintexts that satisfy the corresponding equations and use these plaintexts to solve the T-round cipher.

4.3 Our experimental results

Based on the above analysis, we performed experiments on the Simon cipher. This section reports the average timings obtained using an SAT solver for different rounds on 2 variants: Simon-32/64 and Simon-64/128.

We performed experiments for up to 16 rounds using up to 16 chosen plaintext/ciphertext pairs. Our experiments were performed according to the following procedure. We first generated at random some problem instances (at least 5 and up to 50). Each instance corresponds to a random key and plaintext/ciphertext pairs that produce the best differential trails according to Tables 1 and 2. We then used Magma [8] to generate the ANF equations. For each of the instances, we converted the ANF equations into CNF clauses by using both *dense/sparse* strategies [38]. We also used different settings of parameters (i.e., *max_vars_sparse* and *cutting number*) to convert. From our observations, the ANF equations in which their length and number of variables are from 3 to 6 dominate the systems. We hence ran different strategies, that is *max_vars_sparse* varies from 3 to 5 and *cutting number* varies from 4 to 6, for each instance of problems.[1] The timings were obtained by running CryptoMiniSat 5.0.1 [41] on a 3.6GHz Intel Core i7 CPU with 8GB RAM. Note that we ignore the timings for conversion of ANF equations into CNF clauses and consider it as a preprocessing step. In the case where key bits need to be guessed, we ran the experiment when a correct guess of the bits is made. We thus have to multiply by the total number of key guesses to get an approximate time to recover the correct key.

The following tables present the average timings obtained for rounds from 8 to 15 for Simon-32/64 and Simon-64/128, respectively. As mentioned, we ran different conversion strategies for each instance and our average is taken over the timings obtained from the best strategies over all

[1] See [38] for more details about these parameters.

Table 3 Average timings in seconds on Simon-32/64

Round Attacked	# Pairs	# Key bits guessed	# key bits fixed	Timing (s)	Total time (s)
8	4	0	0	< 1	< 1
9	4	0	0	< 1	< 1
10	3	0	0	32.9	32.9
10	4	0	0	< 1	< 1
11	6	0	0	1.7	1.7
12	12	0	0	48.9	48.9
12	16	0	0	15.8	15.8
13	14	6	0	86.8	5542
13	16	5	0	61.4	1965
13	16	6	0	17	1088
14	16	8	8	51.4	13158
15	16	8	16	328.2	84019

Table 4 Average timings in seconds on Simon-64/128

Round Attacked	# Pairs	# Key bits guessed	# Key bits fixed	Timing (s)	Total time (s)
8	5	0	0	< 1	< 1
9	5	0	0	< 1	< 1
10	5	0	0	< 1	< 1
11	5	0	0	1.2	1.2
11	6	0	0	< 1	< 1
12	6	0	0	12.6	12.6
13	6	6	0	13	832
14	8	6	16	< 1	64
15	8	6	32	< 1	64
16	8	6	48	< 1	64

Table 5 A comparison of Algebraic Attacks on the Simon cipher

Paper	Cipher	Round Attacked	# Pairs	# Key bits fixed	Timing (s)
Raddum [37]	Simon-32/64	11	12	0	Not reported
This paper	Simon-32/64	12	12	0	48.9
This paper	Simon-32/64	12	16	0	15.8
This paper	Simon-32/64	15	16	24	328.2
Courtois et al. [14]	Simon-64/128	10	10	70	417.73
This paper	Simon-64/128	10	5	0	< 1
This paper	Simon-64/128	16	8	54	< 1

the random instances.

Tables 3–4 show that we can break up to 10 rounds of Simon-32/64 within 1 second by using only 4 plaintext/ciphertext pairs and without fixing any key bit. For a class of weak keys (with 16 of the master key bits fixed to 0), we can break 15 rounds by guessing 8 key bits using 16 plaintext/ciphertext pairs for each guess. We obtained even better timings on Simon-64/128 (up to 12 rounds without fixing any keybits). By guessing 6 key bits, we can break up to 16 rounds for a class of weak keys using only 8 chosen plaintext/ciphertexts for each guess.

Table 5 makes a comparison between our experimental results and other results in the literature. We compare our result on Simon-32/64 and Simon-64/128 with the ones in the papers [37] and [14], respectively. To be consistent with the results of other papers, we will compare the timings for a key guess.

Remark 7 There have been several cryptanalytic works on Simon including [6,31,11]. To the best of our knowledge, the best attacks on Simon are the integral attacks found in [11]. The authors demonstrated an improved integral attack that can break Simon-32/64 with a data complexity of

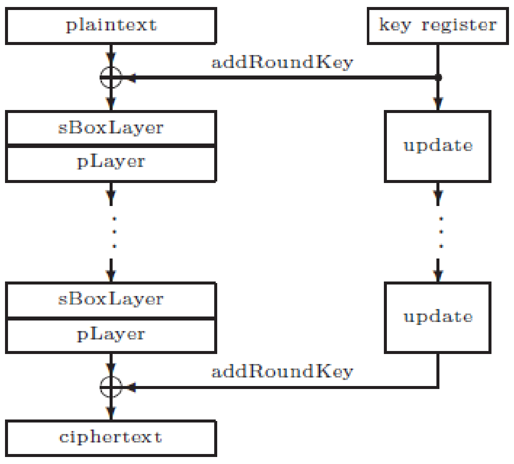

Fig. 2 One round of Present cipher

2^{32} and 25 rounds of Simon-64/128 with a data complexity of 2^{48}. By contrast, we have implemented our attacks on a 3.6GHz Intel Core i7 CPU and all the timings we obtained are less than 10 minutes for a key guess. We emphasize that the focus of this work is to propose better algebraic attacks as evidenced by Table 5.

5 Algebraic attack on the Present cipher

5.1 Description of the Present cipher

Next, we turn to another lightweight block cipher, the Present cipher [7]. Present is a block cipher with block size of 64 bits and key size of 80 or 128 bits. In this paper, we only focus on the 80-bit key Present cipher. Its design is based on a substitution-permutation network with 16 parallel executions of a 4-bit S-box $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$. The action of this box in hexadecimal notation is given by the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

One round of the cipher is shown in Figure 2. For key schedule, the user-supplied key is stored in a key register K and represented as $k_{79}k_{78}\dots k_0$ (for 80-bit version). At round i the 64-bit round key $K_i = \kappa_{63}\kappa_{62}\dots\kappa_0$ consists of the 64 leftmost bits of the current contents of register K. Once the round key K_i is extracted, they key register K will be updated (see [7, Section 3] for more details).

Algebraic attacks on Present were carried out in [36, 43] where ElimLin and the enhanced version of CSA were employed to solve the algebraic system. In these attacks, 21 quadratic equations were used to represent the S-box. We will show that in fact, 4 quadratic equations are sufficient to represent the S-box. Indeed, consider the following 4 equations:

$$\begin{aligned}
x_4y_1 + x_4y_3 + x_2 + x_3 + y_1 + y_4 + 1 &= 0,\\
y_1y_3 + x_4 + y_2 + y_4 + 1 &= 0,\\
x_2x_3 + x_1 + x_2 + x_4 + y_4 &= 0,\\
x_1x_2 + x_1x_3 + x_1 + x_2 + y_1 + y_3 + y_4 + 1 &= 0,
\end{aligned}$$

where x_1, x_2, x_3, x_4 represent the input bits and y_1, y_2, y_3, y_4 represent the output bits.

First, one can check that all the 16 substitutions satisfy the above 4 equations. Next, fix the input bits x_1, x_2, x_3, x_4. We can rewrite the equations as:

$$\begin{aligned} y_4 &= x_2x_3 + x_1 + x_2 + x_4, \\ y_1 + y_3 + y_4 &= x_1x_2 + x_1x_3 + x_1 + x_2 + 1, \\ y_1 + y_4 + x_4(y_1 + y_3) &= x_2 + x_3 + 1, \\ y_1y_3 + y_2 + y_4 &= x_4 + 1 \end{aligned}$$

Observe that from the first equation, one can obtain y_4. Substituting the value of y_4 into the second equation gives $y_1 + y_3$. One then gets y_1 from the third equation and thus, the value of y_3 from the second equation. Finally, one obtains the value of y_2 from the last equation. This shows that whenever (x_1, x_2, x_3, x_4) is fixed, (y_1, y_2, y_3, y_4) is uniquely defined.

We use these four equations to represent the S-box in Present. We may also combine the permutation layer and the round key layer in the next round to form the linear layer. For each round, we thus introduce 128 variables as the inputs of these two layers. Each Present round thus comprises 64 quadratic equations and 64 linear equations in 128 variables.

As for the key schedule, we let $k_0, k_1, \ldots, k_{79}$ denote the master key and introduce 4 new variables for each round. Since these 4 new variables go through an S-box, each round of the key schedule will involve 4 quadratic equations. Combining these key equations to the Present round equations will thus give all the equations necessary to represent the Present cipher.

5.2 Differential trails for Present

Observe that the S-box equations for Present follow the form discussed in Lemma 1 (see Remark 3). Thus, one can hope to find good differential trails that generate many linear equations. By applying Algorithm 1, we selected one for which only the rightmost bit of an S-box, say the rightmost S-box, is active. Here, note that the differential trail tables for differences an active rightmost bit in an S-box are permutations of each other. With this difference, it can be shown that linear equations can be generated up to 4 rounds of the cipher.

Similar to the analysis on Simon, one can extend this differential trail backwards to more rounds involving some fixed key bits. In addition, one can consider multiple plaintext/ciphertext pairs.

We now list down the plaintext, the differences and the fixed key bits for our experiments. In particular, we demonstrate how we may choose up to 16 pairs of plaintexts/ciphertexts. Let the round keys be denoted by $K^i = (K^i_0, K^i_1, \ldots, K^i_{63})$.

- For 7 rounds and fewer: Choose P to be a random plaintext. Let D_0, D_1, D_2, D_3 be 64-bits strings with 1 in bit $4i$, $i = 15, 14, 13, 12$ and 0 everywhere else. Consider the vector space V spanned by these 4 differences and choose the plaintexts with difference (with respect to P) coming from V.
- For 8 rounds: Set $I = \{51, 50, 49, 48, 35, 34, 33, 32, 19, 18, 17, 16, 60, 61, 62, 63\}$. Guess the key bits $K_{0,i}$ for $i \in I$. Choose a plaintext P such that $P^0_i = K_{0,i}$ for $i \in I$. Let D_0, D_1, D_2, D_3 be the following (written in hexadecimal):

$$\begin{aligned} D_0 &= 0x000C000000000000, \\ D_1 &= 0x0000000C00000000, \\ D_2 &= 0x00000000000C0000, \\ D_3 &= 0x000000000000000C. \end{aligned}$$

Table 6 Average timings in seconds on Present

Round Attacked	# Pairs	# Key bits fixed	Timing (s)
5	2	0	3.4
6	4	0	1.4
7	6	0	394.5
7	7	0	270.5
8	6	16	135.2
9	8	32	2.4

Table 7 A comparison of Algebraic Attacks on the Present cipher

Paper	Method	Round Attacked	# Pairs	# Key bits fixed	Timing
Nakahara et al. [36]	ElimLin	5	5	40	< 3 minutes
Sepehrdad [39]	ElimLin	5	16	35	1.845 hours
Sepehrdad [39]	PolyBori	5	10	37	0.523 hours
Yeo et al. [43]	CSA + ElimLin	5	2	28	2120.8(s)
Yeo et al. [43]	CSA + ElimLin	6	1	52	1489.0(s)
This paper	SAT solvers	5	2	0	3.4(s)
This paper	SAT solvers	6	4	0	3.4(s)
This paper	SAT solvers	9	8	32	2.4(s)

As before, we generate the space of 16 differences from this set.

– For 9 rounds: Guess the key bits $K_{0,i}$ for $i = 15, 14, \dots, 0$ and $K_{1,j}$ for $j \in I$. We start from the plaintexts for 8 rounds (see above) and then trace back to obtain the required plaintexts for 9 rounds.

5.3 Our experimental results

Table 6 presents the average timings obtained for rounds from 5 to 9 for the Present cipher. The experiments in this section were set up as the same as those in Section 4.3. By fixing 32 key bits and using 8 pairs of plaintext/ciphertext, we can solve the key for 9 rounds of Present within few seconds for each key guess.

We make a comparison in Table 7 between our method and other techniques in the literature. As shown, the existing algebraic attacks were reported up to 6 rounds only. Moreover, in all existing algebraic attacks on both 5 rounds and 6 rounds Present cipher, a portion of the key bits needs to be fixed. This contrasts with our attack which can solve 6-round Present cipher using 4 chosen plaintexts without fixing any key bit. To the best of our knowledge, our paper is the first report breaking the Present cipher beyond 6 rounds using algebraic attacks.

6 Further Enhancements

Our attack framework described in the previous sections makes use of Elimlin as an essential tool to generate more linear relations among the sets of variables from different plaintext/ciphertext pairs. As explained earlier, these linear relations are helpful to the SAT solver as they may provide more relations that cannot be easily deduced by the SAT solvers from the original equations. On the other hand, these linear relations are efficiently obtained from the ElimLin process through algebraic means. This section will investigate other techniques to further enhance the ElimLin-SAT framework. First, we will take a closer look at the linear equations generated by the ElimLin process. Next, we describe how we may exploit differential representations among different plaintexts to create more relations for the SAT solver. We will demonstrate the effectiveness of these techniques by performing experiments to attack Simon-32/64.

6.1 More about ElimLin and higher-order differentials

In this section, we will shed more light on the ElimLin process on the cipher equations. Concretely, we will examine the possible linear equations that are obtained after performing ElimLin on the sets of equations. We will use the Simon-32/64 cipher to illustrate our ideas.

First, let P and P' be two plaintexts with their difference taking the form $D^0 = (0, 0, \dots, 0||1, 0, \dots, 0)$. Let C and C' be the ciphertexts of P and P' under a certain key K after r rounds. Let $S(P, C)$ and $S(P', C')$ be the two sets of equations representing the (P, C) and (P', C') encryptions. According to Table 1, one can obtain linear relations for some of the output difference bits in terms of the intermediate output bits of one of the plaintext/ciphertext pairs. Such linear relations can be obtained for output difference bits up to round 6. In fact, these linear relations are precisely the linear relations obtained by performing ElimLin on $S(P, C) \cup S(P', C')$ up to round 6.

Similarly, by looking at the difference $C + C'$, one can also obtain some linear equations for the last few rounds. In fact, the following lemma holds:

Lemma 2 *Let $\Delta^i = (\delta^i_{n-1}, \dots, \delta^i_0)$ denote the output difference after round i for $i = 0, 1, \dots, T$. Let K^i denote the round key used in round i. For $j = 0, 1, \dots, n-1$, we have*

– *δ^i_j is a known value derived from the ciphertext bits for $i = T, T-1, T-2$.*
– *$\delta^{T-3}_j = f_j(K^{T-1})$, where f is a function of the bits in K^{T-1} with degree $\deg(f_j) \leq 1$.*
– *$\delta^{T-4}_j = g_j(K^{T-1}, K^{T-2})$, where g_j is a function of the bits in K^{T-1} and K^{T-2} with $\deg(g_j) \leq 2$.*

Proof Let $x^i = (x^i_{n-1}, \dots, x^i_0)$ denote the left half of the output bits after round i for the plaintext P.

– Since both C and C' are known, it follows easily that Δ^T and Δ^{T-1} are known. Now, from Equation (3), we see that Δ^{T-2} is a function of the bits in Δ^T, Δ^{T-1} and x^{T-1}. Clearly, x^{T-1} is the right half of C which is known. Thus, it follows that Δ^{T-2} is known.
– For all $j = 0, 1, \dots, n-1$, it follows from Equation (2) that $x^{T-2}_j = x^{T-1}_{j-1}x^{T-1}_{j-8} + x^{T-1}_{j-2} + x^T_j + K^{T-1}_j = K^{T-1}_j + c_j$ where c_j is a constant. Substituting into Equation (3) then yields:

$$\begin{aligned}\delta^{T-3}_j &= \delta^{T-2}_{j-1}(K^{T-1}_{j-8} + c_{j-8}) + \delta^{T-2}_{j-8}(K^{T-1}_{j-1} + c_{j-1}) \\ &\quad + \delta^{T-2}_{j-1}\delta^{T-2}_{j-8} + \delta^{T-2}_{j-2} + \delta^{T-1}_j \\ &= \delta^{T-2}_{j-1}K^{T-1}_{j-8} + \delta^{T-2}_{j-8}K^{T-1}_{j-1} + b_j,\end{aligned}$$

where b_j is a known constant.
– Similarly, one can show that

$$x^{T-3}_j = K^{T-1}_{j-1}K^{T-1}_{j-8} + L_j(K^{T-1}_{j-1}, K^{T-1}_{j-2}, K^{T-1}_{j-8}, K^{T-2}_j),$$

where $\deg(L_j) \leq 1$.
Hence,

$$\begin{aligned}\delta^{T-4}_j &= (K^{T-1}_{j-2}K^{T-1}_{j-9} + L_{j-1})(\delta^{T-2}_j K^{T-1}_{j-9} + \delta^{T-2}_{j-9}K^{T-1}_j + b_{j-8}) \\ &\quad + (K^{T-1}_{j-9}K^{T-1}_j + L_{j-8})(\delta^{T-2}_{j-2}K^{T-1}_{j-9} + \delta^{T-2}_{j-9}K^{T-1}_{j-2} + b_{j-1}) + g_j \\ &= g'_j,\end{aligned}$$

where both g_j and g'_j have degrees at most 2.

□

It follows from Lemma 2 that the output difference bits in round $T-3$ are linear relations of the last round key bits while only a few linear relations can be expected in round $T-4$ (since C and C' are random). In the case where $T-4 \leq 6$, the two sets of linear equations may further interact to create more linear equations through the ElimLin process. Otherwise, the ElimLin process does not yield additional linear equations. Note that the above argument holds for any arbitrary input difference but we have chosen the input difference that gives the most linear equations.

Next, we consider a set of more than two plaintext/ciphertext pairs. Let S be a set of t plaintexts. By considering the encryptions of any two different plaintexts in S, the above argument suggests that ElimLin will produce linear relations among the output difference bits and the plaintext input bits. We now show that in some instances, one may obtain more linear relations through the so-called higher-order differences.

We will use the following notations. For a plaintext P, let $x_j^i(P)$ denote the j-th bit after round i for the plaintext P. For any set S of plaintexts, let $\delta_j^i(S)$ denote the sum $\sum_{P\in S} x_j^i(P)$.

Lemma 3 *For $i,j = 0,1,\ldots,n-1$, let j_i denote $j-i \pmod n$. Let $J = \{(j_1,j_8),(j_8,j_1)\}$. Let $S = \{P_1,P_2,P_3,P_4\}$ be a set containing 4 distinct plaintexts. We have:*

$$\delta_j^{i+1}(S) = \sum_{(a,b)\in J} (\delta_a^i(P_1,P_2)\delta_b^i(P_1,P_3) + x_a^i(P_4)\delta_b^i(S)) + \delta_{j_1}^i(S)\delta_{j_8}^i(S) + \delta_{j_2}^i(S) + \delta_j^{i-1}(S).$$

Proof The proof works by splitting $\delta_j^{i+1}(S) = \delta_j^{i+1}(P_1,P_2) + \delta_j^{i+1}(P_3,P_4)$ and applying Equation (3). Concretely, we have:

$$\begin{aligned}
\delta_j^{i+1}(S) &= \delta_j^{i+1}(P_1,P_2) + \delta_j^{i+1}(P_3,P_4)\\
&= \sum_{(a,b)\in J} (x_a^i(P_1)\delta_b^i(P_1,P_2) + x_a^i(P_3)\delta_b^i(P_3,P_4)) +\\
&\quad \delta_{j_1}^i(P_1,P_2)\delta_{j_8}^i(P_1,P_2) + \delta_{j_1}^i(P_3,P_4)\delta_{j_8}^i(P_3,P_4) +\\
&\quad \delta_{j_2}^i(P_1,P_2) + \delta_{j_2}^i(P_3,P_4) + \delta_j^{i-1}(P_1,P_2) + \delta_j^{i-1}(P_3,P_4)\\
&= \sum_{(a,b)\in J} (x_a^i(P_1)\delta_b^i(P_1,P_2) + x_a^i(P_3)(\delta_b^i(S) + \delta_b^i(P_1,P_2))) +\\
&\quad \delta_{j_1}^i(P_1,P_2)\delta_{j_8}^i(P_1,P_2) + \delta_{j_1}^i(P_3,P_4)\delta_{j_8}^i(P_3,P_4) +\\
&\quad \delta_{j_2}^i(P_1,P_2) + \delta_{j_2}^i(P_3,P_4) + \delta_j^{i-1}(P_1,P_2) + \delta_j^{i-1}(P_3,P_4)\\
&= \sum_{(a,b)\in J} (\delta_a^i(P_1,P_3)\delta_b^i(P_1,P_2) + x_a^i(P_3)\delta_b^i(S)) +\\
&\quad (\delta_{j_1}^i(S) + \delta_{j_1}^i(P_3,P_4))(\delta_{j_8}^i(S) + \delta_{j_8}^i(P_3,P_4)) + \delta_{j_1}^i(P_3,P_4)\delta_{j_8}^i(P_3,P_4) +\\
&\quad \delta_{j_2}^i(P_1,P_2) + \delta_{j_2}^i(P_3,P_4) + \delta_j^{i-1}(P_1,P_2) + \delta_j^{i-1}(P_3,P_4)\\
&= \sum_{(a,b)\in J} (\delta_a^i(P_1,P_3)\delta_b^i(P_1,P_2) + x_a^i(P_3)\delta_b^i(S) + \delta_a^i(P_3,P_4)\delta_b^i(S)) +\\
&\quad \delta_{j_1}^i(S)\delta_{j_8}^i(S) + \delta_{j_2}^i(P_1,P_2) + \delta_{j_2}^i(P_3,P_4) + \delta_j^{i-1}(P_1,P_2) + \delta_j^{i-1}(P_3,P_4)\\
&= \sum_{(a,b)\in J} (\delta_a^i(P_1,P_3)\delta_b^i(P_1,P_2) + x_a^i(P_4)\delta_b^i(S)) + \delta_{j_1}^i(S)\delta_{j_8}^i(S) + \delta_{j_2}^i(S) + \delta_j^{i-1}(S).
\end{aligned}$$

□

Remark 8 Note that the expression for $\delta_j^i(S)$ given in Lemma 3 is not unique as one can write $\delta_j^{i+1}(S)$ as sums of different pairs. For instance, another way to write $\delta_j^{i+1}(S)$ is $\delta_j^{i+1}(S) = \delta_j^{i+1}(P_1,P_3) + \delta_j^{i+1}(P_2,P_4)$.

Table 8 The left half differential trail for $S_1(P)$

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Round 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 3	0	0	0	0	0	0	0	x_6^2	0	0	0	0	0	0	0	0
Round 4	0	0	0	0	0	x_8^3	$*$	0	0	0	0	0	0	0	$*$	$*$
Round 5	0	0	0	$x_8^2 + x_{12}^4$	$*$	$*$	$*$	$*$	0	0	0	0	$*$	$*$	$*$	0
Round 6	$\delta_{5,14}(P_1, P_3)$	$x_6^2 + x_{10}^4$	$*$	$*$	$*$	$*$	$*$	$\delta_{5,0}(P_1, P_3)$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$

Table 9 The left half differential trail for $S_2(P)$

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Round 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 4	0	0	0	0	0	0	x_6^2	0	0	0	0	0	0	0	0	$*$
Round 5	0	0	0	0	$x_6^2 + x_8^3$	$*$	$*$	$*$	0	0	0	0	0	$*$	$*$	0
Round 6	0	0	$x_8^3 + x_{10}^4$	$*$	$*$	$*$	$*$	$\delta_{5,14}(P_1, P_3, P_5, P_7)$	$*$	0	0	$*$	$*$	$*$	$*$	$*$

For $i = 0, 1, \dots, n-1$, let $d_i = (0, \dots, 0 || 0, \dots, 1, 0, \dots, 0)$, where bit i of the right half of d_i is 1 and all other bits are 0. Let V_i be the linear span of $\{d_{n-1}, d_{n-2}, \dots, d_{n-1-i}\}$ so $|V_i| = 2^{i+1}$. For any plaintext P, let $S_i(P)$ denote the set $S_i(P) = \{P + v : v \in V_i\}$. In this case, we have for $i \neq 0$,

$$\sum_{P' \in S_i(P)} P' = 0.$$

Observe that Table 1 gives the differential trail for $S_0(P)$ for any plaintext P. By applying Lemma 3, one can obtain the differential trail for $S_1(P)$ as follows. Let $S_t(P) = \{P_1, P_2, \dots, P_{2^{t+1}}\}$.

By comparing Tables 1 and 8, we see that one finds new linear relations for $\delta_j^i(S_1)$ for $(i, j) = (3, 15), (3, 8), (3, 1), (4, 10), (4, 3), (5, 12), (5, 5), (6, 14)$.

Similarly, one can generalize this method to compute the differential trails for S_i, $i \geq 2$ up to round 6. As the expression for $\delta_j^i(S_i(P))$ becomes more complicated for bigger i, we propose the following algorithm to find the differential trails.

Algorithm 2 Algorithm to find higher-order differential trails for $S_t(P)$

```
1. Write $S_t(P) = \{P_1, \dots, P_{2^{t+1}}\}$.
2. For all $i, j$ where the difference $P_i + P_j$ has Hamming weight 1, do the following:
3. Compute the differential trail of $P_i + P_j$ in terms of the output bits of one of $P_i$.
4. Collect all the $x_{u,v}(P_i) + x_{u,v}(P_j) = \delta_{u,v}(P_i, P_j)$, where $\deg(\delta_{u,v}) \leq 1$.
5. Let $L$ be the set of all the linear equations for all pairs $(i, j)$ considered above.
6. Perform Gaussian elimination on $L$.
7. Substitute the leading variables in the resulting linear equations into the differential trail equations for the pairs
   $(P_1, P_2), (P_3, P_4), \dots, (P_{2^{t+1}-1}, P_{2^{t+1}})$.
8. Compute $\delta_j^i(S_t(P)) = \delta_j^i(P_1, P_2) + \delta_j^i(P_3, P_4) + \dots + \delta_{P_{2^{t+1}-1}}, P_{2^{t+1}})$.
```

By applying this algorithm, we compute the differential trails for S_i for $i = 2, 3, 4$ below.

Remark 9 The additional linear equations obtained by considering more plaintext/ciphertext pairs help to explain why more plaintext pairs are helpful in our attack. However, performing ElimLin to find more linear equations with a large number of plaintexts is also more difficult as the matrices

Table 10 The left half differential trail for $S_3(P)$

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Round 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 5	0	0	0	0	0	x_6^2	$*$	$*$	0	0	0	0	0	0	$*$	0
Round 6	0	0	0	x_8^3	$*$	$*$	$*$	$\delta_{5,14}(P_1, P_3, \ldots, P_{15})$	$*$	0	0	$*$	$*$	$*$	$*$	$*$

Table 11 The left half differential trail for $S_4(P)$

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Round 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Round 5	0	0	0	0	0	0	0	$*$	0	0	0	0	0	0	0	0
Round 6	0	0	0	0	x_6^2	$*$	$*$	$\delta_{5,14}(P_1, P_3, \ldots, P_{31})$	$*$	0	0	0	$*$	$*$	$*$	$*$

involved in the Gaussian elimination becomes very large. As such, we have only applied Algorithm 2 to compute the differential trails up to $S_4(P)$ and up to round 6 for each differential trail. Nonetheless, by performing experiments on random plaintexts and keys, we have the following observations:

– $\delta_{5,j}(S_5(P)) = 0$ for all $j = 0, 1, \ldots, 15$.
– $\delta_{6,j}(S_6(P)) = 0$ for all $j = 0, 1, \ldots, n-1$.
– $\delta_{7,j}(S_8(P))$ is not a constant for all $j = 0, 1, \ldots, 15$.

6.2 Improved algebraic attack

In this section, we demonstrate further improvements to our ElimLin-SAT attack and apply the improved attack to Simon-32/64.

Thus far, we have only used the linear equations occurring in the differential trail (Table 1) to aid the SAT solving process. We have seen from our experiments that such linear equations have led to better performance of the SAT solvers in terms of solving more rounds of the ciphers. An immediate improvement is to include other higher degree equations in the differential trail to provide more relations among the variables. Essentially, we wish to consider equations with the following characteristics:

– Equations that are not directly deduced from existing equations. For instance, for an equation L, the equations xL or $L + x$ for a variable x may not be useful.
– Equations that will not result in very long clauses.

For example, referring to Table 1, we see that $\delta_{3,15} = x_{14}^2\delta_{2,7} + x_7^2\delta_{2,14} + \delta_{2,14}\delta_{2,7} + \delta_{2,13} + \delta_{1,15} = x_{14}^2 x_6^1 + 1$. Equivalently, we have

$$x_{15}^3(P) + x_{15}^3(P') = x_9^1(P)x_1^2(P) + 1.$$

This is a short quadratic equation, which is not included for the ElimLin-SAT solving system. This example motivates us to add on to the ElimLin-SAT process as follows.

– For any two plaintexts P and P' with Hamming distance 1, define new variables $\delta_j^i(P, P') = x_j^i(P) + x_j^i(P')$. Let L be the set of all these linear equations.

Table 12 Experimental results on Simon-32/64 for the the improved attack by adding the differential equations

Round	With differential equations		Without differential equations	
	No. of (P, C) & Key bits fixed	Timing	#(P, C) & # Key bits fixed	Timing
11	$(4, 0)$	28	$(6, 0)$	1.7
12	$(8, 0)$	59	$(12, 0)$	48.9
13	$(16, 0)$	$\sim$ 8 hours	$(14, 6)$	86.8
13	$(8, 4)$	74	$(16, 6)$	17
14	$(8, 12)$	58	$(16, 16)$	51.4
15	$(8, 20)$	83	$(16, 24)$	328.2
16	$(8, 28)$	60	-	-

– Let $D(P, P') = \{\delta_j^{i+1}(P, P') + x_{j-1}^i \delta_{j-8}^i + x_{j-8}^i \delta_{j-1}^i + \delta_{j-1}^i \delta_{j-8}^i + \delta_{j-2}^i + \delta_j^{i-1} : i = 0, 1, 2, \ldots, r, j = 0, 1, 2, \ldots, n-1\}$.
– Find the set T of linear equations using ElimLin for the set $\bigcup D(P, P')$ over all the pairs (P, P') with Hamming distance 1.
– Consider the set $X = \bigcup S(P, C) \cup \bigcup D(P, P') \cup L \cup T$ where the unions are taken over all the plaintext/ciphertext pairs.
– Simplify X using the short linear equations (say of length at most 3).
– Convert the system into CNF and feed it to a SAT solver.

We performed experiments for Simon-32/64. Table 12 shows the timings of the SAT solvers with and without the differential equations. Note that we used the same plaintext differences and fixed key bits (for 13 and more rounds) as in Section 3. For each experiment, we performed 20 different random instances and record the average SAT solver timings below.

7 Discussion and conclusion

In this paper, we proposed a more efficient method to solve algebraic equations with SAT solvers. When combined with chosen plaintext attacks, our technique enhanced the current state-of-the-art of algebraic cryptanalysis on lightweight block ciphers. We demonstrated the effectiveness of our approach by improving the algebraic attack on both Present cipher and Simon cipher (see Tables 5 and 7, respectively).

To summarize, the main components of our framework comprise the following:

– Construct a differential trail table to relate a difference bit in terms of intermediate input bits as well as difference bits of preceding rounds. Our framework is more effective such a differential trail table contains many linear relations.
– Use of ElimLin to find linear relations among the variables to be added as additional clauses for the SAT solvers;
– A good representation of the cipher, typically involving short equations with small degrees.

Finally, we remark that unlike prior work on algebraic differential cryptanalysis that adds linear equations representing the input and output differences of a differential characteristic with a certain probability, our approach considers the entire differential trail and is independent on the associated probability.

Following the promising results from this work, several directions can be explored. First, we have only considered a class of weak keys for our experiments on Simon in this paper. One can certainly generalize our attacks to generic keys and estimate the complexities of the generic attacks. Next, it will be interesting to apply our techniques, possibly with some adaptations, to block ciphers such as DES or AES. Finally, we have seen that adding in differential equations is helpful to speed up the SAT solving process. As such, it will be interesting to investigate the effectiveness of including short higher-order differential equations with degrees greater than 2.

References

1. Albrecht, M.R., Cid, C.: Algebraic techniques in differential cryptanalysis. In: O. Dunkelman (ed.) Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers, pp. 193–208. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
2. Bard, G.V., Courtois, N., Jefferson., C.: Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over gf(2) via sat-solvers. IACR Cryptology ePrint Archive **2007**, 24 (2007)
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. IACR Cryptology ePrint Archive (2013). URL https://eprint.iacr.org/2013/404
4. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, The Netherlands, The Netherlands (2009)
5. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '90, pp. 2–21 (1991)
6. Biryukov, A., Roy, A., Velichkov, V.: Differential Analysis of Block Ciphers SIMON and SPECK. In: FSE (2014)
7. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. Springer (2007)
8. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. **24**(3-4), 235–265 (1997). DOI 10.1006/jsco.1996.0125. URL http://dx.doi.org/10.1006/jsco.1996.0125. Computational algebra and number theory (London, 1993)
9. Buchberger, B.: Ein algorithmus zum auffinden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal. Dissertation der Universitat Innsbruck (1965)
10. Cannière, C.D., Preneel, B.: Trivium. In: New Stream Cipher Designs - The eSTREAM Finalists, pp. 244–266 (2008)
11. Chu, Z., Chen, H., Wang, X., Dong, X., Li, L.: Improved integral attacks on SIMON32 and SIMON48 with dynamic key-guessing techniques. Security and Communication Networks **2018**, 5160237:1–5160237:11 (2018)
12. Courtois, N., Bard, G.V., Wagner, D.A.: Algebraic and slide attacks on KeeLoq. In: FSE, *Lecture Notes in Computer Science*, vol. 5086, pp. 97–115. Springer (2008)
13. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Advances in Cryptology–EUROCRYPT 2000, pp. 392–407. Springer (2000)
14. Courtois, N., Mourouzis, T., Song, G., Sepehrdad, P., Susil, P.: Combined Algebraic and Truncated Differential Cryptanalysis on Reduced-round Simon. In: SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, 28-30 August, 2014, pp. 399–404 (2014)
15. Courtois, N.T., Bard, G.V.: Algebraic cryptanalysis of the data encryption standard. In: Cryptography and Coding, pp. 152–169. Springer (2007)
16. Courtois, N.T., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Advances in Cryptology–EUROCRYPT 2003, pp. 345–359. Springer (2003)
17. Courtois, N.T., Nohl, K., O'Neil, S.: Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards (2008). URL http://eprint.iacr.org/2008/166. N.courtois@ucl.ac.uk 13983 received 13 Apr 2008, last revised 14 Apr 2008
18. Courtois, N.T., Patarin, J.: About the XL algorithm over GF(2). In: Topics in Cryptology–CT-RSA 2003, pp. 141–157. Springer (2003)
19. Courtois, N.T., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In: Advances in Cryptology–ASIACRYPT 2002, pp. 267–287. Springer (2002)
20. Courtois, N.T., Sepehrdad, P., Sušil, P., Vaudenay, S.: Elimlin algorithm revisited. In: Fast Software Encryption, pp. 306–325. Springer (2012)
21. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher square. In: International Workshop on Fast Software Encryption, pp. 149–165. Springer (1997)
22. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
23. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
24. Een, N., Sorensson, N.: An extensible sat-solver. In: E. Giunchiglia, A. Tacchella (eds.) SAT, *Lecture Notes in Computer Science*, vol. 2919, pp. 502–518. Springer (2003)
25. Faugere, J.C.: A new efficient algorithm for computing gröbner bases (F4). Journal of pure and applied algebra **139**(1), 61–88 (1999)
26. Faugere, J.C.: A new efficient algorithm for computing gröbner bases without reduction to zero (f 5). In: Proceedings of ISSAC, pp. 75–83. ACM (2002)
27. Fengjuan, C., Xiao-Shan, G., Chunming, Y.: A characteristic set method for solving boolean equations and applications in cryptanalysis of stream ciphers*. Journal of Systems Science and Complexity **21**(2), 191–208 (2008)
28. Ganesh, V., Liang, J.H.: Maplesat. https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/. Accessed: 2017-11-28
29. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. Int. J. Wire. Mob. Comput. **2**(1), 86–93 (2007)
30. Huang, Z., Sun, Y., Lin, D.: On the efficiency of solving boolean polynomial systems with the characteristic set method. arXiv preprint arXiv:1405.4596 (2014)

31. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON block cipher family. In: Annual Cryptology Conference, pp. 161–185. Springer (2015)
32. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A cryptanalysis of PRINT cipher: the invariant subspace attack. In: Annual Cryptology Conference, pp. 206–221. Springer (2011)
33. Li, C., Preneel, B.: Improved interpolation attacks on cryptographic primitives of low algebraic degree. In: K.G. Paterson, D. Stebila (eds.) Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Waterloo, ON, Canada, August 12-16, 2019, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 11959, pp. 171–193. Springer (2019)
34. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology, EUROCRYPT '93, pp. 386–397 (1994)
35. Murphy, S., Robshaw, M.J.B.: Essential algebraic structure within the AES. In: Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings, pp. 1–16 (2002)
36. Nakahara Jr, J., Sepehrdad, P., Zhang, B., Wang, M.: Linear (hull) and algebraic cryptanalysis of the block cipher PRESENT. In: Cryptology and Network Security, pp. 58–75. Springer (2009)
37. Raddum, H.: Algebraic Analysis of the Simon Block Cipher Family. In: Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings, pp. 157–169 (2015)
38. Sage: An ANF to CNF Converter using a Dense/Sparse Strategy. http://doc.sagemath.org/html/en/reference/-sat/sage/sat/converters/polybori.html. Accessed: 2017-11-28
39. Sepehrdad, P.: Statistical and algebraic cryptanalysis of lightweight and ultra-lightweight symmetric primitives. Ph.D. thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE (2012)
40. Shimoyama, T., Kaneko, T.: Quadratic relation of s-box and its application to the linear attack of full round DES. In: Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings, pp. 200–211 (1998)
41. Soos, M.: Cryptominisat 5.0.1. https://www.msoos.org/2016/09/cryptominisat-5-0-1-released-with-mit-license/. Accessed: 2017-11-28
42. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: O. Kullmann (ed.) SAT, *Lecture Notes in Computer Science*, vol. 5584, pp. 244–257. Springer (2009)
43. Yeo, S.L., Li, Z., Khoo, K., Low, Y.B.: An enhanced binary characteristic set algorithm and its applications to algebraic cryptanalysis. In: Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings, pp. 518–536 (2017)