Parity (XOR) Reasoning for the Index Calculus Attack

Monika Trimoska, Sorina Ionica, Gilles Dequen

Parity (XOR) Reasoning for the Index Calculus Attack ***

Monika Trimoska and Sorina Ionica and Gilles Dequen

Laboratoire MIS, Université de Picardie Jules Verne, Amiens, France
{monika.trimoska,gilles.dequen, sorina.ionica}@u-picardie.fr

Abstract. Cryptographic problems can often be reduced to solving boolean polynomial systems, whose equivalent logical formulas can be treated using SAT solvers. Given the algebraic nature of the problem, the use of the logical XOR operator is common in SAT-based cryptanalysis. Recent works have focused on advanced techniques for handling parity (XOR) constraints, such as the Gaussian Elimination technique. First, we propose an original XOR-reasoning SAT solver, named WDSat,[1] dedicated to a specific cryptographic problem. Secondly, we show that in some cases Gaussian Elimination on SAT instances does not work as well as Gaussian Elimination on algebraic systems. We demonstrate how this oversight is fixed in our solver, which is adapted to read instances in algebraic normal form (ANF). Finally, we propose a novel preprocessing technique based on the Minimal Vertex Cover Problem in graph theory. This preprocessing technique is, within the framework of multivariate Boolean polynomial systems, used as a DLL branching selection rule that leads to quick linearization of the underlying algebraic system. Our benchmarks use a model obtained from cryptographic instances for which a significant speedup is achieved using the findings in this paper. We further explain how our preprocessing technique can be used as an assessment of the security of a cryptographic system.

1 Introduction

Cryptanalysis is the study of methods to decrypt a ciphertext without any knowledge of the secret key. Academic research in cryptanalysis is focused on deciding whether a cryptosystem is secure enough to be used in the real world. In addition, a good understanding of the complexity of a cryptographic attack allows us to determine the secret key length, making sure that no cryptanalytic effort can find the key in a feasible amount of time. Recommendations for minimum key length requirements given by various academic and governmental organizations [4] are based on the complexity of known attacks.

* The final authenticated version is available online at https://doi.org/10.1007/978-3-030-58475-7_45

** This work is co-financed by the European Union under the 2014/2020 European Regional Development Fund (FEDER).

[1] Weil Descent SAT solving

In recent years, constraint programming (CP) techniques have been used in the cryptanalysis of both public and secret key cryptosystems. A first example in the field of differential cryptanalysis is given by the work of Gerault *et al.* [18,16,17] who showed how to use CP for solving the optimal related-key differential characteristic problem. Using the CP model presented in their work, all optimal related-key differential characteristics for AES-128, AES-192 and AES-256 can be computed in a few hours [17]. We also note the work of Lui *et al.* [20,21], in which a CP model is used to aid the Tolerant Algebraic Side-Channel Analysis, which is a combination of algebraic and side-channel analysis.

In a second line of research, Boolean satisfiability (SAT) solvers have found use in algebraic cryptanalysis. Algebraic cryptanalysis denotes any technique which reduces a cryptographic attack to the problem of solving a multivariate Boolean polynomial system. A common approach for solving these systems is to use Gröbner basis algorithms [12], exhaustive search [6] or hybrid methods [2]. These methods have been compared against SAT solving techniques for attacks on various symmetric cryptosystems such as Bivium, Trivium, Grain. Recent work has also focused on combining algebraic and SAT solving techniques [7]. In public-key cryptography, SAT solvers have been considered for attacking binary elliptic curve cryptosystems using the index calculus attack [14]. In this paper, we tackle this last-mentioned application.

In this paper, we propose a built-from-scratch SAT solver dedicated to solving an important step of the index calculus attack. The solver, named WDSat, is adapted for XOR-reasoning and reads formulas in ANF form. In addition, we show certain limitations of the Gaussian Elimination (GE) technique in XOR-enabled SAT solvers by pointing out a canceling property that is present in algebraic resolution methods but is overseen in current SAT-based GE implementations. We refer to this canceling property as the XG-ext method and we show how it is implemented in our solver. In implementations, the XG-ext method comes at a high computational cost and is thus useful only for benchmarks where it reduces significantly the number of conflicts. Finally, we introduce a graph theory-based preprocessing technique, specifically designed for multivariate Boolean polynomial systems, that allows us to further accelerate the resolution of our benchmarks. This preprocessing technique is designed to allow a rapid linearization of the underlying algebraic system and should be used coupled with the XG-ext method. In fact, when the XG-ext method is not applied, the positive outcome of the preprocessing technique cannot be guaranteed. To confirm, we perform experiments using CryptoMiniSat [27] coupled with our preprocessing technique and show that this combination yields slower running times than CryptoMiniSat alone. Experimental results in Section 6 show that the solver presented in this paper outperforms all existing solving approaches for the introduced problem. These approaches include Gröbner basis techniques [12] and state-of-the-art SAT solvers: MiniSat [11], Glucose [1], MapleLCMDistChronoBT [23], CaDiCaL [3] and CryptoMiniSat [27].

2 Background

Index Calculus In cryptanalysis, the index calculus algorithm is a well-known method for attacking factoring and elliptic curve discrete logarithms, two computational problems which are at the heart of most used public-key cryptosystems. When performing this attack for elliptic curve discrete logarithms, a crucial step is the point decomposition phase. As proposed by Gaudry [15] and Diem [10] independently, a point on the elliptic curve can be decomposed into m other points by solving Semaev's $(m+1)$-th summation polynomial [25], that we denote by S_{m+1}. For elliptic curves defined over binary fields, the second and the third summation polynomials are defined as follows:

$$\begin{aligned} &S_2(X_1, X_2) = X_1 + X_2, \\ &S_3(X_1, X_2, X_3) = X_1^2X_2^2 + X_1^2X_3^2 + X_1X_2X_3 + X_2^2X_3^2 + 1. \end{aligned} \tag{1}$$

For $m > 3$, the m-th summation polynomial is computed by using the following recursive formula:

$$\begin{aligned} &S_m(X_1, \ldots, X_m) = \\ &Res_X(S_{m-k}(X_1, \ldots, X_{m-k-1}, X), S_{k+2}(X_{m-k}, \ldots, X_m, X)), \end{aligned} \tag{2}$$

where Res_X denotes the resultant of two polynomials with respect to the X variable and $1 \leq k \leq m-3$. The zeros of this polynomial will give the x-coordinates of points on the elliptic curve as elements in $\mathbb{F}_{2^n}$. From an implementation point of view, these will be represented as n-bit vectors. In index calculus attacks, the common approach is to decompose a random point given by an n-bit vector x-coordinate into m points whose x-coordinates write as l-bit vectors, with $l \sim \frac{n}{m}$ (see for instance [13,24]). With this choice of parameters, the problem of decomposing a random point by finding the zeros of S_{m+1} can be reduced to solving a system of n Boolean polynomials with ml variables.

We recall that a multivariate Boolean polynomial system is a system of polynomials in several variables and whose coefficients are in $\mathbb{F}_2$ (see for instance [19]). The following example shows a Boolean polynomial system of three equations in the variables $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$:

$$\begin{aligned} &\mathbf{x}_1 + \mathbf{x}_2 \cdot \mathbf{x}_3 = 0 \\ &\mathbf{x}_1 \cdot \mathbf{x}_2 + \mathbf{x}_2 + \mathbf{x}_3 = 0 \\ &\mathbf{x}_1 + \mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 = 0. \end{aligned}$$

In the literature, the modelisation process allowing to obtain a Boolean polynomial system from a polynomial with coefficients in $\mathbb{F}_{2^n}$ (here the summation polynomial) is called a *Weil Restriction* [15] or *Weil Descent* [24]. The polynomial systems obtained in this way serve as our starting point for deriving SAT instances.[2]

[2] Our C code for generating these instances is publicly available [28].

XOR-Enabled SAT Solvers A Boolean polynomial system can be rewritten as a conjunction of logical formulas in algebraic normal form (ANF) as follows: multiplication in $\mathbb{F}_2$ $(\cdot)$ becomes the logical AND operation $(\wedge)$ and addition in $\mathbb{F}_2$ $(+)$ becomes the logical XOR $(\oplus)$. The elements 0 and 1 in $\mathbb{F}_2$ correspond to $\bot$ and $\top$, respectively. Consequently, solving a multivariate Boolean polynomial system is equivalent to solving a conjunction of logical formulas in ANF form. To date, few SAT solvers are adapted to tackle formulas in ANF. A common approach is to transform the ANF form in a CNF-XOR form, which is a conjunction of CNF and XOR clauses. In order to do this, every conjunction of two or more literals $x_1 \wedge x_2 \wedge \ldots \wedge x_k$ has to be replaced by an additional and equivalent variable x' such that $x' \Leftrightarrow x_1 \wedge x_2 \wedge \ldots \wedge x_k$. This equivalence can be rewritten in CNF using a three-step transformation. First, the equivalence is decomposed into two implications:

$$\begin{aligned}&(x' \Rightarrow x_1 \wedge x_2 \wedge \ldots \wedge x_k) \wedge\\ &(x_1 \wedge x_2 \wedge \ldots \wedge x_k \Rightarrow x').\end{aligned}$$

Then, the material implication rule is applied:

$$\begin{aligned}&(\neg x' \vee (x_1 \wedge x_2 \wedge \ldots \wedge x_k)) \wedge\\ &(\neg(x_1 \wedge x_2 \wedge \ldots \wedge x_k) \vee x').\end{aligned}$$

Finally, using distribution on the first, and De Morgan's law on the second constraint, we obtain the following CNF formula:

$$\begin{aligned}&(\neg x' \vee x_1) \wedge\\ &(\neg x' \vee x_2) \wedge\\ &\ldots\\ &(\neg x' \vee x_k) \wedge\\ &(\neg x_1 \vee \neg x_2 \vee \ldots \vee \neg x_k \vee x').\end{aligned} \tag{3}$$

When we substitute all occurrences of conjunctions in an XOR clause by an additional variable, we obtain a formula in CNF-XOR form. This is the form used in the CryptoMiniSat solver [27], which is an extension of the MiniSat solver [11] specifically designed to work on cryptographic problems.

Example 1. Let us consider the Boolean polynomial system:

$$\begin{aligned}&\mathbf{x}_1 + \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_5 + \mathbf{x}_6 + 1 = 0\\ &\mathbf{x}_3 + \mathbf{x}_5 + \mathbf{x}_6 = 0.\end{aligned} \tag{4}$$

One additional variable x' needs to be introduced to substitute the monomial $\mathbf{x}_2 \cdot \mathbf{x}_3$. The corresponding CNF-XOR form for this Boolean system is a conjunction

of the following clauses:

$$\begin{aligned} &x' \vee \neg x_2 \vee \neg x_3 \\ &\neg x' \vee x_2 \\ &\neg x' \vee x_3 \\ &x_1 \oplus x' \oplus x_5 \oplus x_6 \\ &x_3 \oplus x_5 \oplus x_6 \oplus \top. \end{aligned} \tag{5}$$

Finally, one could, of course, consider generic solvers (i.e. MiniSat [11], Glucose [1]) for solving cryptographic problems, but this approach needs to further transform the CNF-XOR model to a CNF one. Transforming an XOR-clause with k literals in CNF representation is a well-known process that gives 2^{k-1} OR-clauses of k literals.

Notation. For simplicity, in the remainder of this paper we will omit the multiplication operator $\cdot$ whenever its use in monomials is implicit. Moreover, due to equivalence between the Boolean polynomial systems and the ANF form, these will be used interchangeably.

3 The WDSat solver

Our WDSat solver is based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [8], which is a state-of-the-art complete SAT solving technique. The solver is designed to treat ANF formulae derived from the Weil Descent modelisation of cryptographic attacks, hence its name: WDSat. The code for the WDSat solver is written in C and is publicly available [29].

WDSat implements three reasoning modules. These include the module for reasoning on the CNF part of the formula and the so-called XORSET and XOR-GAUSS (XG) modules designed for reasoning on XOR constraints. The CNF module is designed to perform classic unit propagation on OR-clauses. The XORSET module performs the operation equivalent to unit propagation, but adapted for XOR-clauses. Practically, this consists in checking the parity of the current interpretation and propagating the unassigned literal. Finally, the XG module is designed to perform GE on the XOR constraints dynamically. We also implement an XG extension, described in Section 4. The following is a detailed explanation of this module.

XOR clauses are normalized and represented as equivalence classes. Recall that an XOR-clause is said to be in *normal form* if it contains only positive literals and does not contain more than one occurrence of each literal. Since we consider that all variables in a clause belong to the same equivalence class (EC), we choose one literal from the EC to be the *representative*. An XOR-clause $(x_1 \oplus x_2 \oplus ... \oplus x_n) \Leftrightarrow \top$ rewrites as

$$x_1 \Leftrightarrow (x_2 \oplus x_3 \oplus ... \oplus x_n \oplus \top). \tag{6}$$

Finally, we replace all occurrences of a representative of an XOR clause with the right side of the equivalence. Applying this transformation, we obtain a simplified system having the following property: a representative of an EC will never be present in another EC.

Let R be the set of representatives and C be the set of clauses. R and C hold the right-hand side and the left-hand side of all equations of type (6) respectively. We denote by C_x the clause in C that is equivalent to x. In other words, C_x is the right-hand side of the EC that has x as representative. Finally, we denote by $var(C_x)$ the set of literals (plus a $\top/\bot$ constant) in the clause C_x and $C[x_1/x_2]$ denotes the following substitution of clauses: for all $C_i \in C$ containing x_1, $C_i \leftarrow C_i \oplus x_1 \oplus x_2$, i.e. x_1 is replaced by x_2 in C_i. When we replace a literal x_1 by a clause C_{x_2}, we adopt a similar notation: $C[x_1/C_{x_2}]$.

Thus, assigning a literal x_1 to $\top$ leads to using one of the rules in Table 1, depending on whether x_1 belongs to R or not. In both cases, propagation occurs when : $\exists\, x_i \neq x_1$ *s.t.* $var(C_{x_i}) = \top/\bot$. Conflict occurs when one constraint leads to the propagation of x_i to $\top$ and another constraint leads to the propagation of x_i to $\bot$.

Table 1 presents inference rules for performing GE in the XG module of WDSat. Applying these rules allows us to maintain the property of the system which states that a representative of an EC will never be present in another EC. For clarity of the notation, the first column of this table contains the premises, the second one contains the conclusion and the third one is an update on the set R which has to be performed when the inference rule is used.

Table 1. Gaussian elimination inference rules.

Premises	Conclusions on C	Updates on R
x_1, C $x_1 \not\in R$	$C[x_1/\top]$	N/A
x_1, C $x_1 \in R$ $x_2 \in var(C_{x_1})$	$C_{x_2} \leftarrow C_{x_1} \oplus x_2 \oplus \top$ $C[x_2/C_{x_2}]$	$R \leftarrow R \setminus \{x_1\}$ $R \leftarrow R \cup \{x_2\}$

We denote by k the number of variables in a XOR-CNF formula. At the implementation level, XOR-clauses are represented as $(k+1)$-bit vectors: a bit for every variable and one for a $\top$, $\bot$ constant. Clauses are stored in an array indexed by the representatives. This representation allows us to perform GE only by XOR-ing bit-vectors and flipping the clause constant. For a compact representation of the $(k+1)$-bit vector we used an array of $\lceil (k+1)/64 \rceil$ integers.

Example 2. Let $k = 7$ and let us consider $x_2 \Leftrightarrow \top \oplus x_1 \oplus x_3 \oplus x_5$. Then we have that $var(C_{x_2}) = \{\top, x_1, x_3, x_5\}$ and the bit-vector representing this clause is 11010100, where the $\top$, $\bot$ constant takes the zero position. Assigning x_1 to $\top$ is equivalent to introducing the constraint $x_1 \oplus \top$. We apply the first rule, simply

by XOR-ing this bit-vector with a mask of the form 11000000. The resulting vector is 00010100, which corresponds to $var(C_{x_2}) = \{\bot, x_3, x_5\}$.

Our DPLL-based solver assigns a truth value to each variable in a formula F, recursively building a binary search tree. After each assignment, either the formula is simplified and other truth values are inferred or a conflict occurs. In the case of a conflict, the last assignment has to be undone for each module via a backtracking procedure. In Algorithm 1, we detail the ASSIGN function of WDSat, which is at the core of the DPLL algorithm. This function synchronises all three modules in the following manner. First, the truth value is assigned in the CNF module and truth values of other variables are propagated. Next, the truth value of the initial variable, as well as the propagated ones are assigned in the XORSET module. If the XOR-adapted unit propagation discovers new truth values, they are assigned in the CNF module, going back to step one. We go back and forth with this process until the two modules are synchronized and there are no more propagations left. Finally, the list of all inferred literals is transferred to the XG module. If the XG module finds new XOR-implied literals, the list is sent to the CNF module and the process is restarted. If a conflict occurs in any of the reasoning modules, the ASSIGN function fails and a backtracking procedure is launched. We briefly detail the other functions used in the pseudo-code. There is a SET_IN function for each module which takes as input a list of literals and a propositional formula F and sets all literals in this list to $\top$ in the corresponding modules. Through this assignment, the function also infers truth values of other literals, according to the specific rules in different modules. For instance, the SET_IN function for the XG module (SET_IN_XG) implements the rules in Table 1, performing a GE on the system. Finally, the LAST_ASSIGNED function in each module returns the list of literals that were assigned during the last call to the respective SET_IN function.

4 The XG-ext Method

In this section, we show how we extend our XG module. First, we present the motivation for this work by giving an example of a case where GE in SAT solvers has certain limitations compared to Algebraic GE. Secondly, we propose a solution to overcome these limitations and we implement it in our solver to develop the XORGAUSS-ext method (XG-ext in short). To introduce new rules for this method, we use the same notation as in Section 3.

Gaussian elimination on a Boolean polynomial system consists in performing elementary operations on equations with the goal of reducing the number of equations as well as the number of terms in each equation. We cancel out terms by adding (XOR-ing) one equation to another. GE can be performed on instances in CNF-XOR form in the same way that it is performed on Boolean polynomial systems presented in algebraic writing. However, we detected a case where a possible cancellation of terms is overseen due to the CNF-XOR form.

Algorithm 1 Function ASSIGN(F, x) : Assigning a truth value to a literal x in a formula F, simplifying F and inferring truth values for other literals.

Input: The propositional formula F, a literal x
Output: $\bot$ if a conflict is reached, $\top$ and a simplified F otherwise

```
1: to_set ← {x}.
2: to_set_in_XG ← {x}.
3: while to_set ≠ ∅ do
4:    while to_set ≠ ∅ do
5:       if SET_IN_CNF(to_set, F) → ⊥ then
6:          return (⊥, – ).
7:       end if
8:       to_set ← LAST_ASSIGNED_IN_CNF().
9:       to_set_in_XG ← to_set.
10:      if SET_IN_XORSET(to_set, F)→ ⊥ then
11:         return (⊥, – ).
12:      end if
13:      to_set ← LAST_ASSIGNED_IN_XORSET().
14:      to_set_in_XG ← to_set ∪ to_set_in_XG.
15:   end while
16:   if SET_IN_XG(to_set_in_XG, F)→ ⊥ then
17:      return (⊥, – ).
18:   end if
19:   to_set ← LAST_ASSIGNED_XG().
20: end while
21: return (⊤, F).
```

Example 3. We will reuse the Boolean polynomial system in Example 1 to demonstrate a case where a cancellation of a term is missed by a XOR-enabled SAT solver. Let us consider that in Equation (4), we try to assign the value of 1 to $\mathbf{x}_2$. As the monomial $\mathbf{x}_2\mathbf{x}_3$ will be equal to 1 only if both terms $\mathbf{x}_2$ and $\mathbf{x}_3$ are equal to 1, we get the following result:

$$\mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_5 + \mathbf{x}_6 + 1 = 0$$
$$\mathbf{x}_3 + \mathbf{x}_5 + \mathbf{x}_6 = 0.$$

After XORing the two equations, we infer that $\mathbf{x}_1 = 1$.
However, when we assign x_2 to $\top$ in the corresponding CNF-XOR clause in Equation (5), as per unit propagation rules, we get the following result:

$$x' \vee \neg x_3$$
$$\neg x' \vee x_3$$
$$x_1 \oplus x' \oplus x_5 \oplus x_6$$
$$x_3 \oplus x_5 \oplus x_6 \oplus \top.$$

When we XOR the second clause to the first one we can not infer that x_1 is $\top$ at this point.

Note that $(x' \vee \neg x_3) \wedge (\neg x' \vee x_3)$ rewrites as $x' \Leftrightarrow x_3$, but if the solver does not syntactically search for this type of occurrences regularly, x' will not be replaced by x_3. Moreover, this type of search adds an additional computational cost to the resolution.

Omissions as the one detailed in Example 3 can occur every time a variable is set to $\top$. As a result, we define the following rule with the goal to improve the performance of XOR-enabled SAT solvers:

$$\frac{x' \quad x_1 \Leftrightarrow (x' \wedge x_2)}{x_1 \Leftrightarrow x_2}. \tag{7}$$

This rule can be generalised for the resolution of higher-degree Boolean polynomial systems:

$$\frac{x' \quad x_1 \Leftrightarrow (x' \wedge x_2 \wedge \ldots \wedge x_d)}{x_1 \Leftrightarrow (x_2 \wedge \ldots \wedge x_d)}. \tag{8}$$

Even though these rules are standard in Boolean logic, they are presently not implemented in XOR-enabled SAT solvers. Note that when a solver takes as input an instance in CNF-XOR form, the second premise is lost or has to be inferred by syntactic search. To have knowledge of the second premise, the solver needs to read the instance in ANF. To this purpose, we defined a new ANF input format for SAT solvers.

This extension of the XG module is implemented as part of the SET_IN_XG function used in the ASSIGN algorithm. The following is a detailed explanation of how the rule in Equation (7) is applied in our implementation. Recall that the XG module has the following property: a representative of an EC will never be present in another EC. This property will be maintained in the XG-ext method as well. Using the conclusion in Equation (7), we derive in Table 2 six inference rules that allow us to perform the substitution of a variable x_1 by a variable x_2 while maintaining the unicity-of-representatives property. Applying one of the inference rules in Table 2 can result in conflict or it can propagate a newly discovered truth value. Note that $var(C_{x_1} \oplus C_{x_2})$ is given by the symmetric difference $(var(C_{x_1}) \cup var(C_{x_2})) \setminus (var(C_{x_2}) \cap var(C_{x_1}))$.

5 Our Preprocessing Technique

Let us reconsider the DPLL-based algorithm. It is well known that the number of conflicts needed to prove the inconsistency is correlated to the order in which the variables are assigned. Among the state-of-the-art branching rules you can find two categories according to the type of heuristics. The first are based on Maximum number of Occurrences in the Minimum clauses Size (MOMs) whereas the second adopt the Variable State Independent Decaying Sum (VSIDS) branching heuristic.

In this work, we were interested in developing a criterion for defining the order of variables on CNF-XOR instances derived from Boolean polynomial systems.

Table 2. Inference rules for the substitution of x_1 by x_2.

Premises	Conclusions on C	Updates on R
C, $x_1 \Leftrightarrow x_2$ $x_1 \notin R$ $x_2 \notin R$	$C[x_1/x_2]$	N/A
C, $x_1 \Leftrightarrow x_2$ $x_1 \in R$ $x_2 \notin R$ $x_2 \notin var(C_{x_1})$	$C_{x_2} \leftarrow C_{x_1}$ $C[x_2/C_{x_2}]$	$R \leftarrow R \setminus \{x_1\}$ $R \leftarrow R \cup \{x_2\}$
C, $x_1 \Leftrightarrow x_2$ $x_1 \in R$ $x_2 \notin R$ $x_2 \in var(C_{x_1})$ $x_3 \in var(C_{x_1})$	$C_{x_3} \leftarrow C_{x_1} \oplus x_2 \oplus x_3$ $C[x_3/C_{x_3}]$	$R \leftarrow R \setminus \{x_1\}$ $R \leftarrow R \cup \{x_3\}$
C, $x_1 \Leftrightarrow x_2$ $x_1 \notin R$ $x_2 \in R$ $x_1 \notin var(C_{x_2})$	$C[x_1/C_{x_2}]$	N/A
C, $x_1 \Leftrightarrow x_2$ $x_1 \notin R$ $x_2 \in R$ $x_1 \in var(C_{x_2})$ $x_3 \in var(C_{x_2})$	$C_{x_3} \leftarrow C_{x_2} \oplus x_1 \oplus x_3$ $C[x_1/x_2, x_3/C_{x_3}]$	$R \leftarrow R \setminus \{x_2\}$ $R \leftarrow R \cup \{x_3\}$
C, $x_1 \Leftrightarrow x_2$ $x_1 \in R$ $x_2 \in R$ $x_3 \in var(C_{x_1} \oplus C_{x_2})$	$C_{x_3} \leftarrow C_{x_1} \oplus C_{x_2} \oplus x_3$ $C[x_3/C_{x_3}]$	$R \leftarrow R \setminus \{x_1, x_2\}$ $R \leftarrow R \cup \{x_3\}$

We set the goal to choose branching variables that will lead as fast as possible to a linear polynomial system, which can be solved using GE in polynomial time. In terms of SAT solving, choosing this order for branching will cancel out all clauses in the CNF part of the formula as a result of unit propagation. When only the XOR part of the CNF-XOR formula is left, the solver performs GE on the remaining XOR constraints in polynomial time.

After setting this goal, choosing which variable to assign next according to the number of their occurrences in the system is no longer an optimal technique. We explain this idea on an example. For simplicity, we only use the Boolean algebra terminology in this section. However, the methods described are applicable to both SAT solving and algebraic techniques based on the process of recursively making assumptions on the truth values of variables in the system (as with the DPLL algorithm).

Example 4. Consider the following Boolean polynomial system:

$$\begin{aligned} &\mathbf{x}_1 + \mathbf{x}_2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_4\mathbf{x}_5 = 0 \\ &\mathbf{x}_1 + \mathbf{x}_2\mathbf{x}_3 = 0 \\ &\mathbf{x}_1 + \mathbf{x}_3\mathbf{x}_5 + \mathbf{x}_6 = 0 \\ &\mathbf{x}_1 + \mathbf{x}_2\mathbf{x}_5\mathbf{x}_6 + \mathbf{x}_6 = 0 \end{aligned} \tag{9}$$

In this example, the variable with the highest number of occurrences is $\mathbf{x}_1$. However, $\mathbf{x}_1$ does not occur in any monomial of degree > 1. Thus, assigning first $\mathbf{x}_1$ does not contribute to the linearization of the system and we need to find a more suitable criterion.

The solution we propose is inspired by graph theory. Particularly, we identified a parallel between the problem of defining the order in which the variables are assigned and the Minimal Vertex Cover Problem (MVC).

In graph theory, a *vertex cover* is a subset of vertices such that for every edge (v_i, v_j) of the graph, either v_i or v_j is in the vertex cover. Given an undirected graph, the Minimum Vertex Cover Problem is a classic optimization problem of finding a vertex cover of minimal size.

An undirected graph is derived from a Boolean polynomial system as follows.

- Each variable $\mathbf{x}_i$ from the system becomes a vertex v_i in the graph G.
- An edge (v_i, v_j) is in G if and only if (in the corresponding Boolean system) there exists a monomial of degree $n \geq 2$ which contains both $\mathbf{x}_i$ and $\mathbf{x}_j$.

When we use this representation of a Boolean polynomial system as a graph, a vertex cover defines a subset of variables whose assignment will result in a linear Boolean polynomial system in the remaining non-assigned variables. Consequently, finding the MVC of the graph is equivalent to finding the minimal subset of variables one has to assign to obtain a linear system.

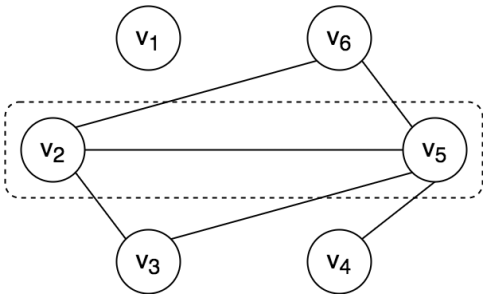

Fig. 1. Graph derived from Example 4

Figure 1 shows the graph derived from Example 4. The MVC of this graph is $\{v_2, v_5\}$. As a result, when all variables in the subset $\{\mathbf{x}_2, \mathbf{x}_5\}$ are assigned, the

remaining polynomial system is linear. We give here the system derived after the assignment $\mathbf{x}_2 = 1$ and $\mathbf{x}_5 = 1$.

$$\begin{aligned} \mathbf{x}_1 + \mathbf{x}_3 &= 0 \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_6 &= 0 \\ \mathbf{x}_1 &= 0. \end{aligned}$$

For all other possible assignments of $\mathbf{x}_2$ and $\mathbf{x}_5$, we obtain similar linear systems.

Defining the order of branching variables will serve as a preprocessing technique that consists in (i) deriving a graph from a Boolean polynomial system and (ii) finding the MVC of the resulting graph. During the solving process, variables corresponding to vertices in the MVC are assigned first. Even though the MVC problem is NP-complete, its execution for graphs derived from cryptographic models always finishes in negligible running time due to the small number of variables. Our solver does not use any other MOMs or VSIDS-based heuristic during the solving process, as the order of the branching variables is predetermined by the MVC preprocessing technique.

When variables are assigned in the order defined by this preprocessing technique, the worst-case time complexity of a DPLL-based algorithm drops from $O(2^k)$ to $O(2^{k'})$, where k' is the number of vertices in the MVC set. Note that the MVC of a complete graph is equal to the number of its vertices. Consequently, when the corresponding graph of a Boolean polynomial system is a complete graph, solving the system using this preprocessing technique is as hard as solving the system without it.

Finding the MVC corresponding to a Boolean polynomial system can also be used as an assessment of the security of the underlying cryptosystem. Indeed, an exhaustive search on a subset of variables, which are the variables in the MVC, results in linear systems that can be solved in polynomial time. This straightforward approach yields an upper bound on the complexity of solving the system at hand. In short, to assess the security of a cryptographic system, assuming that this is based on solving the Boolean polynomial system first, one computes the MVC of this system and deduces that $O(2^{k'})$ is a bound on the complexity of the attack.

6 Experimental Results

To support our claims, we experimented with benchmarks derived from two variants of the index calculus attack on the discrete logarithm problem over binary elliptic curves. As explained in Section 2, a SAT solver can be used for solving Semaev's summation polynomials in the point decomposition phase. Our model is derived from the Boolean multivariate polynomial system given by the $m+1$-th summation polynomial, with $m \geq 2$. This model has previously been examined in [14]. We compare the WDSat solver presented in this paper to the following approaches: the best currently available implementation of Gröbner

basis (F4 [12] in MAGMA [5]), the solvers MiniSat, [11], Glucose [1], MapleLCMDistChronoBT [23], CaDiCaL [3] and CryptoMiniSat [27] with enabled GE.[3] Note that MapleLCMDistChronoBT and CaDiCaL are the winners in the main track of the latest SAT competition [22] in 2018. All tests were performed on a 2.40GHz Intel Xeon E5-2640 processor and are an average of 100 runs.

For SAT models derived from cryptographic problems, the preprocessing technique is executed only once, since all instances presenting a specific cryptographic problem are equivalent except for the constant in the XOR constraints. Even though the MVC problem is NP-complete, its execution for graphs derived from our models always finished in negligible running time, due to the small number of nodes.

We conducted experiments using both the third and the fourth polynomials. Results on solving the third summation polynomial ($m = 2$) are shown in Table 4. The parameters used to obtain these benchmarks are $n = 41$ and $l = 20$. As a result, we obtained a Boolean polynomial system of 41 equations in 40 variables (see Section 2). We show running-time averages on satisfiable and unsatisfiable instances separately, as these values differ between the two cases.

As different variants of our solver can yield better results for different benchmarks, we compared all variants to decide on the optimal one. We also tested the solver with and without our preprocessing technique (denoted by mvc in the tables). The results in Table 3 show that WDSat yields optimal results for these benchmarks when the XG-ext method is used coupled with the preprocessing technique. This outcome is not surprising when we examine the MVC obtained by the preprocessing technique. The number of variables in the system is $k = 40$, but the number of vertices in the MVC is 20. This means that by using the optimization techniques described in this paper, the worst-case time complexity of the examined models drops from 2^k to $2^{\frac{k}{2}}$. This is the case for every instance derived from the third summation polynomial.

Table 3. Comparing different versions of WDSat for solving the third summation polynomial.

WDSat+	SAT		UNSAT	
	Runtime (s)	#Conflicts	Runtime (s)	#Conflicts
XG	6028.4	200957178	11743.2	354094821
XG+mvc	639.6	21865963	2973.0	94489361
XG-ext	375.9	4911099	870.1	10789518
XG-ext+mvc	**4.2**	**27684**	**13.5**	**86152**

By analyzing the average running time and the average number of conflicts in Table 4, we see that the chosen variant of the WDSat solver outperforms all other approaches for solving instances derived from the third summation polynomial.

[3] Enabling GE in CryptoMiniSat yielded better performance for these benchmarks.

Current versions of CryptoMiniSat do not allow choosing the order of the branching variables as its authors claim that this technique almost always results in slower running times. To verify this claim, we modified the source code of CryptoMiniSat in order to test our preprocessing technique coupled with this solver (see line CryptoMiniSat+mvc in Table 4). We set a timeout of 10 minutes and only 9 out of 100 unsatisfiable and 54 out of 100 satisfiable instances were solved. This confirms that the MVC preprocessing technique is strongly linked to our XG-ext method. Indeed, when the XG-ext method is not used, one can not guarantee that when all variables from the MVC are assigned the system becomes linear. This is confirmed also by looking at the number of conflicts for the CryptoMiniSat+mvc approach, which is greater than $2^{\frac{k}{2}}$ even for benchmarks that were solved before the timeout. Recall that $\frac{k}{2}$ is the size of the MVC. On the other hand CryptoMiniSat without the preprocessing technique succeeds in solving these instances after less than $2^{\frac{k}{2}}$ conflicts. We conclude that the searching technique in CryptoMiniSat used to decide on the next branching variable is optimal for this solver.

The solvers which are not XOR-enabled did not solve any of the 200 satisfiable and unsatisfiable instances before the 10-minute timeout. This is not surprising as instances derived from the third summation polynomial are solved a lot faster when a GE technique is used.

Table 4. Comparing different approaches for solving the third summation polynomial.

Solving approach	SAT		UNSAT	
	Runtime (s)	#Conflicts	Runtime (s)	#Conflicts
Gröbner	16.8	N/A	18.7	N/A
MiniSat	> 600		> 600	
Glucose	> 600		> 600	
MapleLCMDistChronoBT	> 600		> 600	
CaDiCaL	> 600		> 600	
CryptoMiniSat	29.0	226668	84.3	627539
CryptoMiniSat+mvc	237.4	1263601	> 600	
WDSat+XG-ext+mvc	**4.2**	**27684**	**13.5**	**86152**

Experimental results in Table 5 are performed using benchmarks derived from the fourth summation polynomial. We obtain our model using a symmetrization technique proposed by Gaudry [15]. According to our parameter choice, the initial polynomial system contains 52 equations in 51 variables. However, only 18 out of the 51 variables are 'crucial'. The other 33 variables are introduced as a result of Gaudry's symmetrization technique. Our experiments show that performing GE on these instances does not result in faster running times. On the contrary, running times are significantly slower when the XG module of the WDSat solver is enabled. Running times become even slower with the XG-ext method. We attribute this fallout to the particularly small improvement in the

number of conflicts, compared to the significant computational cost of performing the GE technique. Indeed, the graph corresponding to the model for the fourth summation polynomial is complete and thus the size of the MVC is equivalent to the number of variables in the formula. This leads us to believe there is no optimal choice for the order of branching variables and the system generally does not become linear until the second-to-last branching. We conclude that for solving these instances WDSat without GE is the optimal variant, since it outperforms both the Gröbner basis method and current state-of-the-art solvers.

To sum up, when WDSat is used for the index calculus attack, our recommendation is to enable the XG-ext option for instances obtained from the third summation polynomial and to completely disable the XG module for instances from the fourth polynomial. For ANF instances arising from other cryptographic problems, it would be best to solve smaller instances of the problem and analyse the number of conflicts. If the number of conflicts is only slightly better when the XG module is enabled, then disabling the XG module is likely to yield faster running times for higher scale instances of that problem.

Table 5. Comparing different approaches for solving the fourth summation polynomial.

Solving approach	SAT		UNSAT	
	Runtime (s)	#Conflicts	Runtime (s)	#Conflicts
Gröbner	229.3	N/A	229.4	N/A
MiniSat	239.7	1840190	517.0	3433304
Glucose	189.2	1527158	274.8	2056575
MapleLCMDistChronoBT	655.1	4035131	918.7	5378945
CaDiCaL	43.6	254194	141.3	629869
CryptoMiniSat	331.8	1791188	707.9	3416526
WDSat	**0.6**	**48438**	**3.8**	**255698**
WDSat+XG	19.0	85282	49.8	252949

Our solver is dedicated to problems arising from a Weil descent. However, we tested it on Trivium [9] instances as they are extensively used in the SAT literature. We created instances using a modelization similar to the one in Grain of Salt [26], a tool for deriving instances for keystream generators comprised of Nonlinear-Feedback Shift Registers (NLFSR). Our experience is that CryptoMiniSat yields faster running times than all of the WDSat variants for Trivium instances. WDSat does not implement any of the optimizations for Trivium such as dependent variable removal, sub-problem detection, etc. as there are no such occurrences in systems arising from a Weil descent.

7 Conclusion

In this paper, we revisited XOR-enabled SAT solvers and their use in cryptanalysis. We proposed a novel SAT solver, named WDSat, dedicated to solv-

ing instances derived from the index calculus attack on binary elliptic curves. We conducted experiments comparing WDSat to the algebraic Gröbner basis resolution method, as well as to five state-of-the-art SAT solvers. Our solver outperforms all existing resolution approaches for this specific problem.

References

1. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 399–404 (2009), http://ijcai.org/Proceedings/09/Papers/074.pdf
2. Bettale, L., Faugère, J., Perret, L.: Hybrid approach for solving multivariate systems over finite fields. J. Mathematical Cryptology **3**(3), 177–197 (2009). https://doi.org/10.1515/JMC.2009.009
3. Biere, A.: CaDiCaL Simplified Satisfiability Solver. http://fmv.jku.at/cadical/, accessed: 2020-05-27
4. BlueKrypt: Cryptographic key length recommendation. https://www.keylength.com (2018), accessed: 2020-05-27
5. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. Journal of Symbolic Computation **24**(3-4), 235–265 (1997). https://doi.org/10.1006/jsco.1996.0125
6. Bouillaguet, C., Chen, H.C., Cheng, C.M., Chou, T., Niederhagen, R., Shamir, A., Yang, B.Y.: Fast Exhaustive Search for Polynomial Systems in F_2. In: Proceedings of the 12th International Conference on Cryptographic Hardware and Embedded Systems. p. 203–218. CHES'10, Springer-Verlag, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_14
7. Choo, D., Soos, M., Chai, K.M.A., Meel, K.S.: Bosphorus: Bridging ANF and CNF solvers. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019. pp. 468–473 (2019). https://doi.org/10.23919/DATE.2019.8715061
8. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (Jul 1962). https://doi.org/10.1145/368273.368557
9. De Cannière, C.: Trivium: A stream cipher construction inspired by block cipher design principles. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) Information Security. pp. 171–186. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
10. Diem, C.: On the discrete logarithm problem in elliptic curves. Compositio Mathematica **147**(1), 75–104 (2011). https://doi.org/10.1112/S0010437X10005075
11. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Theory and Applications of Satisfiability Testing. pp. 502–518. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
12. Faugère, J.C.: A New Efficient Algorithm for Computing Gröbner basis (F4). Journal of Pure and Applied Algebra **139**(1-3), 61–88 (1999). https://doi.org/10.1145/780506.780516
13. Faugère, J., Perret, L., Petit, C., Renault, G.: Improving the Complexity of Index Calculus Algorithms in Elliptic Curves over Binary Fields. In: Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings. pp. 27–44 (2012). https://doi.org/10.1007/978-3-642-29011-4_4

14. Galbraith, S.D., Gebregiyorgis, S.W.: Summation polynomial algorithms for elliptic curves in characteristic two. In: Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India. Lecture Notes in Computer Science, vol. 8885, pp. 409–427. Springer (2014). https://doi.org/10.1007/978-3-319-13039-2_24
15. Gaudry, P.: Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. J. Symb. Comput. **44**(12), 1690–1702 (2009). https://doi.org/10.1016/j.jsc.2008.08.005
16. Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Revisiting AES related-key differential attacks with constraint programming. Inf. Process. Lett. **139**, 24–29 (2018). https://doi.org/10.1016/j.ipl.2018.07.001
17. Gerault, D., Lafourcade, P., Minier, M., Solnon, C.: Computing AES related-key differential characteristics with constraint programming. Artif. Intell. **278** (2020). https://doi.org/10.1016/j.artint.2019.103183
18. Gerault, D., Minier, M., Solnon, C.: Using constraint programming to solve a cryptanalytic problem. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 4844–4848. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/679
19. Lidl, R., Niederreiter, H.: Introduction to Finite Fields and Their Applications. Cambridge University Press, USA (1986)
20. Liu, F., Cruz, W., Ma, C., Johnson, G., Michel, L.: A tolerant algebraic side-channel attack on aes using cp. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming. pp. 189–205. Springer International Publishing, Cham (2017)
21. Liu, F., Cruz, W., Michel, L.: A complete tolerant algebraic side-channel attack for aes with cp. In: Hooker, J. (ed.) Principles and Practice of Constraint Programming. pp. 259–275. Springer International Publishing, Cham (2018)
22. van Maaren, H., Franco, J.: The International SAT Competition Web Page. http://www.satcompetition.org/, accessed: 2020-05-27
23. Nadel, A., Ryvchin, V.: Chronological backtracking. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10929, pp. 111–121. Springer (2018). https://doi.org/10.1007/978-3-319-94144-8_7
24. Petit, C., Quisquater, J.: On Polynomial Systems Arising from a Weil Descent. In: Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security. Lecture Notes in Computer Science, vol. 7658, pp. 451–466. Springer (2012). https://doi.org/10.1007/978-3-642-34961-4_28
25. Semaev, I.A.: Summation polynomials and the discrete logarithm problem on elliptic curves. IACR Cryptology ePrint Archive **2004**, 31 (2004), http://eprint.iacr.org/2004/031
26. Soos, M.: Grain of Salt — an Automated Way to Test Stream Ciphers through SAT Solvers. In: Tools'10: the Workshop on Tools for Cryptanalysis 2010. pp. 131–144. London, United Kingdom (Jun 2010), https://hal.archives-ouvertes.fr/hal-01288922
27. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT Solvers to Cryptographic Problems. In: SAT. Lecture Notes in Computer Science, vol. 5584, pp. 244–257. Springer (2009). https://doi.org/10.1007/978-3-642-02777-2_24

28. Trimoska, M., Ionica, S., Dequen, G.: EC Index Calculus Benchmarks. https://github.com/mtrimoska/EC-Index-Calculus-Benchmarks (2020)
29. Trimoska, M., Ionica, S., Dequen, G.: WDSat Solver. https://https://github.com/mtrimoska/WDSat (2020)