# A SAT-based approach for index calculus on binary elliptic curves

Monika Trimoska and Sorina Ionica and Gilles Dequen

Laboratoire MIS, Université de Picardie Jules Verne
33 Rue Saint Leu Amiens 80039 - France

**Abstract.** Logical cryptanalysis, first introduced by Massacci in 2000, is a viable alternative to common algebraic cryptanalysis techniques over boolean fields. With XOR operations being at the core of many cryptographic problems, recent research in this area has focused on handling XOR clauses efficiently. In this paper, we investigate solving the point decomposition step of the index calculus method for prime degree extension fields $\mathbb{F}_{2^n}$, using SAT solving methods. We experimented with different SAT solvers and decided on using WDSAT, a solver dedicated to this specific problem. We extend this solver by adding a novel breaking symmetry technique and optimizing the time complexity of the point decomposition step by a factor of $m!$ for the $(m+1)^{\text{th}}$ Semaev's summation polynomial. While asymptotically solving the point decomposition problem with this method has exponential worst time complexity in the dimension $l$ of the vector space defining the factor base, experimental running times show that the the presented SAT solving technique is significantly faster than current algebraic methods based on Gröbner basis computation. For the values $l$ and $n$ considered in the experiments, the WDSAT solver coupled with our breaking symmetry technique is up to 300 times faster then MAGMA's F4 implementation, and this factor grows with $l$ and $n$.

**Keywords:** discrete logarithm, index calculus, elliptic curves, point decomposition, symmetry, satisfiability, DPLL algorithm

## 1 Introduction

The index calculus algorithm originally denoted a technique to compute discrete logarithms modulo a prime number, but it now refers to a whole family of algorithms adapted to other finite fields and to some algebraic curves. It includes the Number Field Sieve (NFS) [21], dedicated to logarithms in $\mathbb{Z}_q$ and the algorithms of Gaudry [14] and Diem [7] for algebraic curves in $\mathbb{F}_{q^n}$, where $q = p^k$. Index calculus algorithms proceed in two main steps. The *sieving* (or *point decomposition*) step concentrates most of the number theory and algebraic geometry needed overall. By splitting random elements over a well-chosen factor base, it produces a large sparse matrix, the rows of which are "relations". In a second phase, the *matrix step* produces "good" combinations of the relations by finding a non-trivial vector in the kernel of this matrix. This in turn enables the efficient computation of any discrete logarithm on the input domain. A crucial step of the index calculus on elliptic curves is to solve the *point decomposition problem* (PDP), by generating sufficiently many relations among suitable points on the curve. Using the so-called summation polynomials attached to the curve, this boils down to solving a system of polynomial equations whose solutions are the coordinates of points. The resulting algorithm has complexity $O(q^{2-2/n})$, but this hides an exponential factor in $n$ which comes from the hardness of solving the point decomposition problem.

Consequently, when $q$ is large, $n \geq 3$ is small and $\log q > cm$ for some constant $c$, the Gaudry-Diem algorithm has a better asymptotic complexity then generic methods for solving the discrete logarithm problem and Gröbner basis algorithms have become a well-established technique [17] to solve these systems. Since a large number of instances of PDP needs to be solved, most of the research in the area has focused on improving the complexity of this step. Several simplifications such as symmetries and polynomials with lower degree obtained from the algebraic structure of the curve have been proposed [9].

When we consider elliptic curves defined over $\mathbb{F}_{2^n}$ with $n$ prime, solving the PDP system via Gröbner basis quickly becomes a bottleneck, and index calculus algorithms are slower than generic attacks, from a theoretical and a practical point of view. Moreover, it is not known how to define the factor base in order to exploit all the symmetries coming from the algebraic structure of the curve, without increasing the number of variables when solving PDP [33]. Finally, note that for random systems, pure Gröbner basis algorithms are both theoretically and practically slower than simpler methods, typically exhaustive search [5,22], hybrid methods [2] and SAT solvers. It is thus natural that we turn our attention towards combinatorics tools to solve the PDP in characteristic 2.

Until recent years, SAT solvers have been proven to be a powerful tool in the cryptanalysis of symmetric schemes. They were successfully used for attacking secret key cryptosystems such as Bivium, Trivium, Grain, AES [15,20,16,28,27]. However, their use in public key cryptosystems has rarely been considered. A prominent example is the work of Galbraith and Gebregiyorgis [13], where they explore the possibility of replacing available Gröbner basis implementations with generic SAT solvers (such as MINISAT), as a tool for solving the polynomial system for the PDP over binary curves. They observe experimentally that the use of SAT solvers may potentially enable larger factor bases to be considered.

In this paper, we take important steps towards fully replacing Gröbner basis techniques for solving PDP with constraint programming ones. First, we model the point decomposition problem as a logical formula, with a reduced number of clauses, when compared to the model used in [13]. We compare different SAT solvers and decide that the recently introduced WDSAT solver [31] is most adapted to this problem and yields fastest running times. Secondly, we propose a breaking symmetry technique and we implement it as an extension of this solver. We show that by using the extended solver, the proven worst case complexity of solving a PDP is $O(\frac{2^{ml}}{m!})$, where $m$ is the number of points in the decomposition and $l$ is the dimension of the vector space defining the factor base. This is to be compared against the Gröbner basis algorithm proposed in [10], whose runtime $O(2^{\omega n/2})$ (with $n \sim ml$ and $\omega$ the linear algebra constant) is proven under heuristic assumptions.

We experimented with the index calculus attack on the discrete logarithm for elliptic curves over prime degree binary extension fields. We obtain an important speedup in comparison with the best currently available implementation of Gröbner basis (F4 [10] in MAGMA [4]) and generic solvers [29,1,28]). Consequently, we were able to display results for a range of parameters $l$ and $n$ that were not feasible with previous approaches. In addition, our experiments show that Gröbner basis cannot compete with SAT solvers techniques in terms of memory requirements. To illustrate, a system, which is solved with the extended WDSAT solver using only 17MB of memory, requires more than 200GB when using the Gröbner basis method.

Our experiments suggest that this improved PDP resolution does not render the index calculus attack faster than generic methods for solving the ECDLP in the case of prime degree extension fields $\mathbb{F}_{2^n}$.

This paper is organized as follows. Section 2 gives an overview of the index calculus algorithm on elliptic curves, introduces the PDP problem and briefly recalls algebraic and combinatorial techniques used in the literature to solve this problem. Section 3 details the reasoning models used in our experiments. Section 4 explains the breaking symmetry technique that we implement in a SAT solver. In Section 5 we give worst time complexity estimates for solving a PDP instance and derive the complexity of our SAT-based index calculus algorithm. Finally, Section 6 presents benchmarks obtained with our implementation. We compare this against results obtained using MAGMA's F4 implementation and several available best generic SAT-solvers, such as MINISAT [29] and CRYPTOMINISAT [28].

## 2 An Overview of Index Calculus

In 2008 and 2009, Gaudry [14] and Diem [7] independently proposed a technique to perform the point decomposition step of the index calculus attack for elliptic curves over extension fields, using Semaev's summation polynomials [24]. Since this paper focuses on binary elliptic curves, we introduce Semaev's summation polynomials here directly for these curves.

Let $\mathbb{F}_{2^n}$ be a finite field and $E$ be an elliptic curve defined by the equation

$$E : y^2 + xy = x^3 + ax^2 + b, \tag{1}$$

with $a, b \in \mathbb{F}_{2^n}$. Using standard notation, we take $\bar{\mathbb{F}}_{2^n}$ to be the algebraic closure of $\mathbb{F}_{2^n}$ and $E(\mathbb{F}_{2^n})$ (resp. $E(\bar{\mathbb{F}}_{2^n})$) to be the set of points on the elliptic curve defined over $\mathbb{F}_{2^n}$ (resp. $\bar{\mathbb{F}}_{2^n}$). Let $\mathcal{O}$ to be the point at infinity on the elliptic curve. For $m \in \mathbb{N}$, the $m^{th}$-summation polynomial is a multivariate polynomial in $\mathbb{F}_{2^n}[X_1, \ldots, X_m]$ with the property that, given points $P_1, \ldots, P_m \in E(\bar{\mathbb{F}}_{2^n})$, then $P_1 + \ldots + P_m = \mathcal{O}$ if and only if $S_m(\mathbf{x}_{P_1}, \ldots, \mathbf{x}_{P_m}) = 0$. We have that

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

and for $m \geq 4$ we have the following recursive formula:

$$S_m(X_1, \ldots, X_m) = \qquad (3)$$
$$Res_X(S_{m-k}(X_1, \ldots, X_{m-k-1}, X), S_{k+2}(X_{m-k}, \ldots, X_m, X)).$$

The polynomial $S_m$ is symmetric and has degree $2^{m-2}$ in each of the variables. Let $V$ be a vector subspace of $\mathbb{F}_{2^n}/\mathbb{F}_2$, whose dimension $l$ will be defined later. We define the factor basis $\mathcal{B}$ to be :

$$\mathcal{B} = \{(\mathbf{x}, \mathbf{y}) \in E(\mathbb{F}_{2^n}) | \mathbf{x} \in V\}.$$

Heuristically, we can easily see that the factor base has approximatively $2^l$ elements. Given a point $R \in E(\mathbb{F}_{2^n})$, the point decomposition problem is to find $m$ points $P_1, \ldots, P_m \in \mathcal{B}$ such that $R = P_1 + \ldots + P_m$. Using Semaev's polynomials, this problem is reduced to the one of solving a multivariate polynomial system.

**Definition 1.** *Given $s \geq 1$ and an $l$-dimensional vector subspace $V$ of $\mathbb{F}_{2^n}/\mathbb{F}_2$ and $f \in \mathbb{F}_{2^n}[X_1, \ldots, X_m]$ any multivariate polynomial of degree bounded by $s$, find $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) \in V^m$ such that $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) = 0$.*

Using the fact that $\mathbb{F}_{2^n}$ is an $n$-dimensional vector space over $\mathbb{F}_2$, the equation $f(\mathbf{x}_1, \ldots, \mathbf{x}_m) = 0$ can be rewritten as a system of $n$ equations over $\mathbb{F}_2$, with $ml$ variables. In the literature, this is called a *Weil restriction* [14] or *Weil descent* [23]. The probability of having a solution to this system depends on the ratio between $n$ and $l$. Roughly, when $n/l \sim m$ the system has a reasonable chance to have a solution.

Recent work on solving the decomposition problem has focused on using advanced methods for Gröbner basis computation such as Faugère's $F_4$ and $F_5$ algorithms [10,11]. This is a natural approach, given that similar techniques for small degree extension fields in characteristic $> 2$ yielded index calculus algorithms which are faster than the generic attacks on the DLP.

A common technique when working with Semaev's polynomials is to use a symmetrization process in order to further reduce the degree of the polynomials appearing in the PDP system. In short, since $S_m$ is symmetric, we can rewrite it in terms of the elementary symmetric polynomials $e_1 = \sum_{1 \leq i_1 \leq m} X_{i_1}$, $e_2 = \sum_{1 \leq i_1, i_2 \leq m} X_{i_1} X_{i_2}$, ..., $e_m = \prod_{1 \leq i \leq m} X_i$. We denote $S'_{m+1}$ the polynomial obtained after symmetrizing $S_{m+1}$ in the first $m$ variables, i.e. we have $S'_{m+1} \in \mathbb{F}_{2^n}[e_1, \ldots, e_m, X_{m+1}]$.

In [33], the authors report on experiments lead on systems obtained using a careful choice of the vector space $V$ and application of the symmetrization process. Using Magma's $F_4$ available implementation, we experimented with both the symmetric and the non-symmetric version for PDP systems and found, as in [33], that the symmetric version yields better results. Therefore, in order to set the notation, we detail this approach here.

Let $t$ be a root of a defining polynomial of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. Following [33], we choose the vector space $V$ to be the dimension-$l$ subspace generated by $1, t, t^2, \ldots, t^{l-1}$. Therefore we can write:

$$\begin{aligned}
e_1 &= d_{1,0} + \ldots + d_{1,l-1}t^{l-1} \\
e_2 &= d_{2,0} + \ldots + d_{2,2l-2}t^{2l-2} \\
&\ldots \\
e_m &= d_{m,0} + \ldots + d_{m,m(l-1)}t^{m(l-1)}
\end{aligned} \tag{4}$$

where the $d_{i,j}$ with $1 \leq i \leq m$, $0 \leq j \leq i(l-1)$ are binary variables. After choosing $\mathbf{x}_{m+1} \in \mathbb{F}_{2^n}$ and substituting $e_1, \ldots, e_m$ as in Equation (4), we get:

$$S'_{m+1}(e_1, \ldots, e_m, \mathbf{x}_{m+1}) = f_0 + \ldots + f_{n-1}t^{n-1},$$

where $f_i$, $0 \leq i \leq n-1$ are polynomials in the binary variables $d_{i,j}$, $1 \leq i \leq m$, $0 \leq j \leq i(l-1)$ . After a Weil descent, we obtain the following polynomial system

$$f_0 = f_1 = \ldots = f_{n-1} = 0. \tag{5}$$

One can see that with this approach, the number of variables is increased by a factor $m$, but the degrees of the polynomials in the system are seriously reduced. Further simplification of this system can be obtained if the elliptic curve has a rational point of order 2 or 4 [13]. Since this is a restriction, we did not implement this approach and used the system in Equation (5) as the starting point for our SAT model of the point decomposition problem.

### 2.1 Solving the decomposition problem using SAT solvers

Before presenting our approach for finding solutions of the PDP using SAT solvers, we give preliminaries on the Satisfiability problem, its terminology and solving techniques. A SAT solver is a special purpose program to solve the SAT problem. Using SAT solvers as a cryptanalytic tool requires expressing the cryptographic problem as a Boolean formula in conjunctive normal form (CNF). The basic building block of a CNF formula is a *literal*, which is either a propositional variable or its negation. An OR-*clause* is a non-exclusive disjunction ($\vee$) of literals $x_1 \vee x_2 \vee \ldots \vee x_k$. A CNF formula is a unique OR-clause or a conjunction ($\wedge$) of at least two OR-clauses. An *interpretation* of a given propositional formula consists in assigning a truth value (TRUE /FALSE) to each of its variables. A CNF formula is said to be *satisfiable* if there exists at least one interpretation under which the formula is TRUE, and it is said to be *unsatisfiable* otherwise. The propositional satisfiability problem (SAT) is the problem of determining whether a (usually CNF) formula is satisfiable.

In the remainder of this paper, we will refer to an OR-clause simply by a clause, since CNF is the standard form used in SAT solvers. A clause where the operation between literals is an exclusive OR, will be referred to as a XOR-clause. The use of the logical XOR operator ($\oplus$) is common in cryptography. When working on cryptographic problems the CNF form can be extended to a CNF-XOR form, which is a conjunction of both OR-clauses and XOR-clauses.

The most straightforward method for solving the SAT problem is to complete the truth table associated to the formula in question. This is equivalent to an exhaustive search method and thus impractical. Luckily, in some cases a *partial* assignment on the set of variables can determine whether a clause is satisfiable. Assigning $l$, a literal from the partial assignment, to TRUE will lead to :

1. Every clause containing $l$ is removed (since the clause is satisfied).
2. In every clause that contains $\neg l$ this literal is deleted (since it can not contribute to the clause being satisfied).

The second rule above can lead to obtaining a clause composed of a single literal, called a *unit* clause. Since this is the only literal left which can satisfy the clause, it must be set to TRUE and therefore *propagated*. The described method is called *unit propagation*. The reader can refer to [3] for more details.

A *conflict* occurs when it exists at least one clause with all literals assigned to FALSE in the formula. If this case is a consequence of a direct assignment, or eventually of Unit Propagation, this has to be undone. This is commonly known as *backtracking*.

*Example 1.* For instance, these two atomic operations can be illustrated thanks to the following sample built on a set of 5 clauses numbered $C_1$ to $C_5$.

$$\begin{aligned} &C_1 : \neg x_1 \vee x_2 \vee \neg x_4 \\ &C_2 : x_1 \vee x_3 \vee x_4 \\ &C_3 : x_1 \vee \neg x_3 \\ &C_4 : x_1 \vee x_3 \\ &C_5 : x_2 \vee x_4 \end{aligned}$$

Assigning the variable $x_1$ to FALSE leads the clause $C_1$ to be satisfied by the literal $x_1$. As well and as a consequence, clauses $C_2$, $C_3$ and $C_4$ cannot be satisfied thanks to the literal $x_1$. Hence, $x_1$ can be deleted from these clauses. Then, $C_3$ is a unit clause composed of the literal $\neg x_3$ and as a consequence, $x_3$ has to be assigned to FALSE. We say that the truth value of $x_3$ is inferred through unit propagation.

When we set $x_3$ to its inferred value FALSE, we apply the second rule to clauses $C_2$ and $C_4$. As a consequence, clause $C_4$ can not be satisfied by any of its literals. This constitutes a conflict and it invokes a backtracking procedure. The backtracking procedure consists in going back to the state that the formula was in, before the last assumption was made. In our example, the last assumption was that $x_1$ is FALSE and thus, we go back to the initial state.

The basic backtracking search with unit propagation that we described composes the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [6], which is a state-of-the-art complete SAT solving technique. DPLL works by trying to assign a truth value to each variable in the CNF formula, recursively building a binary search tree of height equivalent (at worst) to the number of variables. After each variable assignment, the formula is simplified by unit propagation. If a *conflict* is met, a backtracking procedure is launched and the opposite truth value is assigned to the last assigned literal. If the opposite truth value results in conflict as well, we backtrack to an earlier assumption or conclude that the formula is *unsatisfiable* - when there are no earlier assumptions left. The number of conflicts is a good measure for the time complexity of a SAT problem solved using a DPLL-based solver. If the complete search tree is built, the worst case complexity is $O(2^v)$, where $v$ is the number of variables in the formula. Figure 1 illustrates the binary search tree resulting from the resolution of Example 1.

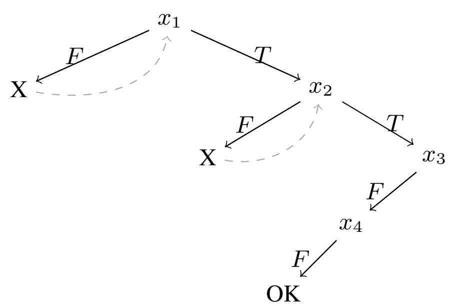


Fig. 1: Binary search tree constructed with the DPLL algorithm.

A common variation of the DPLL is the conflict-driven clause learning (CDCL) algorithm [26]. In this variation, each encountered conflict is described as a new clause and added to the formula (*learning*). State-of-the-art CDCL solvers, such as MINISAT and GLUCOSE, have been shown to be a powerful tool for solving CNF formulas. However, they are not equipped to handle XOR-clauses and thus parity constraints have to be translated into CNF. Since handling CNF-clauses derived from XOR constraints is not necessarily efficient, recent works have concentrated on coupling CDCL solvers with a XOR-reasoning module. Furthermore, these techniques can be enhanced by Gaussian elimination, as in the works of Soos *et al*. (resulting in the CRYPTOMINISAT solver) [28,27], Han and Jiang [16], Laitinen *et al*.[20,19].

## 3 Model description

This section gives in full detail the three models we used in our experiments: the algebraic one used by Yun-Ju *et al* [33], the CNF model used by Galbraith and Gebregiyorgis [13] and the model we propose.

### 3.1 The algebraic model

Since the logical models are constructed starting from the algebraic one, we present first the model used when solving the PDP problem using Gröbner basis. The elementary symmetric polynomials $e_i$ are written in terms of the $d_{i,j}$ binary variables, as in Equation (4). Similarly, since we look for a set of solutions $(\mathbf{x}_1, \ldots, \mathbf{x}_m) \in V^m$, the $X_i$

variables are written formally as follows:

$$
\begin{aligned}
X_1 &= c_{1,0} + \ldots + c_{1,l-1}t^{l-1} \\
X_2 &= c_{2,0} + \ldots + c_{2,l-1}t^{l-1} \\
&\ldots \\
X_m &= c_{m,0} + \ldots + c_{m,l-1}t^{l-1}
\end{aligned}
$$

where $c_{i,j}$, with $1 \leq i \leq m$, $0 \leq j \leq l-1$, are binary variables. Using Equation (**??**), we derive the following equations:

$$
\begin{aligned}
d_{1,0} &= c_{1,0} + \ldots + c_{m,0} \\
d_{1,1} &= c_{1,1} + \ldots + c_{m,1} \\
&\ldots \\
d_{m,m(l-1)} &= c_{1,l} \cdot \ldots \cdot c_{m,l}.
\end{aligned} \tag{6}
$$

The remaining equations correspond to polynomials $f_i$, $0 \leq i \leq n-1$, obtained in Equation (4) via the Weil descent on $S'_{m+1}$. Recall that these are polynomials in the binary variables $d_{i,j}$. We now describe how we derive logical formulas from this system.

### 3.2 The CNF-XOR model

When creating constraints from a boolean polynomial system, the multiplication of variables becomes a conjunction of literals and the sum of multiple terms becomes a XOR-clause. From the two sets of equations in the algebraic model, we obtain two sets of XOR-clauses, where the terms are single literals or conjunctions. To illustrate, the logical formula derived from Equation (6) is as follows:

$$
\begin{aligned}
&\neg d_{1,0} \oplus c_{1,0} \oplus \ldots \oplus c_{m,0} \\
&\neg d_{1,1} \oplus c_{1,1} \oplus \ldots \oplus c_{m,1} \\
&\ldots \\
&\neg d_{m,m(l-1)} \oplus (c_{1,l} \wedge \ldots \wedge c_{m,l}).
\end{aligned} \tag{7}
$$

SAT solvers adapted for XOR reasoning in the literature contain XOR clauses formed by xoring single literals, and not conjunctions of several ones. To follow this paradigm, we have to transform the system above further. We substitute all conjunctions in a XOR clause by a newly added variable. Let $c'$ be the variable substituting a conjunction $(c_{i_1,j_1} \wedge c_{i_2,j_2} \wedge ... \wedge c_{i_k,j_k})$. We have $c' \Leftrightarrow (c_{i_1,j_1} \wedge c_{i_2,j_2} \wedge ... \wedge c_{i_k,j_k})$, which rewrites as

$$
\begin{aligned}
&(c' \vee \neg c_{i_1,j_1} \vee \neg c_{i_2,j_2} \vee ... \vee \neg c_{i_k,j_k}) \wedge \\
&(\neg c' \vee c_{i_1,j_1}) \wedge \\
&(\neg c' \vee c_{i_2,j_2}) \wedge \\
&\ldots \\
&(\neg c' \vee c_{i_k,j_k})
\end{aligned} \tag{8}
$$

For clarity, variables introduced by substitution of monomials containing exclusively the variables $c_{i,j}$ will be denoted $c'$ and clauses derived from these substitutions are said to be in the $X$-substitutions set of clauses. Similarly, substitutions of the monomials containing only the $d_{i,j}$ variables are denoted by $d'$ and the resulting set is referred to as the $E$-substitutions set of clauses.

Note from Equation (8) that the number of clauses obtained by substitution of a $k$-degree monomial is $k+1$. This will be further discussed in our complexity analysis.

After substituting conjunctions, we will refer to the set of clauses obtained from Equation (7) as the $E$-$X$-relation set of clauses. Finally, the equations corresponding to polynomials $f_i$, $0 \leq i \leq n-1$, are derived in the same manner and the resulting clauses will be referred to as the $F$ set of clauses.

That concludes the four sets of clauses in our SAT model. This model does not represent a CNF formula, since the $E$-$X$-relation set and the $F$ set are made up of XOR-clauses. Hence, it will be referred to as the CNF-XOR model.

**Proposition 1.** *Assigning all $c_{i,j}$ variables, for $1 \leq i \leq m$ and $1 \leq j \leq l$, leads to the assignment of all variables in the* CNF-XOR *model through unit propagation.*

*Proof.* Let us examine the unit propagation process for each set of clauses separately.

1. Clauses in the $X$-substitutions set are obtained by transforming $c' \Leftrightarrow (c_{i_1,j_1} \wedge c_{i_2,j_2} \wedge ... \wedge c_{i_k,j_k})$. We note that on the right of these equivalences there are only $c_{i,j}$ variables and on the left there is one single $c'$ variable. The assignment of all of the $c_{i,j}$ variables will yield the assignment of all variables on the left of the equivalences, i.e. all $c'$ variables.
2. Clauses in the $E$-$X$-relations set are obtained by transforming the algebraic system in (6). We observe that on the right of the equations there are only $c_{i,j}$ and $c'$ variables and on the left there is one single $d_{i,j}$ variable. When all $c_{i,j}$ and all $c'$ variables are assigned, all $d_{i,j}$ variables will have their truth value assigned through unit propagation on the $E$-$X$-relation set.
3. Clauses in the $E$-substitutions set are obtained by transforming $d' \Leftrightarrow (d_{i_1,j_1} \wedge d_{i_2,j_2} \wedge ... \wedge d_{i_k,j_k})$. Similarly as with the $X$-substitutions set, we have only $d_{i,j}$ variables on the right of these equivalences and one single $d'$ variable on the left. The assignment of all of the $d_{i,j}$ variables will thus yield the assignment of all $d'$ variables.
4. Finally, parity constraints in set $F$ decide whether the obtained interpretation satisfies the formula.

This concludes the four types of variables present in the CNF-XOR model. □

### 3.3 The CNF model

Since most of the modern SAT solvers can read and process CNF formulas, we explain the classical technique for transforming a CNF-XOR model to a CNF model. In fact, this is also the technique used in MAGMA's available implementation for deriving a CNF model from a boolean polynomial system.

A XOR-clause is said to be satisfied when it evaluates to TRUE, i.e. when there are an odd number of literals set to TRUE. The CNF-encoding of a ternary XOR-clause $(x_1 \oplus x_2 \oplus x_3)$ is

$$\begin{aligned} (x_1 \vee \neg x_2 \vee \neg x_3) \wedge \\ (\neg x_1 \vee x_2 \vee \neg x_3) \wedge \\ (\neg x_1 \vee \neg x_2 \vee x_3) \wedge \\ (x_1 \vee x_2 \vee x_3) \end{aligned} \tag{9}$$

Similarly, a XOR-clause of size $k$ can be transformed to a conjunction of $2^{k-1}$ OR-clauses of size $k$. Since the number of introduced clauses grows exponentially with the size of the XOR-clause, it is a good practice to cut up the XOR-clause into manageable size clauses before proceeding with the transformation. To cut a XOR-clause $(x_1 \oplus \ldots \oplus x_k)$ of size $k$ in two, we introduce a new variable $\boldsymbol{x'}$ and we obtain the following two XOR-clauses:

$$\begin{aligned} &(x_1 \oplus \ldots \oplus x_i \oplus \boldsymbol{x'}) \wedge \\ &(x_{i+1} \oplus \ldots \oplus x_k \oplus \neg \boldsymbol{x'}). \end{aligned}$$

In our experiments with MINISAT in Section 6, we used a CNF model obtained after cutting into ternary XOR-clauses, since any XORSAT problem reduces in polynomial time to a 3-XORSAT problem [3]. To the best of our knowledge, MAGMA's implementation adopts a size 5 for XOR clauses. The optimal size at which to cut the XOR-clauses depends on the nature of the model and can be determined by running experiments using different values. Running these experiments was out of the scope of our work, as the WDSAT solver does not use the CNF model.

We implemented all three models described in this section and we present Table 1 to serve as a comparison on the number of variables, equations and clauses. Values for the algebraic and CNF-XOR model are exact, whereas those for the CNF model are averages obtained from experiments presented in Section 6.

| | | Gröbner model | | CNF model | | CNF-XOR model | | |
|---|---|---|---|---|---|---|---|---|
| l | n | #Vars | #Equations | #Vars | #CNF-clauses | #Vars | #CNF-clauses | #XOR-clauses |
| 6 | 19 | 51 | 52 | 5019 | 19577 | 767 | 2364 | 52 |
| 7 | 23 | 60 | 62 | 8223 | 32201 | 1101 | 3466 | 62 |
| 8 | 23 | 69 | 68 | 11036 | 43210 | 1510 | 4835 | 68 |
| 9 | 37 | 78 | 88 | 20969 | 82721 | 2000 | 6495 | 88 |
| 10 | 47 | 87 | 104 | 32866 | 130040 | 2577 | 8470 | 104 |
| 11 | 59 | 96 | 122 | 49538 | 196434 | 3247 | 10784 | 122 |

Table 1: The number of variables and equations/clauses for the three models.

In 2014, Galbraith and Gebregiyorgis [13] used MAGMA's implementation to compute the equivalent CNF logical formulas of the polynomial system resulting from the

Weil descent of a PDP system and ran experiments using the general-purpose MINISAT solver to get solutions for these formulas. One can see from Table 1 that the model they used has a significantly larger number of clauses and variables, when compared to the CNF-XOR model. This motivated our choice of the CNF-XOR model for this work.

# 4 Breaking symmetry

Since Semaev's summation polynomials are symmetric, if $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ is a solution, then all permutations of this set are solutions as well. These solutions are equivalent and finding more than one is of no use for the PDP. When a DPLL-based SAT solver is used (see Section 2.1), we observe redundancy in the binary search tree. Indeed, for $m = 3$ when a potential solution $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ has been eliminated, $\{\mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_3\}$ does not need to be tried out. To avoid this redundancy, we establish the following constraint $\mathbf{x}_1 \leq \mathbf{x}_2 \leq \ldots \leq \mathbf{x}_m$.

It would be tedious to add this constraint in the model itself. Any approach implies adding clauses and weighing the SAT model. Instead, we decided to add this constraint in the DPLL algorithm using a tree-pruning-like technique. In a classical DPLL implementation we try out both FALSE and TRUE for the truth value of a chosen variable. In our breaking symmetry variation of DPLL, in some cases the truth value of FALSE will not be tried out as all potential solutions after this assignment would not satisfy the constraint $\mathbf{x}_1 \leq \mathbf{x}_2 \leq \ldots \leq \mathbf{x}_m$. Our variation of DPLL is detailed in Algorithm 1 and the line numbers that distinguish it from a classical DPLL algorithm are in bold. Note that one crucial difference between the two algorithms is the choice of a variable on line 4. While this choice is arbitrary in a classical DPLL algorithm, in Algorithm 1 variables need to be chosen in the order from the leading bit of $\mathbf{x}_1$ to the trailing bit of $\mathbf{x}_m$. If this is not respected, our algorithm does not yield a correct answer.

Using the notation from Section 3, $c_{i,j}$ corresponds to the $j^{\text{th}}$ bit of the $i^{\text{th}}$ $\mathbf{x}$-vector, where $2 \leq i \leq m$ and $1 \leq j \leq l$. We recall from Proposition 1 that assigning all $c_{i,j}$ variables in the CNF-XOR model leads to the assignment of all variables through unit propagation. In Algorithm 1, we decide whether to try out the truth value of FALSE for $c_{i,j}$ or not by comparing two $\mathbf{x}$-vectors bit for bit, in the same way that we would compare binary numbers. When we are deciding on the truth value of $c_{i,j}$ we have the following reasoning:

- If $c_{i-1,j}$ is FALSE, we try to set $c_{i,j}$ both to FALSE and TRUE (if FALSE fails). When $c_{i,j}$ is set to FALSE, all of the potential $\mathbf{x}_i$ solutions are greater than or equal to $\mathbf{x}_{i-1}$, thus we continue with the same bit comparison on the next level. However, when $c_{i,j}$ is set to TRUE, all of the potential $\mathbf{x}_i$ solutions are strictly greater than $\mathbf{x}_{i-1}$ and we no longer do bit comparison on further levels.
- If $c_{i-1,j}$ is TRUE, we only try out the truth value of FALSE and we continue to do bit comparison since the potential $\mathbf{x}_i$ solutions are greater than or equal to $\mathbf{x}_{i-1}$ at this point.

Lastly, we give further information which explain in full detail Algorithm 1. We use a flag denoted *compare* to instruct whether to do bit comparison at the current search tree level or not. On line 6 we reset the *compare* flag to TRUE since $c_{i,j}$, when $j = 0$,

**Algorithm 1** Function DPLL_BR_SYM($F$, $compare$) : Recursive function implementing the DPLL algorithm coupled with our breaking symmetry technique.

**Input**: Propositional formula $F$ and a flag $compare$
**Output**: TRUE if formula is satisfiable, FALSE otherwise.

1: **if** all clauses and all XOR-clauses are satisfied **then**
2:     **return** TRUE.
3: **end if**
**4:** choose next $c_{i,j}$.
**5:** **if** j=0 **then**
**6:**     $compare \leftarrow$ TRUE.
**7:** **end if**
**8:** **if** ($compare$ is FALSE) **or** $(i = 1)$ **or** ($c_{i-1,j}$ is set to FALSE) **then**
9:     $(contradiction, F') \leftarrow$ ASSIGN($F$, $\neg c_{i,j}$).
10:     **if** $contradiction$ **then**
11:         BACKTRACK().
**12:**         $compare \leftarrow$ FALSE.
13:     **else**
14:         **if** DPLL_BR_SYM($F'$, $compare$) returns FALSE **then**
15:             BACKTRACK().
**16:**             $compare \leftarrow$ FALSE.
17:         **else**
18:             **return** TRUE.
19:         **end if**
20:     **end if**
**21:** **end if**
22: $(contradiction, F') \leftarrow$ ASSIGN($F$, $c_{i,j}$).
23: **if** $contradiction$ **then**
24:     BACKTRACK().
25:     **return** FALSE.
26: **end if**
27: **return** DPLL_BR_SYM($F'$, $compare$).

corresponds to a leading bit of the next **x**-vector. Lastly, if-conditions on line 8 have to be checked in the specified order.

Algorithm 1 presents a depth-first transversal of a binary search tree with a symmetry breaking technique. We specifically designed it for the PDP, but it can be applied to similar problems that deal with symmetry.

## 5 Time complexity analysis

As we explained in Section 2, the time complexity of a SAT problem in a DPLL context is measured by the number of conflicts. This essentially corresponds to the number of leaves created in the binary search tree. The worst case complexity of the algorithm is thus $2^h$, where $h$ is the height of the tree.

As per Proposition 1, we only reason on $c_{i,j}$ variables from the CNF-XOR model. Therefore, $h = ml$ and the worst-case complexity for the PDP is $2^{ml}$.

Furthermore, with the symmetry breaking technique explained in Section 4, we optimize this complexity by a factor of $m!$. Indeed, out of the $m!$ permutations of the solution set $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, only one satisfies $\mathbf{x}_1 \leq \mathbf{x}_2 \leq \ldots \leq \mathbf{x}_m$ (neglecting the equality).

This concludes that the worst-case number of conflicts reached for one PDP computation is

$$\frac{2^{ml}}{m!}. \tag{10}$$

Going further in the time complexity analysis, we observe that to find one conflict we go through (in the worst case) all clauses in the model during unit propagation. Hence, the running time per conflict grows linearly with the number of clauses. First, let us count the number of clauses in the $X$-substitution set. For every $2 \leq d \leq m$ there exist $\binom{m}{d} \cdot l^d$ monomials of degree $d$ given by products of variables $c_{i,j}$, and they each yield $d+1$ clauses (see Equation (8)). In total, the number of clauses in the $X$-substitutions set is

$$(\sum_{d=2}^{m} \binom{m}{d} \cdot l^d)(d+1).$$

Recall that degree one monomials are not substituted and thus do not produce new clauses. We can adapt this reasoning for the $E$-substitutions set as well.

The number of XOR-clauses in the CNF-XOR model is equivalent to the number of equations in the algebraic model. We have $\frac{m(m+1)}{2}(l-1)+m$ in the $E$-$X$-relation set and $n$ in the $F$ set.

*Remark 1.* Using this analysis, we approximate the number of clauses, denoted by $C$, for $m = 3$, as all experiments presented in this paper are performed using the fourth summation polynomial.

$$\begin{aligned} C &\approx \binom{3}{2} \cdot 3l^2 + \binom{3}{3} \cdot 4l^3 + \left(\binom{3}{2}\right) \cdot 3(3l-2)^2 + (6l-3) + n \approx \\ &\approx 4l^3 + 171l^2 - 210l + n + 69. \end{aligned} \tag{11}$$

In practice, many monomials have no occurrence in the system after the Weil descent process. In fact, the value in (11) is a huge overestimate and exact values for $l \in \{6, \ldots, 11\}$ are shown in Table 1.

Assuming that we take $m$ small, we conclude that the number of clauses in our model is polynomial in $l$. Let $T$ be a constant representing the time to process one clause. The running time of the PDP is bounded by

$$T \cdot C \cdot 2^{ml}/m!.$$

This allows us to establish the following result on the complexity of our SAT-based index calculus algorithm.

**Theorem 1.** *The complexity of the index calculus algorithm for solving ECDLP on a curve defined over $\mathbb{F}_{2^n}$, using a factor base given by a vector space of dimension $l$, is $\tilde{O}(2^{n+l})$, where the $\tilde{O}$ hides a polynomial factor in $l$.*

*Proof.* In order to perform a whole ECDLP computation, one has to find $2^l$ relations. Following [8], the probability that a random point can be written as a sum of $m$ factor basis elements is heuristically approximated by $\frac{2^{ml}}{m!2^n}$. The time complexity for the full decomposition phase, using a dedicated WDSAT solver is:

$$CT2^{n+l}.$$

□

This worst case complexity is to be compared to the $O(2^{\omega\frac{n}{2}+l})$ complexity of Faugère *et al* [12]. Both approaches rely on the heuristic approximation of the probability that a random point can be decomposed in the factor base. However, we underline here that Faugère *et al*'s proof of this result is based on heuristic assumption on the Gröbner basis computation for PDP, while our analysis for the SAT-based approach simply relies on the rigorously proved worst case for the DPLL search tree (10).

## 6 Experimental Results

We conducted experiments using $S_4'$ on binary Koblitz elliptic curves [18] defined over $\mathbb{F}_{2^n}$. We experimented with Gröbner basis and SAT approaches. In [31], WDSAT is reported to outperform the Gröbner basis methods, as well as all generic SAT solvers for this particular problem. First, we confirm this by experimenting with higher parameters and results are reported in Table 2. Secondly, we extend the WDSAT solver with our symmetry breaking algorithm described in Section 4. Our symmetry breaking algorithm yields faster running times and we were able to perform experiments using greater parameters. Results are shown in Table 3. All tests were performed on a 2.40GHz Intel Xeon E5-2640 processor.

The Gröbner basis approach takes as input an algebraic model. We used the *grevlex* ordering, as this is considered to be optimal in the literature. The MINISAT solver processes a CNF model input, whereas CRYPTOMINISAT and WDSAT use the CNF-XOR model. WDSAT can also process directly an algebraic model in ANF form. Using the CNF-XOR model is a huge advantage, as it has far less clauses and variables than the CNF model. Gaussian elimination can be beneficial for SAT instances derived from cryptographic problems. However, it has been reported to yield slower running times for some instances, as performing the operation is very costly. For this reason, CRYPTOMINISAT and WDSAT do not include Gaussian elimination by default, but the feature can be turned on explicitly. We experimented with both variants for both XOR-able solvers.

With WDSAT we set a custom order of branching variables, which allowed us to make use of our findings in Proposition 1 and branch only on the $c_{i,j}$ variables. CryptoMiniSat does not have this feature in current version as authors report that custom order of branching variables leads to slower running times in most cases. We added this feature to the source code of CryptoMiniSat and we ran tests both with a custom order as per Proposition 1 and with the order chosen by the solver.

Table 2 compares different approaches, showing results from optimal variants of each solving tool. Running times of all variants of CryptoMiniSat and WDSAT are given in Appendix A. We experimented with different values of $n$ for each $l$ and we performed tests on 20 instances for each parameter size. Half of the instances have a solution and the other half do not. We show running time and memory averages on satisfiable and unsatisfiable instances separately, since these values differ between the two cases. SAT solvers stop as soon as they find a solution and if this is not the case they need to respond with certainty that a solution does not exist. Hence, running times of SAT solvers are significantly slower when there is no solution. On the other hand, [33] indicates that the computational complexity of Gröbner basis is lower when a solution does not exist.

We set a timeout of 10 hours and a memory limit of 200GB for each run. Using MiniSat, we were not able to solve the highest parameter instances ($l = 8$) within this time frame. On the other hand, Gröbner basis computations for these instances halted before timeout because of the memory limit. This data is in line with previous works. [33] and [25] show experiments using the fourth summation polynomial with $l = 6$, whereas the highest parameter size achieved in [13] is $l = 8$.

Table 2 shows the average runtime in seconds, the average number of conflicts and the average memory use in MB. The WDSat solver allocates memory statically, according to predefined constant memory requirements. This explains why memory averages do not vary much between the different size parameters, or between satisfiable and unsatisfiable instances.

| | | | SATisfiable | | | UNSATisfiable | | |
|---|---|---|---|---|---|---|---|---|
| Approach | l | n | Runtime | #Conflicts | Memory | Runtime | #Conflicts | Memory |
| Gröbner | 6 | 17 | 207.220 | NA | 3601 | 142.119 | NA | 3291 |
| | | 19 | 215.187 | NA | 3940 | 155.765 | NA | 4091 |
| | 7 | 19 | 3854.708 | NA | 38763 | 2650.696 | NA | 38408 |
| | | 23 | 3128.844 | NA | 35203 | 2286.136 | NA | 35162 |
| | 8 | 23 | | | >200GB | | | >200GB |
| | | 26[1] | | | >200GB | | | >200GB |
| MiniSat | 6 | 17 | 62.702 | 408189 | 12.7 | 270.261 | 1463309 | 24.2 |
| | | 19 | 229.055 | 1778377 | 23.6 | 388.719 | 2439933 | 29.8 |
| | 7 | 19 | 406.918 | 1919565 | 33.6 | 6777.431 | 25180492 | 105 |
| | | 23 | 12945.613 | 61610582 | 152 | 13260.586 | 59289671 | 163 |
| | 8 | 23 | 8027.974 | 63384411 | 256 | >10 hours | | |
| | | 26 | >10 hours | | | >10 hours | | |
| CMS with Prop.1 | 6 | 17 | 15.673 | 61812 | 34.5 | 62.396 | 260843 | 39.3 |
| | | 19 | 14.128 | 53767 | 33.2 | 64.563 | 259688 | 42.1 |
| | 7 | 19 | 176.463 | 484098 | 41.5 | 843.367 | 2077747 | 72.3 |
| | | 23 | 300.021 | 638152 | 48.9 | 1012.412 | 2070190 | 73.6 |
| | 8 | 23 | 1700.949 | 2420937 | 76.7 | 11959.938 | 16756106 | 82.4 |
| | | 26 | 3000.831 | 4179236 | 79.4 | 14412.193 | 16783213 | 81.8 |

[1] The non-prime degree case of $n = 26$ is not handled differently. The factor base is an $l$-dimensional vector space and the Weil descent does not include specific reductions which can be applied to non-prime degrees.

| | | | SATisfiable | | | UNSATisfiable | | |
|---|---|---|---|---|---|---|---|---|
| Approach | l | n | Runtime | #Conflicts | Memory | Runtime | #Conflicts | Memory |
| WDSAT with Prop.1 | 6 | 17 | .601 | 49117 | 1.4 | 3.851 | 254686 | 1.4 |
| | | 19 | .470 | 38137 | 1.4 | 3.913 | 255491 | 1.4 |
| | 7 | 19 | 9.643 | 534867 | 16.7 | 44.107 | 2073089 | 16.7 |
| | | 23 | 9.303 | 477632 | 16.7 | 47.347 | 2067168 | 16.7 |
| | 8 | 23 | 68.929 | 2646071 | 16.8 | 525.057 | 16666331 | 16.8 |
| | | 26 | 185.480 | 6261107 | 16.9 | 533.607 | 16684378 | 16.9 |

Table 2: Comparing different approaches for solving the PDP.

Choosing the WDSAT variant without Gaussian elimination as optimal, we continued experiments for bigger size parameters using this variant coupled with the breaking symmetry technique. Table 3 shows results for $l = \overline{6, 11}$ and $n$ sizes up to 89. All values are an average of 100 runs, as running times for satisfiable instances can vary remarkably. If we compare the number of conflicts for the first three $l$ sizes of the complete WDSAT solver with its symmetrical variant in Table 2, we observe a speedup factor that rapidly approaches 6.[2] This confirms our claims in Section 5 that the symmetry breaking technique proposed in this paper yields a speedup by a factor of $m!$.

Comparing results for $l = 6$ and $l = 7$ in Table 3 with the equivalent results for the Gröbner basis method in Table 2, we observe that WDSAT is up to 300 times faster than Gröbner basis for the cases where there is no solution and up to 1700 times faster for instances allowing a solution. This is a rough comparison, as the factor grows with parameters $l$ and $n$.

Lastly, we experimented with the collision search [32] generic method, using the open source code at [30]. This implementation solves the discrete log problem in the case of prime field curves. We did not adapt the code for extension fields and the computation time for multiplication on the curve might vary between the two cases. Even so, this allows for a rough comparison between the running times of generic methods and the work presented in this paper. In a uni-thread environment, a whole collision search computation for parameter $n = 59$ has an average runtime of 0.8 hours on our platform. Computing $2^l$ successful decompositions for parameters $n = 59$ and $l = 9$ would take more than 86 hours according to results in Table 3. The estimated running time becomes immensely worse when we take into account unsuccessful decompositions as well. We conclude that for the case of prime degree extension fields, even with the significant speedup that we achieved for the PDP, index calculus attacks are still not practical compared to the PCS generic method.

## 7 Conclusions and Future Work

Gröbner basis methods have been shown powerful in solving the PDP in the index calculus attack for elliptic curves defined over small degree extension fields in characteristic $> 2$. In this paper we argue that for finite fields in characteristic 2 a SAT-based approach

[2] We compare the cases where there is no solution, as these have more stable averages.

| | | SATisfiable | | | UNSATisfiable | | |
|---|---|---|---|---|---|---|---|
| l | n | Runtime | #Conflicts | Memory | Runtime | #Conflicts | Memory |
| 6 | 17 | .220 | 17792 | 1.4 | .605 | 43875 | 1.4 |
| | 19 | .243 | 19166 | 1.4 | .639 | 44034 | 1.4 |
| 7 | 19 | 2.205 | 130062 | 1.4 | 6.859 | 351353 | 1.4 |
| | 23 | 3.555 | 189940 | 1.4 | 7.478 | 350257 | 1.4 |
| 8 | 23 | 29.584 | 1145966 | 17.0 | 81.767 | 2800335 | 17.0 |
| | 26 | 39.214 | 1426216 | 17.0 | 85.822 | 2803580 | 17.0 |
| 9 | 37 | 447 | 10557129 | 17.1 | 1048 | 22396994 | 17.1 |
| | 47 | 609 | 12675174 | 17.2 | 1167 | 22381494 | 17.2 |
| | 59 | 611 | 11297325 | 17.3 | 1327 | 22390211 | 17.3 |
| | 67 | 677 | 11608420 | 17.4 | 1430 | 22388053 | 17.4 |
| 10 | 47 | 5847 | 95131900 | 17.3 | 11963 | 179019409 | 17.3 |
| | 59 | 6849 | 97254458 | 17.4 | 13649 | 179067171 | 17.4 |
| | 67 | 6530 | 88292215 | 17.4 | 14555 | 179052277 | 17.4 |
| | 79 | 7221 | 86174432 | 17.5 | 16294 | 179043408 | 17.5 |
| 11 | 59 | 64162 | 727241718 | 19.2 | 135801 | 1432191354 | 19.2 |
| | 67 | 70075 | 741222864 | 19.3 | 145357 | 1432183842 | 19.3 |
| | 79 | 61370 | 599263451 | 19.4 | 161388 | 1432120827 | 19.4 |
| | 89 | 85834 | 736610196 | 19.5 | 175718 | 1432099666 | 19.5 |

Table 3: Experimental results using the complete WDSAT solver. Running times are in seconds and memory use is in MB.

yields more practical results. We started by explaining that general-purpose SAT solvers cannot yield considerably faster running times because the number of variables in a SAT model is significantly larger than the number of variables in the algebraic model.

Our first contribution is to propose a PDP CNF-XOR model with only $ml$ core variables, whose assignment propagates all remaining variables in the model. To solve this model we use a SAT solver dedicated to solving systems derived from a Weil descent. As our second contribution, we optimized the time complexity of this solver by a factor of $m!$ using a symmetry breaking technique.

We presented experiments for the PDP on prime degree extension fields in characteristic 2, using parameter sizes of up to $l = 11$ and $n = 89$. This presents a significant improvement over the current state-of-the-art, as experiments using $l$ greater than 8 have never been shown before for this case. Moreover, memory is no longer a constraint for the PDP when the Gröbner basis computation is replaced with SAT solving.

Regarding future perspectives, it would be interesting to test the performance of SAT solvers on the simplified system obtained by considering the action of 2-torsion and 4-torsion points on the factor base, as in [13].

## References


1. Gilles Audemard and Laurent Simon. 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*, pages 399–404, 2009.

2. Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Hybrid approach for solving multivariate systems over finite fields. *J. Mathematical Cryptology*, 3(3):177–197, 2009.
3. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
4. Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).
5. Charles Bouillaguet, Chen-Mou Cheng, Tony (Tung) Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. Cryptology ePrint Archive, Report 2010/313, 2010. https://eprint.iacr.org/2010/313.
6. Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
7. Claus Diem. On the discrete logarithm problem in elliptic curves. *Compositio Mathematica*, 147(1):75–104, 2011.
8. Claus Diem. On the discrete logarithm problem in elliptic curves II. *Algebra & Number Theory*, 7(6):1281–1323, 2013.
9. J.-C. Faugère, L. Huot, A. Joux, G. Renault, and V. Vitse. Symmetrized summation polynomials: using small order torsion points to speed up elliptic curve index calculus. In *Advances in Cryptology- Eurocrypt 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 40–57. Springer, 2014.
10. Jean-Charles Faugère. A New Efficient Algorithm for Computing Gröbner basis (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.
11. Jean-Charles Faugère. A New Efficient Algorithm for Computing Gröbner Basis Without Reduction to Zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM.
12. Jean-Charles Faugère, Ludovic Perret, Christophe Petit, and Guénaël Renault. 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*, pages 27–44, 2012.
13. Steven D. Galbraith and Shishay W. Gebregiyorgis. Summation polynomial algorithms for elliptic curves in characteristic two. In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India*, volume 8885 of *Lecture Notes in Computer Science*, pages 409–427. Springer, 2014.
14. Pierrick Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symb. Comput.*, 44(12):1690–1702, 2009.
15. David Gérault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Revisiting AES related-key differential attacks with constraint programming. *Inf. Process. Lett.*, 139:24–29, 2018.
16. Cheng-Shen Han and Jie-Hong Roland Jiang. When Boolean Satisfiability Meets Gaussian Elimination in a Simplex Way. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification*, pages 410–426, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
17. A. Joux and V. Vitse. Cover and Decomposition Index Calculus on Elliptic Curves made practical. Application to a seemingly secure curve over $\mathbb{F}_{p^6}$. In *Advances in Cryptology-Eurocrypt 2012*, volume 7237, pages 9–26. Springer, 2012.
18. Neal Koblitz. CM-Curves with Good Cryptographic Properties. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 279–287, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

19. T. Laitinen, T. Junttila, and I. Niemela. Equivalence Class Based Parity Reasoning with DPLL(XOR). In *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pages 649–658, Nov 2011.

20. Tero Laitinen, Tommi A. Junttila, and Ilkka Niemelä. Conflict-Driven XOR-Clause Learning (extended version). *CoRR*, abs/1407.6571, 2014.

21. A. K. Lenstra, M. S. Manasse, , and J. M. Pollard. *The Number Field Sieve*, pages 11–42. Springer Berlin Heidelberg, 1993.

22. Daniel Lokshtanov, Ivan Mikhailin, Ramamohan Paturi, and Pavel Pudlák. Beating Brute Force for (Quantified) Satisfiability of Circuits of Bounded Treewidth. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 247–261, 2018.

23. Christophe Petit and Jean-Jacques Quisquater. 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*, volume 7658 of *Lecture Notes in Computer Science*, pages 451–466. Springer, 2012.

24. Igor A. Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. *IACR Cryptology ePrint Archive*, 2004:31, 2004.

25. Michael Shantz and Edlyn Teske. Solving the Elliptic Curve Discrete Logarithm Problem Using Semaev Polynomials, Weil Descent and Gröbner basis methods - an experimental study. In *Number Theory and Cryptography - Papers in Honor of Johannes Buchmann on the Occasion of His 60th Birthday*, pages 94–107, 2013.

26. João P. Marques Silva and Karem A. Sakallah. Conflict Analysis in Search Algorithms for Satisfiability. In *ICTAI*, pages 467–469. IEEE Computer Society, 1996.

27. Mate Soos. Enhanced Gaussian elimination in DPLL-based SAT solvers. In *In Pragmatics of SAT*, 2010.

28. Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT Solvers to Cryptographic Problems. In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

29. Niklas Sörensson and Niklas Eén. A SAT Solver with Conflict-Clause Minimization. *Proc. Theory and Applications of Satisfiability Testing*, 2005.

30. Monika Trimoska, Sorina Ionica, and Gilles Dequen. PCS. https://github.com/mtrimoska/PCS, 2019.

31. Monika Trimoska, Sorina Ionica, and Gilles Dequen. Parity (xor) reasoning for the index calculus attack, 2020.

32. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

33. Huang Yun-Ju, Christophe Petit, Naoyuki Shinohara, and Tsuyoshi Takagi. Improvement to Faugère et al.'s method to solve ECDLP. In Kazuo Sakiyama and Masayuki Terada, editors, *Advances in Information and Computer Security - 8th International Workshop on Security, IWSEC 2013*, volume 8231 of *Lecture Notes in Computer Science*, pages 115–132. Springer, 2013.

# A Comparing different variations of CryptoMiniSat and WDSAT

| | | | SATisfiable | | | UNSATisfiable | | |
|---|---|---|---|---|---|---|---|---|
| Approach | l | n | Runtime | #Conflicts | Memory | Runtime | #Conflicts | Memory |
| CMS | 6 | 17 | 133.983 | 775948 | 48.4 | 363.513 | 1709971 | 59.5 |
| | | 19 | 560.080 | 3396192 | 64.1 | 1172.740 | 5726372 | 70.1 |
| | 7 | 19 | 1210.612 | 5713259 | 85.3 | 10258.351 | 26079224 | 117 |
| | | 23 | 3637.032 | 12159752 | 80.4 | 19857.454 | 47086152 | 130 |
| | 8 | 23 | 9846.554 | 18509058 | 123 | >10 hours | | |
| | | 26 | 6905.477 | 13269631 | 115 | >10 hours | | |
| $\text{CMS}_{\text{GE}}$ | 6 | 17 | 119.866 | 677336 | 54.5 | 436.811 | 1877699 | 64.2 |
| | | 19 | 224.484 | 1219840 | 58.7 | 615.952 | 2763754 | 76.5 |
| | 7 | 19 | 893.425 | 3722805 | 86.5 | 3587.929 | 8642108 | 107 |
| | | 23 | 580.007 | 1753040 | 82.4 | 3253.786 | 8183887 | 132 |
| | 8 | 23 | 11265.010 | 19604250 | 155 | >10 hours | | |
| | | 26 | 3933.637 | 7920920 | 157 | >10 hours | | |
| CMS WITH PROP.1 | 6 | 17 | 15.673 | 61812 | 34.5 | 62.396 | 260843 | 39.3 |
| | | 19 | 14.128 | 53767 | 33.2 | 64.563 | 259688 | 42.1 |
| | 7 | 19 | 176.463 | 484098 | 41.5 | 843.367 | 2077747 | 72.3 |
| | | 23 | 300.021 | 638152 | 48.9 | 1012.412 | 2070190 | 73.6 |
| | 8 | 23 | 1700.949 | 2420937 | 76.7 | 11959.938 | 16756106 | 82.4 |
| | | 26 | 3000.831 | 4179236 | 79.4 | 14412.193 | 16783213 | 81.8 |
| $\text{CMS}_{\text{GE}}$ WITH PROP.1 | 6 | 17 | 17.698 | 62161 | 39.1 | 86.049 | 294428 | 63.2 |
| | | 19 | 16.301 | 52730 | 39.8 | 88.738 | 293859 | 62.7 |
| | 7 | 19 | 220.037 | 479197 | 51.2 | 2551.277 | 2418051 | 72.5 |
| | | 23 | 367.105 | 653673 | 59.4 | 1329.494 | 2380614 | 93.1 |
| | 8 | 23 | 2493.328 | 2419268 | 112 | 19058.671 | 19359334 | 164 |
| | | 26 | 4956.952 | 4171674 | 126 | 19907.670 | 19534832 | 167 |
| WDSAT with Prop.1 | 6 | 17 | .601 | 49117 | 1.4 | 3.851 | 254686 | 1.4 |
| | | 19 | .470 | 38137 | 1.4 | 3.913 | 255491 | 1.4 |
| | 7 | 19 | 9.643 | 534867 | 16.7 | 44.107 | 2073089 | 16.7 |
| | | 23 | 9.303 | 477632 | 16.7 | 47.347 | 2067168 | 16.7 |
| | 8 | 23 | 68.929 | 2646071 | 16.8 | 525.057 | 16666331 | 16.8 |
| | | 26 | 185.480 | 6261107 | 16.9 | 533.607 | 16684378 | 16.9 |
| $\text{WDSAT}_{\text{GE}}$ with Prop.1 | 6 | 17 | 9.193 | 48178 | 1.4 | 56.718 | 253123 | 1.4 |
| | | 19 | 7.041 | 36835 | 1.4 | 58.876 | 252799 | 1.4 |
| | 7 | 19 | 169.629 | 528383 | 16.7 | 736.863 | 2062232 | 16.7 |
| | | 23 | 159.101 | 473223 | 16.7 | 779.432 | 2060501 | 16.7 |
| | 8 | 23 | 1290.702 | 2630567 | 16.8 | 9124.361 | 16639322 | 16.8 |
| | | 26 | 3404.765 | 6231289 | 16.9 | 9623.677 | 16636122 | 16.9 |

Table 4: Comparing different variations of CryptoMiniSat and WDSAT for solving the PDP.