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 propose an original XOR-reasoning SAT solver, named WDSAT, dedicated to this specific problem. While asymptotically solving the point decomposition problem with our method has exponential worst time complexity in the dimension $l$ of the vector space defining the factor base, experimental running times show that our solver is significantly faster than current algebraic methods based on Gröbner basis computation. For the values $l$ and $n$ considered in the experiments, WDSAT was up to 300 times faster than Magma’s F4 implementation, and this factor grows with $l$ and $n$. Our solver outperforms as well current best state-of-the-art SAT solvers for this specific problem.

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) [24], dedicated to logarithms in $\mathbb{Z}_p$ and the algorithms of Gaudry [15] and Diem [7] for algebraic curves in $\mathbb{F}_p^n$. 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 $O(p^{2-2/n})$, but this hides an exponential factor in $n$ which comes from the hardness of solving the point decomposition problem.

Consequently, when $p$ is large and $n$ is small, the Gaudry-Diem algorithm has a better asymptotic complexity than generic methods for solving the discrete logarithm problem and Gröbner basis algorithms have become a well-established technique [19] 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 [35]. Finally, note that for random systems, pure Gröbner basis algorithms are both theoretically and practically slower than simpler methods, typically exhaustive search [5,25], 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 etc. [16,22,17,33,32]. However, their use in public key cryptosystems has rarely been considered. A prominent example is the work of Galbraith and Gebregiyorgis [14], 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 [14]. Secondly, we propose a dedicated SAT solver for our model. Specifically, our solver is adapted for XOR-reasoning, which is one of the core operations in polynomial systems. We implement a DPLL-based backtracking procedure with three modules - a state-of-the-art unit propagation module and two XOR-reasoning modules. We show that by using this solver, the proven worst case complexity of solving a PDP is $O(2^{ml})$, 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 [34,33]). 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 our WDSAT solver using only 17MB of memory, requires more than 200GB when using the Gröbner basis method.

We estimate that our solver can be used for any attack that can be reduced to the problem of solving a multivariate boolean polynomial system. Notable examples are the HFE cryptosystem [26] and HFE-inspired schemes which are promising candidates for the NIST Post-Quantum Cryptography Standardization [27,13,1].

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. Sections 3 and 4 detail the reasoning models used in our experiments and our 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, CRYPTO-MINISAT and GLUCOSE.

2 An Overview of Index Calculus

In 2008 and 2009, Gaudry [15] 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 [29]. 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

$$E : y^2 + xy = x^3 + ax^2 + b,$$

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 $S_m(x_{P_1}, \ldots, x_{P_m}) = 0$. We have that

$$S_2(X_1, X_2) = X_1 + X_2,$$
$$S_3(X_1, X_2, X_3) = X_1^2 X_2^2 + X_1^2 X_3^2 + X_1 X_2 X_3 + X_2^2 X_3^2 + b,$$

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

$$S_m(X_1, \ldots, X_m) = \text{Res}_X(S_{m-k}(X_1, \ldots, X_{m-k-1}, X), S_{k+2}(X_{m-k}, \ldots, X_m, X)).$$
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 $B$ to be:

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

Heuristically, we can easily see that the factor base has approximately $2^l$ elements.

Given a point $R \in E(\mathbb{F}_{2^n})$, the point decomposition problem is to find $m$ points $P_1, \ldots, P_m \in B$ 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 $(x_1, \ldots, x_m) \in V^m$ such that $f(x_1, \ldots, x_m) = 0$.

Using the fact that $\mathbb{F}_{2^n}$ is an $n$-dimensional vector space over $\mathbb{F}_2$, the equation $f(x_1, \ldots, x_m) = 0$ 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 [15] or Weil descent [28]. 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.

2.1 Solving the decomposition problem using Gröbner basis

In this section, we give a brief account on the complexity of the index calculus algorithm obtained using the Gröbner approach. It is well known that the complexity of Gröbner basis algorithms based on Lazard’s Gaussian elimination on Macaulay matrices [23] strongly depends on the maximal degree $D$ reached during the computation, called the degree of regularity of the system.

In 2012, Faugère et al. [12] proposed a dedicated Gröbner basis algorithm based on techniques “à la Lazard” which solves PDP in time $O(2^{\omega \tau})$ and memory $O(2^{2\tau})$, where $\omega$ is the linear algebra constant and $\tau \approx n/2$. This running time is obtained under the unproven yet plausible heuristic assumption that a certain set of equations arising in the algorithm is linearly independent and also by establishing a certain bound on the degree of regularity. As a consequence, they derive an index calculus algorithm solving the DLP over an elliptic curve defined over a binary field $\mathbb{F}_{2^n}$ having a worst time complexity $O(2^{\omega \tau + l})$. 

Later on, experiments carried out by Petit and Quisquater [28] using Magma’s $F_4$ [4] implementation suggested that the degree of regularity of the system derived from a PDP instance via the Weil restriction is much smaller than the bound obtained

\footnote{In [12] the authors state the complexity is $O(2^{2\tau})$. In the proof of their Theorem 3, the exact time is $2^n \log m + l^{1+\omega \tau}$. Since $l$ is large, we cannot ignore it here.}
in [12]. Going further, the authors of [28] introduce new heuristic assumptions on the bound on the degree of regularity of such a system and claim the existence of a subexponential index calculus algorithm under these assumptions. If correct, the result of Petit and Quisquater would be remarkable. Unfortunately, the experimental data available in the literature concerns all small values of $l$ and $n$ and several papers in this area [18,30] debate on the validity of the assumptions made in [28]. For this reason, we only compare ourselves to the results obtained in Faugère et al.

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

\[ e_1 = \sum_{1 \leq i_1 \leq m} X_{i_1}, \]
\[ e_2 = \sum_{1 \leq i_1, i_2 \leq m} X_{i_1}X_{i_2}, \]
\[ \ldots \]
\[ e_m = \prod_{1 \leq i \leq m} X_i. \]

We denote $S'_{m+1}$ the polynomial obtained after symmetrizing $S_{m+1}(X_1, X_2, \ldots, X_m, X_{m+1})$ 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 [35], 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 [35], 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 [35], 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:

\[ e_1 = d_{1,0} + \ldots + d_{1,l-1}t^{l-1} \]
\[ e_2 = d_{2,0} + \ldots + d_{2,2l-2}t^{2l-2} \]
\[ \ldots \]
\[ e_m = d_{m,0} + \ldots + d_{m,m(l-1)}t^{m(l-1)} \]

where the $d_{i,j}$ with $1 \leq i \leq m$, $0 \leq j \leq i(l-1)$ are binary variables. After choosing $x_{m+1} \in \mathbb{F}_{2^n}$ and substituting $e_1, \ldots, e_m$ as in Equation (5), we get:

\[ S'_{m+1}(e_1, \ldots, e_m, x_{m+1}) = f_0 + \ldots + f_{n-1}t^{n-1}, \]

5
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

\[
f_0 = f_1 = \ldots = f_{n-1} = 0. \tag{6}
\]

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 [14]. Since this is a restriction, we did not implement this approach and used the system in Equation (6) as the starting point for our SAT model of the point decomposition problem.

2.2 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.

Propositional variables can take two possible truth values: TRUE and FALSE. We denote a propositional variable by \( x \).

- A literal is a signed propositional variable. Therefore, it can be positive (denoted by \( x \)) or negative (denoted by \( \neg x \)). A literal \( x \) (resp. \( \neg x \)) is satisfied if it is assigned to TRUE (resp. FALSE). A literal \( x \) (resp. \( \neg x \)) is falsified if it is assigned to FALSE (resp. TRUE);

- An OR-clause is a non-exclusive disjunction (\( \lor \)) of literals (e.g. \( x_1 \lor \neg x_2 \lor x_3 \)). An OR-clause is said to be falsified if all of its literals are falsified and it is set to be satisfied if at least one of its literals is satisfied;

- A XOR-clause is an exclusive disjunction (\( \oplus \)) of literals. (e.g. \( x_1 \oplus \neg x_2 \oplus x_3 \)). A XOR-clause is said to be satisfied (resp. falsified) if an even (resp. odd) number of its literals is satisfied;

- An interpretation of a given propositional formula consists in assigning a truth value to a set of its variables;

- A CNF formula is a conjunction (\( \land \)) of one or more OR-clauses. A CNF formula is said to be satisfiable if there exists at least one interpretation which satisfies all of its OR-clauses, and it is said to be unsatisfiable when the opposite is true. Every formula in propositional logic has a closed-CNFForm.

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 propositional satisfiability problem (SAT) is the problem of determining whether a (usually CNF) formula is satisfiable. A SAT solver is a special purpose program to solve the SAT problem.

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:

6
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 \( \text{TRUE} \) 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 \( \text{FALSE} \) 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 10 clauses numbered \( C_1 \) to \( C_{10} \).

\[
\begin{align*}
C_1 : x_1 \lor x_2 \lor x_3 \\
C_2 : \neg x_3 \lor x_4 \lor \neg x_5 \\
C_3 : \neg x_3 \lor x_1 \lor \neg x_4 \\
C_4 : \neg x_2 \lor x_5 \lor x_6 \\
C_5 : \neg x_6 \lor \neg x_4 \lor x_5 \\
C_6 : x_1 \lor \neg x_4 \lor x_6 \\
C_7 : x_1 \lor x_4 \lor \neg x_5 \\
C_8 : x_1 \lor \neg x_2 \lor \neg x_4 \\
C_9 : x_4 \lor \neg x_5 \lor \neg x_6 \\
C_{10} : \neg x_2 \lor x_3 \lor x_4
\end{align*}
\]

Assigning the variable \( x_5 \) to \( \text{TRUE} \) leads the clauses \( C_4 \) and \( C_5 \) to be satisfied by the literal \( x_5 \). As well and as a consequence, clauses \( C_2, C_7 \) and \( C_9 \) cannot be satisfied thanks to the literal \( x_5 \). Hence, \( \neg x_5 \) can be deleted from these clauses. At this step, our formula is :

\[
\begin{align*}
C_1 : x_1 \lor x_2 \lor x_3 \\
C_4 : \\
C_7 : x_1 \lor x_4 \\
C_8 : x_1 \lor \neg x_2 \lor \neg x_4 \\
C_{10} : \neg x_2 \lor x_3 \lor x_4
\end{align*}
\]

Using the same reasoning, assigning the variable \( x_3 \) to \( \text{TRUE} \) leads the clauses \( C_1, C_7 \) and \( C_{10} \) to be satisfied and the literal \( \neg x_3 \) to be deleted from the clauses \( C_2 \) and \( C_3 \). At this step, our formula is :

\[
\begin{align*}
C_1 : \\
C_2 : x_4 \\
C_4 : \\
C_7 : C_8 : x_1 \lor \neg x_2 \lor \neg x_4 \\
C_{10} : \neg x_2 \lor x_3 \lor x_4
\end{align*}
\]
Then, $C_2$ is a unit clause composed of the literal $x_4$ and as a consequence, $x_4$ has to be assigned to TRUE. We say that the truth value of $x_4$ is inferred through unit propagation.

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 (at worst) to the number of elementary 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 $O(c2^n)$, where $c$ is the number of clauses in the formula and $v$ is the number of variables.

A common variation of the DPLL is the conflict-driven clause learning (CDCL) algorithm [31]. 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) [33,32], Han and Jiang [17], Laitinen et al.[22,21]. The CRYPTOMINISAT solver, specifically designed for exploiting the XOR operation in cryptographic problems, offers the possibility either to perform only top-level Gaussian elimination or to perform this operation dynamically during the CDCL process. One has to choose the best strategy depending on the problem at hand.

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 [35], the CNF model used by Galbraith and Gebregiyorgis [14] 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 (5). Similarly, since we look for a set of solutions $(x_1, \ldots, x_m) \in V^m$, the $X_i$
variables are written formally as follows:

\[ X_1 = c_{1,0} + \ldots + c_{1,l-1}t^{l-1} \]
\[ X_2 = c_{2,0} + \ldots + c_{2,l-1}t^{l-1} \]
\[ \ldots \]
\[ X_m = c_{m,0} + \ldots + c_{m,l-1}t^{l-1} \]

where \( c_{i,j} \), with \( 1 \leq i \leq m \), \( 0 \leq j \leq l - 1 \), are binary variables. Using Equation (4), we derive the following equations:

\[ d_{1,0} = c_{1,0} + \ldots + c_{m,0} \]
\[ d_{1,1} = c_{1,1} + \ldots + c_{m,1} \] (7)
\[ \ldots \]
\[ d_{m,m(l-1)} = c_{1,l} \ldots c_{m,l} \]

The remaining equations correspond to polynomials \( f_i \), \( 0 \leq i \leq n - 1 \), obtained in Equation (5) via the Weil descent on \( S_{m+1}^l \). 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 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 (7) is as follows:

\[ \neg d_{1,0} \oplus c_{1,0} \oplus \ldots \oplus c_{m,0} \]
\[ \neg d_{1,1} \oplus c_{1,1} \oplus \ldots \oplus c_{m,1} \] (8)
\[ \ldots \]
\[ \neg d_{m,m(l-1)} \oplus (c_{1,l} \ldots c_{m,l}) \]

SAT solvers adapted for XOR reasoning in the literature contain XOR clauses formed by xoring single literals, and not conjunctions of several ones. Since our solver follows 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 \((c_{i_1,j_1} \land c_{i_2,j_2} \land \ldots \land c_{i_k,j_k})\). We have \( c' \leftrightarrow (c_{i_1,j_1} \land c_{i_2,j_2} \land \ldots \land c_{i_k,j_k}) \), which rewrites as

\[ (c' \lor \neg c_{i_1,j_1} \lor \neg c_{i_2,j_2} \lor \ldots \lor \neg c_{i_k,j_k}) \land \]
\[ (\neg c' \lor c_{i_1,j_1}) \land \]
\[ (\neg c' \lor c_{i_2,j_2}) \land \]
\[ \ldots \]
\[ (\neg c' \lor c_{i_k,j_k}) \] (9)
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 (9) 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 (8) 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 XOR model. The WDSAT solver is adapted for XOR reasoning and takes a XOR model as input.

**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 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 $c' \iff (c_{i_1,j_1} \land c_{i_2,j_2} \land \ldots \land c_{i_k,j_k})$. 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 (7). 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 $d' \iff (d_{i_1,j_1} \land d_{i_2,j_2} \land \ldots \land d_{i_k,j_k})$. 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 XOR model. $\square$

Transformation of the polynomial $S'_{m+1}$ to a reasoning model is certainly not the bottleneck of the PDP. It is however done once for each point decomposition, hence at least $2^l$ times in a full ECDLP computation. To optimize the transformation, we split the process in two phases. The first phase is done after choosing a factor base, and before starting point decomposition. At this point, we can do the whole Weil descent...
process except for the last step - multiplying all monomials with the coefficients of vector $X_{m+1}$. Three sets of clauses (X-substitutions, E-substitutions and $E$-$X$-relations) are created during this phase.

When a random point to decompose is chosen, we can do the last step and deduce the $F$ set. Without giving too much detail, we implement the transformation in a way that the second phase is computationally faster at the expense of the first one.

### 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 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

$$
(x_1 \lor \neg x_2 \lor \neg x_3) \land \\
(\neg x_1 \lor x_2 \lor \neg x_3) \land \\
(\neg x_1 \lor \neg x_2 \lor x_3) \land \\
(x_1 \lor x_2 \lor x_3)
$$

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 $x'$ and we obtain the following two XOR-clauses:

$$(x_1 \oplus \ldots \oplus x_i \oplus x') \land$$

$$(x_{i+1} \oplus \ldots \oplus x_k \oplus \neg x').$$

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. Determining the optimal size at which to cut the XOR-clauses was out of scope for our work and we did not experiment with this parameter.

Table 1 serves as a comparison on the number of variables, equations and clauses between the three models described in this section. Values for the algebraic and XOR model are exact, whereas those for the CNF model are averages obtained from experiments presented in Section 6.

In 2014, Galbraith and Gebregiyorgis [14] 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 XOR model. This motivated our choice of the XOR model for this work.

4 A dedicated SAT solver for solving PDP

In this Section, we present a built-from-scratch SAT solver, named WDSAT, dedicated to solving the model described in Section 3. We recall from Proposition 1 that assigning all $c_{i,j}$ variables in the XOR model leads to the assignment of all variables through unit propagation. Following the DPLL algorithm explained in Section 2, we construct a binary search tree by trying to recursively assign a truth value to each $c_{i,j}$ variable. The solver implements three reasoning modules - one module for the CNF-part and two for XOR-reasoning. The input of the modules is a set of CNF-clauses, or respectively, XOR-clauses and the output is a list of inferred truth values or a conflict signal. They are briefly detailed below.

- **CNF module** This module, dedicated to reasoning from the CNF-part of the model, contains structures designed for fast unit propagation on CNF-clauses.

<table>
<thead>
<tr>
<th>n</th>
<th>#Vars</th>
<th>#Equations</th>
<th>#Vars</th>
<th>#CNF-clauses</th>
<th>#Vars</th>
<th>#CNF-clauses</th>
<th>#XOR-clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>13</td>
<td>4</td>
<td>42</td>
<td>2474</td>
<td>9552</td>
<td>302</td>
<td>1505</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>5</td>
<td>44</td>
<td>2941</td>
<td>11433</td>
<td>502</td>
<td>1505</td>
</tr>
<tr>
<td>7</td>
<td>17</td>
<td>5</td>
<td>50</td>
<td>4686</td>
<td>18237</td>
<td>767</td>
<td>2364</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>5</td>
<td>51</td>
<td>5019</td>
<td>19577</td>
<td>767</td>
<td>2364</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td>6</td>
<td>58</td>
<td>6981</td>
<td>27216</td>
<td>1103</td>
<td>3466</td>
</tr>
<tr>
<td>10</td>
<td>23</td>
<td>6</td>
<td>62</td>
<td>8223</td>
<td>32201</td>
<td>1103</td>
<td>3466</td>
</tr>
<tr>
<td>11</td>
<td>23</td>
<td>7</td>
<td>66</td>
<td>11036</td>
<td>43210</td>
<td>1510</td>
<td>4835</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>8</td>
<td>71</td>
<td>12074</td>
<td>47374</td>
<td>1510</td>
<td>4835</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>8</td>
<td>88</td>
<td>20969</td>
<td>82721</td>
<td>2000</td>
<td>6495</td>
</tr>
<tr>
<td></td>
<td>47</td>
<td>9</td>
<td>98</td>
<td>25456</td>
<td>100709</td>
<td>2000</td>
<td>6495</td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>10</td>
<td>110</td>
<td>31942</td>
<td>126702</td>
<td>2000</td>
<td>6495</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>11</td>
<td>118</td>
<td>35917</td>
<td>142632</td>
<td>2000</td>
<td>6495</td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>12</td>
<td>122</td>
<td>49538</td>
<td>196434</td>
<td>3247</td>
<td>10784</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>13</td>
<td>130</td>
<td>55310</td>
<td>219553</td>
<td>3247</td>
<td>10784</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>14</td>
<td>142</td>
<td>63531</td>
<td>252485</td>
<td>3247</td>
<td>10784</td>
</tr>
<tr>
<td></td>
<td>89</td>
<td>15</td>
<td>152</td>
<td>71556</td>
<td>284626</td>
<td>3247</td>
<td>10784</td>
</tr>
</tbody>
</table>

Table 1: The number of variables and equations/clauses for the three models. The models for $n = 26$ are obtained by using an $l$-dimensional vector space to define the factor base and by doing a Weil descent on $\mathbb{F}_2$, in a similar way to the prime case. We did not use specific reductions which can be obtained by working in subfields.
• **XORSET module** This module performs unit propagation on the parity constraints. When all except one literal in a XOR clause is assigned, we infer the truth value of the last literal according to parity reasoning.

• **XORGAUSS module** The second XOR-reasoning module performs Gaussian elimination on the XOR system dynamically - once before starting the solving process and then on each level of the binary search tree. During the initialization process, XOR-clauses are normalized and represented as equivalence classes. A clause in normal form contains only positive literals and does not contain more than one occurrence of the same variable. Presenting the XOR-clauses simply as equivalence classes allows us to avoid performing matrix operations, that can be computationally intensive.

All modules implement a `SET_IN` function which takes as input a list of literals and a propositional formula $F$. It consists in setting all literals from the list to `TRUE` and in consequence, simplifying $F$ and inferring truth values for other literals through the appropriate unit propagation technique. This function sends a conflict signal when a conflict is reached. In addition, all modules have a `LAST_ASSIGNED` function which returns the list of literals that were assigned during the last call to the respective `SET_IN` function.

When an assumption of a truth value for a literal $x$ is made, the solver initially tries to set it in the `CNF` module. If the literal $x$ is set successfully (it does not result in conflict), it will infer truth values for other literals. All of the inferred, as well as the initial literal are set in the `XORSET` module. The new list of XOR-implied literals is then set in the `CNF` module. We do this process back and forth until there are no more new implications.

When the process stops, the list of all inferred literals since the beginning of the process, as well as the initial $x$, are transferred to the `XORGAUSS` module. If the `XORGAUSS` module finds new XOR-implied literals, the list is sent to the `CNF` module and the process is restarted. When all modules are stabilized and there are no more inferred literals to set, we go one level further in the search tree and a new assumption is made.

If a conflict is reached in any of the reasoning modules, the process is stopped and a backtracking procedure is launched.

This concludes the entire process of assigning a truth value which is presented as the `ASSIGN` function detailed in Algorithm 1. In the algorithm, `to_set` is a list containing literals to be set in the `CNF` and `XORSET` modules, whereas `to_set` in `XORGAUSS` is a list containing literals to be set in the `XORGAUSS` module.

The `ASSIGN` function is called by a recursive `SOLVE` function which is at the core of the WDSAT solver and is detailed in Algorithm 2. On line 4 of Algorithm 2, we arbitrarily choose one of the $c_{i,j}$ variables. At first, we try to assign a truth value of `FALSE` to the chosen $c_{i,j}$ variable. If the formula can not be satisfied after this assumption, we apply the classic backtracking technique (function `BACKTRACK`) and the opposite truth value is set. If the formula can not be satisfied with a value of `TRUE` for $c_{i,j}$ either, we conclude that the formula is unsatisfiable.

**Remark 1.** The conflict-driven clause learning (CDCL) variation [31] of the DPLL algorithm has been shown to yield a significant performance improvement for a number of
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: Propositional formula $F$, Literal $x$

Output: FALSE if a conflict is reached, TRUE and $F$ simplified thanks to unit propagation within the three modules otherwise.

1: $to-set$ $\leftarrow \{x\}$.
2: $to-set_{in\_XORGAUSS}$ $\leftarrow \{x\}$.
3: while $to-set \neq \emptyset$ do
   4:      while $to-set \neq \emptyset$ do
      5:         if $SET_{in\_CNF}(to-set, F)$ $\rightarrow$ conflict then
         6:               return (FALSE, $-$).
      7:      end if
     8:      $to-set$ $\leftarrow$ $LAST\_ASSIGNED\_IN\_CNF()$.
    9:      $to-set_{in\_XORGAUSS}$ $\leftarrow$ $to-set$.
   10:     if $SET_{in\_XORSET}(to-set, F)$ $\rightarrow$ conflict then
       11:        return (FALSE, $-$).
   12:    end if
  13:    $to-set$ $\leftarrow$ $LAST\_ASSIGNED\_IN\_XORSET()$.
  14:    $to-set_{in\_XORGAUSS}$ $\leftarrow$ $to-set \cup to-set_{in\_XORGAUSS}$.
  15: end while
16: if $SET_{in\_XORGAUSS}(to-set_{in\_XORGAUSS}, F)$ $\rightarrow$ conflict then
  17:    return (FALSE, $-$).
18: end if
19: $to-set$ $\leftarrow$ $LAST\_ASSIGNED\_XORGAUSS()$.
20: end while
21: return (TRUE, $F$).

SAT problems. Indeed, it is at the core of many modern general-purpose SAT solvers. However, because of the nature of SAT models coming from a boolean polynomial system, we estimate that the cost of CDCL would outweigh its eventual benefit for this particular problem. Indeed, the only information that the CNF-part holds is the equivalence between a literal and a conjunction. A set of clauses of the form (9) is independent from other such sets. Consequently, WDSAT does not implement CDCL techniques. Comparisons of running times obtained with WDSAT to running times for CDCL based solvers, in Section 6, confirm this assumption.

4.1 Breaking symmetry

From the symmetry of Semaev’s summation polynomials we have that when $\{x_1, \ldots, x_m\}$ is a solution, all permutations of this set are a solution as well. These solutions are equivalent and finding more than one is of no use for the PDP. We observe redundancy in the binary search tree. Indeed, for $m = 3$ when a potential solution $\{x_1, x_2, x_3\}$ has been eliminated, $\{x_2, x_1, x_3\}$ does not need to be tried out. To avoid this redundancy, we establish the following constraint $x_1 \leq x_2 \leq \ldots \leq x_m$.

It would be tedious to add this constraint in the model itself. Any solution implies adding clauses and weighing the SAT model. Instead, we decided to implement this
Algorithm 2 Function solves(F) : Recursive function for solving a SAT formula derived from PDP.

**Input:** Propositional formula F

**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 one ci,j.
5. (contradiction, F') ← ASSIGN(F, ¬ci,j).
6. if contradiction then
7. BACKTRACK().
8. else
9. if solves(F') returns FALSE then
10. BACKTRACK().
11. else
12. return TRUE.
13. end if
14. end if
15. (contradiction, F') ← ASSIGN(F, ci,j).
16. if contradiction then
17. BACKTRACK().
18. return FALSE.
19. end if
20. return solves(F').

constraint in the solver using a tree-pruning-like technique. We apply this technique on top of the recursive function solves in Algorithm 2. In the function solves we were trying out both FALSE and TRUE for the truth value of a chosen variable. In the breaking symmetry variation of solves, denoted solves_BR_SYM, 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 x1 ≤ x2 ≤ ... ≤ xm. The new algorithm for the solves_BR_SYM function is detailed in Algorithm 3 and the line numbers that distinguish it from Algorithm 2 are in bold. Note that one crucial difference between the two algorithms is the choice of a variable on line 4. While this choice was arbitrary in Algorithm 2, in Algorithm 3 variables need to be chosen in the order from the leading bit of x1 to the trailing bit of xm. If this is not respected, solves_BR_SYM does not yield a correct answer.

Continuing with the notation from Section 3, ci,j corresponds to the jth bit of the ith x-vector, where 2 ≤ i ≤ m and 1 ≤ j ≤ l. In Algorithm 3, we decide whether to try out the truth value of FALSE for ci,j or not by comparing two x-vectors bit for bit, in the same way that we would compare binary numbers. When we are deciding on the truth value of ci,j we have the following reasoning:

- If ci−1,j is FALSE, we try to set ci,j both to FALSE and TRUE (if FALSE fails). When ci,j is set to FALSE, all of the potential x solutions are greater than or equal to xi−1, thus we continue with the same bit comparison on the next level. However, when ci,j is set to TRUE, all of the potential x solutions are strictly greater than 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 \( x_i \) solutions are greater than or equal to \( x_{i-1} \) at this point.

Lastly, we give further information which explain in full detail Algorithm 3. We use a flag denoted \( \text{compare} \) to instruct whether to do bit comparison at the current search tree level or not. On line 6 we reset the \( \text{compare} \) flag to TRUE since \( c_{i,j} \), when \( j = 0 \), 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 3 hides 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.

**Algorithm 3** Function \( \text{SOLVE_BR_SYM}(F, \text{compare}) \) : Recursive function for solving a SAT formula derived from PDP.

**Input:** Propositional formula \( F \) and a flag \( \text{compare} \)

**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 one \( c_{i,j} \).
5. if \( j = 0 \) then
6. \( \text{compare} \leftarrow \) TRUE.
7. end if
8. if (\( \text{compare} \) is FALSE) or (\( i = 1 \)) or (\( c_{i-1,j} \) is set to FALSE) then
9. (\( \text{contradiction}, F' \) ) \( \leftarrow \) \( \text{ASSIGN}(F, \neg c_{i,j}) \).
10. if contradiction then
11. \text{BACKTRACK}().
12. \( \text{compare} \leftarrow \) FALSE.
13. else
14. if \( \text{SOLVE_BR_SYM}(F', \text{compare}) \) returns FALSE then
15. \text{BACKTRACK}().
16. \( \text{compare} \leftarrow \) FALSE.
17. end if
18. end if
19. return TRUE.
20. end if
21. end if
22. (\( \text{contradiction}, F' \) ) \( \leftarrow \) \( \text{ASSIGN}(F, c_{i,j}) \).
23. if contradiction then
24. \text{BACKTRACK}().
25. return FALSE.
26. end if
27. return \( \text{SOLVE_BR_SYM}(F', \text{compare}) \).
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 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 4.1, we optimize this complexity by a factor of $m!$. Indeed, out of the $m!$ permutations of the solution set $\{x_1, \ldots, x_m\}$, only one satisfies $x_1 \leq x_2 \leq \ldots \leq x_m$ (neglecting the equality).

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

$$\frac{2^{ml}}{m!}.$$  \hspace{1cm} (11)

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 (9)). In total, the number of clauses in the $X$-substitutions set is

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

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 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 2.** 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.

$$C \approx \left(\binom{3}{2} \cdot 3l^2 + \binom{3}{3} \cdot 4l^3 + \left(\binom{3}{2}\right) \cdot 3(3l - 2)^2 + (6l - 3) + n \right) \approx 4l^3 + 171l^2 - 210l + n + 69.$$ \hspace{1cm} (12)

In practice, many monomials have no occurrence in the system after the Weil descent process. In fact, the value in (12) 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 $l \approx \frac{n}{m}$, 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], when $l \approx \frac{n}{m}$ the probability that a random point can be written as a sum of $m$ factor basis elements is heuristically approximated by $\frac{2^{ml}}{2^{lm}}$. The time complexity for the full decomposition phase, using our dedicated WDSAT solver is:

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

This worst case complexity is to be compared to the $O(2^{-\frac{n}{2}+l})$ complexity of Faugère et al [12]. Moreover, 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 rigourously proved worst case for the DPLL search tree.

### 6 Experimental Results

We conducted experiments using $S'_4$ on binary Koblitz elliptic curves [20] defined over $\mathbb{F}_{2^n}$. We experimented with Gröbner basis and SAT approaches. 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. MINISAT and GLUCOSE solvers process a CNF model input, whereas CRYPTO MINISAT and WDSAT use the XOR model. Using the 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 runtimes for some instances as performing the operation is very costly. For this reason, CryptoMiniSat does not include Gaussian elimination by default, but the feature can be turned on explicitly. We experimented with both variants, as we did for WDSAT.

In Table 2 CRYPTO MINISAT and CRYPTO MINISAT denote CryptoMiniSat with and without Gaussian elimination respectively. The same notation is adopted for the WDSAT solver. These experiments were performed using WDSAT before applying the breaking symmetry technique as Table 2 serves as comparison between the different approaches and breaking symmetry is a feature that none of the other solving tools possess. The performance of the complete WDSAT solver is shown in Table 3 where we increase $l$ up to 11. With other approaches, we could not handle these larger values of $l$ in reasonable time.

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, [35] 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 generic SAT solvers, 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. [35] and [30] show experiments using the fourth summation polynomial with $l = 6$, whereas the highest parameter size achieved in [14] is $l = 8$.

Table 2 shows the average runtime in seconds, the average number of conflicts and the average memory use in MB. WDSAT 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.

<table>
<thead>
<tr>
<th>Approach</th>
<th>n</th>
<th>Runtime</th>
<th>#Conflicts</th>
<th>Memory</th>
<th>Runtime</th>
<th>#Conflicts</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Gröbner</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>207.220</td>
<td>NA</td>
<td>3601</td>
<td>142.119</td>
<td>NA</td>
<td>3291</td>
</tr>
<tr>
<td>19</td>
<td>215.187</td>
<td>NA</td>
<td>3940</td>
<td>155.765</td>
<td>NA</td>
<td>4091</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>3854.708</td>
<td>NA</td>
<td>38763</td>
<td>2650.696</td>
<td>NA</td>
<td>38408</td>
</tr>
<tr>
<td>23</td>
<td>3128.844</td>
<td>NA</td>
<td>35203</td>
<td>2286.136</td>
<td>NA</td>
<td>35162</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td>&gt;200GB</td>
<td>NA</td>
<td>&gt;200GB</td>
<td>&gt;200GB</td>
<td>NA</td>
<td>&gt;200GB</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>&gt;200GB</td>
<td>NA</td>
<td>&gt;200GB</td>
<td>&gt;200GB</td>
<td>NA</td>
<td>&gt;200GB</td>
</tr>
<tr>
<td><strong>MiniSat</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>62.702</td>
<td>408189</td>
<td>12.7</td>
<td>270.261</td>
<td>463309</td>
<td>24.2</td>
</tr>
<tr>
<td>19</td>
<td>229.055</td>
<td>1778377</td>
<td>23.6</td>
<td>388.719</td>
<td>2439933</td>
<td>29.8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>406.918</td>
<td>1919565</td>
<td>33.6</td>
<td>6777.431</td>
<td>25180492</td>
<td>105</td>
</tr>
<tr>
<td>23</td>
<td>12945.613</td>
<td>61610582</td>
<td>152</td>
<td>13260.586</td>
<td>59289671</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td>8027.974</td>
<td>63384411</td>
<td>256</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
</tr>
<tr>
<td><strong>GLUCOSE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>81.898</td>
<td>711918</td>
<td>11.9</td>
<td>119.694</td>
<td>815185</td>
<td>18.5</td>
</tr>
<tr>
<td>19</td>
<td>299.175</td>
<td>2332066</td>
<td>16.7</td>
<td>269.212</td>
<td>2077689</td>
<td>16.7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>908.091</td>
<td>5357976</td>
<td>19.7</td>
<td>1356.990</td>
<td>5884897</td>
<td>22.0</td>
</tr>
<tr>
<td>23</td>
<td>2585.200</td>
<td>12528231</td>
<td>21.8</td>
<td>3760.138</td>
<td>16898505</td>
<td>28.3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td>6755.026</td>
<td>20886673</td>
<td>31.9</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
</tr>
<tr>
<td><strong>CryptoMiniSat</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>133.983</td>
<td>775948</td>
<td>48.4</td>
<td>363.513</td>
<td>1709971</td>
<td>59.5</td>
</tr>
<tr>
<td>19</td>
<td>560.080</td>
<td>396192</td>
<td>64.1</td>
<td>1172.740</td>
<td>5726372</td>
<td>70.1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>1210.612</td>
<td>5713259</td>
<td>85.3</td>
<td>10258.351</td>
<td>26079224</td>
<td>117</td>
</tr>
<tr>
<td>23</td>
<td>3637.032</td>
<td>12159752</td>
<td>80.4</td>
<td>19857.454</td>
<td>47086152</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td>9846.554</td>
<td>18509058</td>
<td>123</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>6905.477</td>
<td>13269631</td>
<td>115</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
<td>&gt;10 hours</td>
</tr>
</tbody>
</table>
As expected, the Gröbner basis approach was outperformed by state-of-the-art SAT solvers when a solution exists. When there is no solution, it yields faster running times than solvers, but it quickly becomes impractical because of the memory requirements.

Our dedicated WDSAT solver yields significantly faster running times than any of the state-of-the-art tools. As we explained in Section 5, the number of conflicts found by WDSAT is bounded by $2^{3l}$. We observe however, that running times are slower for the WDSATXG variant. This is explained by observing that the number of conflicts is only slightly better when Gaussian elimination is used. The cost of performing a Gaussian elimination at every level of the binary search-tree outweighs the benefit of having reached less conflicts.

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 $l = 6, 11$ 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 $m!$.

We compare the cases where there is no solution, as these have more stable averages.
<table>
<thead>
<tr>
<th>l</th>
<th>1</th>
<th>n</th>
<th>Runtime</th>
<th>#Conflicts</th>
<th>Memory</th>
<th>1</th>
<th>n</th>
<th>Runtime</th>
<th>#Conflicts</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>17</td>
<td>.220</td>
<td>17792</td>
<td>1.4</td>
<td>43875</td>
<td>1.4</td>
<td>17</td>
<td>.605</td>
<td>44034</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>.243</td>
<td>19166</td>
<td>1.4</td>
<td>4034</td>
<td>1.4</td>
<td>19</td>
<td>2.205</td>
<td>351353</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>3.555</td>
<td>189940</td>
<td>1.4</td>
<td>35025</td>
<td>1.4</td>
<td>23</td>
<td>7.478</td>
<td>350257</td>
<td>1.4</td>
</tr>
<tr>
<td>7</td>
<td>23</td>
<td>29.584</td>
<td>1145966</td>
<td>17.0</td>
<td>81767</td>
<td>17.0</td>
<td>26</td>
<td>39.214</td>
<td>2803580</td>
<td>17.0</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>447</td>
<td>10557129</td>
<td>17.1</td>
<td>1048</td>
<td>17.1</td>
<td>37</td>
<td>447</td>
<td>22396994</td>
<td>17.1</td>
</tr>
<tr>
<td></td>
<td>47</td>
<td>609</td>
<td>12675174</td>
<td>17.2</td>
<td>1167</td>
<td>17.2</td>
<td>47</td>
<td>609</td>
<td>22381494</td>
<td>17.2</td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>611</td>
<td>11297325</td>
<td>17.3</td>
<td>1327</td>
<td>17.3</td>
<td>59</td>
<td>611</td>
<td>22390211</td>
<td>17.3</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>677</td>
<td>11608420</td>
<td>17.4</td>
<td>1430</td>
<td>17.4</td>
<td>67</td>
<td>677</td>
<td>22388053</td>
<td>17.4</td>
</tr>
<tr>
<td>8</td>
<td>47</td>
<td>5847</td>
<td>95131900</td>
<td>17.3</td>
<td>11963</td>
<td>17.3</td>
<td>59</td>
<td>6849</td>
<td>179091409</td>
<td>17.3</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>6530</td>
<td>88292215</td>
<td>17.4</td>
<td>14555</td>
<td>17.4</td>
<td>67</td>
<td>6530</td>
<td>179052277</td>
<td>17.4</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>7221</td>
<td>86174432</td>
<td>17.5</td>
<td>16294</td>
<td>17.5</td>
<td>79</td>
<td>7221</td>
<td>179043408</td>
<td>17.5</td>
</tr>
<tr>
<td>9</td>
<td>59</td>
<td>64162</td>
<td>727241718</td>
<td>19.2</td>
<td>135801</td>
<td>19.2</td>
<td>79</td>
<td>61370</td>
<td>1432120827</td>
<td>19.4</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>61370</td>
<td>599263451</td>
<td>19.4</td>
<td>1432191354</td>
<td>19.2</td>
<td>89</td>
<td>85834</td>
<td>1432099666</td>
<td>19.5</td>
</tr>
</tbody>
</table>

Table 3: Running times and number of conflicts using the complete WDSAT solver.

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 1300 times faster for instances allowing a solution. This is a rough comparison, as the factor grows with parameters \( l \) and \( n \).

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 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 XOR model with only \( m l \) core variables, whose assignment propagates all remaining variables in the model. Consequently, with appropriate solving methods the time complexity of the PDP is in fact \( 2^{ml} \). To this end, we conceived a SAT solver dedicated to solving systems derived from a Weil descent, named WDSAT. We further 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 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 the case of prime degree extension fields in characteristic 2. Moreover, memory is no longer a constraint for the PDP when the Gröbner basis computation is replaced with WDSAT solving.

Regarding future perspectives, we consider finding and setting an early abort threshold for our solver, since experiments suggest that PDP instances that have no solution have slower running times than instances allowing a solution. It would also be interesting to test the performance of WDSAT on the simplified system obtained by considering the action of 2-torsion and 4-torsion points on the factor base, as in [14].

References


