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

No Author Given

No Institute Given

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 choose one SAT solver and we extend it by adding a novel breaking symmetry technique. 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 chosen solver is significantly faster than current algebraic methods based on Gröbner basis computation. For the values l and n considered in the experiments, the solver 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) [24], dedicated to logarithms in $\mathbb{Z}_q$ and the algorithms of Gaudry [15] 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 [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 [37]. 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 [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 use a dedicated SAT solver for our model. Specifically, this solver is adapted for XOR-reasoning, which is one of the core operations in polynomial systems. 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 ω 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 a 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 6 we give worst time complexity estimates for solving a PDP instance and derive the complexity of our SAT-based index calculus algorithm. Finally, Section 7 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, CRYPTOMINISAT 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

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

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

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

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

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

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* [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 ω 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 $O(2^{\omega\tau+l})$. [1]

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 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

[1] In [12] the authors state the complexity is $O(2^{\omega\tau})$. In the proof of their Theorem 3, the exact time is $2^{m \log m + l + \omega\tau}$. Since l is large, we cannot ignore it here.

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

$$\begin{aligned} e_1 &= \sum_{1 \le i_1 \le m} X_{i_1}, \\ e_2 &= \sum_{1 \le i_1, i_2 \le m} X_{i_1} X_{i_2}, \\ &\dots \\ e_m &= \prod_{1 \le i \le m} X_i. \end{aligned} \tag{4}$$

We denote S'_{m+1} the polynomial obtained after symmetrizing $S_{m+1}(X_1, X_2, \dots, X_m, X_{m+1})$ in the first m variables, i.e. we have

$$S'_{m+1} \in \mathbb{F}_{2^n}[e_1, \dots, e_m, X_{m+1}].$$

In [37], 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 [37], 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 [37], we choose the vector space V to be the dimension-l subspace generated by $1, t, t^2, \dots, t^{l-1}$. Therefore we can write:

$$\begin{aligned} e_1 &= d_{1,0} + \dots + d_{1,l-1} t^{l-1} \\ e_2 &= d_{2,0} + \dots + d_{2,2l-2} t^{2l-2} \\ &\dots \\ e_m &= d_{m,0} + \dots + d_{m,m(l-1)} t^{m(l-1)} \end{aligned} \tag{5}$$

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

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

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

$$f_0 = f_1 = \dots = f_{n-1} = 0. \tag{6}$$

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 ($\vee$) of literals (e.g. $x_1 \vee \neg x_2 \vee x_3$). 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 ($\wedge$) 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-CNF form.

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 :

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 10 clauses numbered C_1 to C_{10}.

$$\begin{array}{ccc} C_1 : x_1 \vee x_2 \vee x_3 & C_2 : \neg x_3 \vee x_4 \vee \neg x_5 & C_3 : \neg x_3 \vee x_1 \vee \neg x_4 \\ C_4 : \neg x_2 \vee x_5 \vee x_6 & C_5 : \neg x_6 \vee \neg x_4 \vee x_5 & C_6 : x_1 \vee \neg x_4 \vee x_6 \\ C_7 : x_3 \vee x_4 \vee \neg x_5 & C_8 : x_1 \vee \neg x_2 \vee \neg x_4 & C_9 : x_4 \vee \neg x_5 \vee \neg x_6 \\ & C_{10} : \neg x_2 \vee x_3 \vee x_4 & \end{array}$$

Assigning the variable x_5 to 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 :

$$\begin{array}{ccc} C_1 : x_1 \vee x_2 \vee x_3 & C_2 : \neg x_3 \vee x_4 & C_3 : \neg x_3 \vee x_1 \vee \neg x_4 \\ C_4 : & C_5 : & C_6 : x_1 \vee \neg x_4 \vee x_6 \\ C_7 : x_3 \vee x_4 & C_8 : x_1 \vee \neg x_2 \vee \neg x_4 & C_9 : x_4 \vee \neg x_6 \\ & C_{10} : \neg x_2 \vee x_3 \vee x_4 & \end{array}$$

Using the same reasoning, assigning the variable x_3 to 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 :

$$\begin{array}{ccc} C_1 : & C_2 : x_4 & C_3 : x_1 \vee \neg x_4 \\ C_4 : & C_5 : & C_6 : x_1 \vee \neg x_4 \vee x_6 \\ C_7 : & C_8 : x_1 \vee \neg x_2 \vee \neg x_4 & C_9 : x_4 \vee \neg x_6 \\ & C_{10} : & \end{array}$$

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 equivalent (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 $O(c2^v)$, 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* [37], 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 $(\mathbf{x}_1, \ldots, \mathbf{x}_m) \in V^m$, 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 (4), we derive the following equations:

$$\begin{aligned} d_{1,0} &= c_{1,0} + \ldots + c_{m,0} \\ d_{1,1} &= c_{1,1} + \ldots + c_{m,1} \\ &\ldots \\ d_{m,m(l-1)} &= c_{1,l} \cdot \ldots \cdot c_{m,l}. \end{aligned} \tag{7}$$

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}. 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:

$$\begin{aligned} &\neg d_{1,0} \oplus c_{1,0} \oplus \ldots \oplus c_{m,0} \\ &\neg d_{1,1} \oplus c_{1,1} \oplus \ldots \oplus c_{m,1} \\ &\ldots \\ &\neg d_{m,m(l-1)} \oplus (c_{1,l} \wedge \ldots \wedge c_{m,l}). \end{aligned} \tag{8}$$

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 $(c_{i_1,j_1} \wedge c_{i_2,j_2} \wedge ... \wedge c_{i_k,j_k})$. 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

$$\begin{aligned} &(c' \vee \neg c_{i_1,j_1} \vee \neg c_{i_2,j_2} \vee ... \vee \neg c_{i_k,j_k}) \wedge \\ &(\neg c' \vee c_{i_1,j_1}) \wedge \\ &(\neg c' \vee c_{i_2,j_2}) \wedge \\ &\ldots \\ &(\neg c' \vee c_{i_k,j_k}) \end{aligned} \tag{9}$$

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 DPLL 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 $c' \Leftrightarrow (c_{i_1,j_1} \wedge c_{i_2,j_2} \wedge ... \wedge c_{i_k,j_k})$. 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 (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 $d' \Leftrightarrow (d_{i_1,j_1} \wedge d_{i_2,j_2} \wedge ... \wedge d_{i_k,j_k})$. 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 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 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

$$\begin{aligned} &(x_1 \vee \neg x_2 \vee \neg x_3) \wedge \\ &(\neg x_1 \vee x_2 \vee \neg x_3) \wedge \\ &(\neg x_1 \vee \neg x_2 \vee x_3) \wedge \\ &\quad (x_1 \vee x_2 \vee x_3) \end{aligned} \tag{10}$$

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 7, 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 DPLL 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 XOR model are exact, whereas those for the CNF model are averages obtained from experiments presented in Section 7.

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 give background details on the solver used for 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, the solver constructs a binary search tree by

[2] 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.

		Gröbner model		CNF model		XOR model		
l	n	#Vars	#Equations	#Vars	#CNF-clauses	#Vars	#CNF-clauses	#XOR-clauses
5	13	42	40	2474	9552	502	1505	40
	17	42	44	2941	11433	502	1505	44
6	17	51	50	4686	18237	767	2364	50
	19	51	52	5019	19577	767	2364	52
7	19	60	58	6981	27216	1101	3466	58
	23	60	62	8223	32201	1101	3466	62
8	23	69	68	11036	43210	1510	4835	68
	26[2]	69	71	12074	47374	1510	4835	71
9	37	78	88	20969	82721	2000	6495	88
	47	78	98	25456	100709	2000	6495	98
	59	78	110	31942	126702	2000	6495	110
	67	78	118	35917	142632	2000	6495	118
10	47	87	104	32866	130040	2577	8470	104
	59	87	116	40203	159437	2577	8470	116
	67	87	124	45394	180232	2577	8470	124
	79	87	136	52510	208743	2577	8470	136
11	59	96	122	49538	196434	3247	10784	122
	67	96	130	55310	219553	3247	10784	130
	79	96	142	63531	252485	3247	10784	142
	89	96	152	71556	284626	3247	10784	152

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

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.
- **XORSET module** This module performs unit propagation on the parity constraints. When all except one literal in a XOR clause is assigned, the truth value of the last literal is inferred 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.

Algorithm 1 shows a pseudo-code of the DPLL algorithm which is at the core of the solver and it works in the following way. On line 4 of Algorithm 1, 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.

Algorithm 1 Function SOLVE(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 $c_{i,j}$.
5: $(contradiction, F') \leftarrow$ ASSIGN(F, $\neg c_{i,j}$).
6: **if** $contradiction$ **then**
7:     BACKTRACK().
8: **else**
9:     **if** SOLVE(F') returns FALSE **then**
10:         BACKTRACK().
11:     **else**
12:         **return** TRUE.
13:     **end if**
14: **end if**
15: $(contradiction, F') \leftarrow$ ASSIGN(F, $c_{i,j}$).
16: **if** $contradiction$ **then**
17:     BACKTRACK().
18:     **return** FALSE.
19: **end if**
20: **return** SOLVE(F').

5 Breaking symmetry

From the symmetry of Semaev's summation polynomials we have that when $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ 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 $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ 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 solution implies adding clauses and weighing the SAT model. Instead, we decided to implement this constraint in the solver using a tree-pruning-like technique. We apply this technique on top of the recursive function SOLVE in Algorithm 1. In the function SOLVE we were trying out both FALSE and TRUE for the truth value of a chosen variable. In the breaking symmetry variation of SOLVE, denoted SOLVE_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 $\mathbf{x}_1 \leq \mathbf{x}_2 \leq \ldots \leq \mathbf{x}_m$. The new algorithm for the SOLVE_BR_SYM function is detailed in Algorithm 2 and the line numbers that distinguish it from Algorithm 1 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 1, in Algorithm 2 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, SOLVE_BR_SYM does not yield a correct answer.

Continuing with the notation from Section 3, $c_{i,j}$ corresponds to the j^{th} bit of the i^{th} **x**-vector, where $2 \leq i \leq m$ and $1 \leq j \leq l$. In Algorithm 2, we decide whether to try out the truth value of FALSE for $c_{i,j}$ 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 $c_{i,j}$ 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 2. 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 $j = 0$, 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 2 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.

6 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 5, 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

$$\frac{2^{ml}}{m!}. \tag{11}$$

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

Algorithm 2 Function SOLVE_BR_SYM(F, $compare$) : Recursive function for solving a SAT formula derived from PDP.

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

```
if all clauses and all XOR-clauses are satisfied then
    return TRUE.
end if
choose one c_{i,j}.
if j=0 then
    compare ← TRUE.
end if
if (compare is FALSE) or (i = 1) or (c_{i−1,j} is set to FALSE) then
    (contradiction, F′) ← ASSIGN(F, ¬c_{i,j}).
    if contradiction then
        BACKTRACK().
        compare ← FALSE.
    else
        if SOLVE_BR_SYM(F′, compare) returns FALSE then
            BACKTRACK().
            compare ← FALSE.
        else
            return TRUE.
        end if
    end if
end if
(contradiction, F′) ← ASSIGN(F, c_{i,j}).
if contradiction then
    BACKTRACK().
    return FALSE.
end if
return SOLVE_BR_SYM(F′, compare).
```

they each yield $d+1$ clauses (see Equation (9)). 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 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.

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

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, 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 DPLL 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 (11).

7 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 CRYPTOMINISAT and our chosen solver 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 our chosen DPLL solver.

In Table 2 CRYPTOMINISAT$_{\text{XG}}$ and CRYPTOMINISAT denote CryptoMiniSat with and without Gaussian elimination respectively. The same notation is adopted for the DPLL solver. These experiments were performed using the DPLL solver 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 DPLL 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, [37] indicates that the computational complexity of Gröbner basis is lower when a solution does not exist.

We chose parameters such that $n > lm$, as per [30]. However, we believe that choosing n much higher than ml is not beneficial for the SAT-solving method. Indeed, SAT solvers perform worse when there is no solution and increasing n results in treating a greater number of unsatisfiable instances.[3]

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. [37] 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. The DPLL 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

[3] Conclusion derived from the probability that a random point can be decomposed $\frac{2^{ml}}{m!2^n}$

			SATisfiable			UNSATisfiable		
Approach	l	n	Runtime	#Conflicts	Memory	Runtime	#Conflicts	Memory
		23	3128.844	NA	35203	2286.136	NA	35162
	8	23			>200GB			>200GB
		26			>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		
Glucose	6	17	81.898	711918	11.9	119.694	815185	18.5
		19	299.175	2332066	16.7	269.212	2077689	16.7
	7	19	908.091	5357976	19.7	1356.990	5884897	22.0
		23	2585.200	12528231	21.8	3760.138	16898505	28.3
	8	23	6755.026	20886673	31.9	>10 hours		
		26	>10 hours			>10 hours		
CryptoMiniSat	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		
CryptoMiniSat$_{XG}$	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		
DPLL	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

			SATisfiable			UNSATisfiable		
Approach	l	n	Runtime	#Conflicts	Memory	Runtime	#Conflicts	Memory
DPLL_{XG}	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[4]	3404.765	6231289	16.9	9623.677	16636122	16.9

Table 2: Comparing different approaches for solving the PDP.

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.

The DPLL solver yields significantly faster running times than any of the state-of-the-art tools. As we explained in Section 6, the number of conflicts found by DPLL is bounded by 2^{3l}. We observe however, that running times are slower for the DPLL_{XG} 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. Further research is needed to explain why Gaussian elimination does not result in a significantly lesser number of conflicts. A potential explanation can be found in Table 1, where we see that the number of variables in the XOR model is greater than the number of XOR clauses. The Gaussian elimination reasons on an underdetermined system.

Choosing the DPLL 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 DPLL solver with its symmetrical variant in Table 2, we observe a speedup factor that rapidly approaches 6.[5] This confirms our claims in Section 6 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 DPLL 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.

Lastly, we experimented with the Parallel Collision Search [36] generic method, using the open source code at [35]. This implementation solves the discrete log problem in the case of prime field curves. We did not adapt the code for extension fields

[4] 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.

[5] 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 DPLL solver. Running times are in seconds and memory use is in MB.

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 PCS 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 2. 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.

8 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 ml 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 chose to use a SAT solver dedicated to solving systems derived from a Weil descent. 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 DPLL 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 [14].

References

1. A. Casanova and Jean-Charles Faugère and Gilles Macario-Rat and Jacques Patarin and Ludovic Perret and Jocelyn Ryckeghem. GeMSS, 2017.
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. Jean-Charles Faugère, Ludovic Perret, and Jocelyn Ryckeghem. DualModeMS, 2017.
14. 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.
15. 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.
16. 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.
17. 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.
18. Ming-Deh A. Huang, Michiel Kosters, and Sze Ling Yeo. Last Fall Degree, HFE, and Weil Descent Attacks on ECDLP. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 581–600, 2015.
19. 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.
20. 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.
21. 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.
22. Tero Laitinen, Tommi A. Junttila, and Ilkka Niemelä. Conflict-Driven XOR-Clause Learning (extended version). *CoRR*, abs/1407.6571, 2014.
23. Daniele Lazard. Göbner bases, Gaussian elimination and resolution of systems of algebraic equations. In J. A. van Hulzen, editor, *Computer Algebra, EUROCAL '83, European Computer Algebra Conference, London, England, March 28-30, 1983, Proceedings*, volume 162 of *Lecture Notes in Computer Science*. Springer, 1983.
24. A. K. Lenstra, M. S. Manasse, , and J. M. Pollard. *The Number Field Sieve*, pages 11–42. Springer Berlin Heidelberg, 1993.
25. 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.
26. Jacques Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 33–48, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
27. Jacques Patarin, Nicolas Courtois, and Louis Goubin. QUARTZ, 128-Bit Long Digital Signatures. In *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, pages 282–297, 2001.
28. 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.
29. Igor A. Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. *IACR Cryptology ePrint Archive*, 2004:31, 2004.

30. 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.
31. João P. Marques Silva and Karem A. Sakallah. Conflict Analysis in Search Algorithms for Satisfiability. In *ICTAI*, pages 467–469. IEEE Computer Society, 1996.
32. Mate Soos. Enhanced Gaussian elimination in DPLL-based SAT solvers. In *In Pragmatics of SAT*, 2010.
33. 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.
34. Niklas Sörensson and Niklas Eén. A SAT Solver with Conflict-Clause Minimization. *Proc. Theory and Applications of Satisfiability Testing*, 2005.
35. Monika Trimoska, Sorina Ionica, and Gilles Dequen. PCS. https://github.com/mtrimoska/PCS, 2019.
36. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.
37. 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.