Combinatorics in algebraic and logical cryptanalysis

Monika Trimoska

▶ **To cite this version:**

Monika Trimoska. Combinatorics in algebraic and logical cryptanalysis. Other [cs.OH]. Université de Picardie Jules Verne, 2021. English. NNT : 2021AMIE0005 . tel-03624620

HAL Id: tel-03624620
https://theses.hal.science/tel-03624620

Submitted on 30 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse de Doctorat

Mention Informatique

présentée à l'École Doctorale en Sciences, Technologie, Santé (ED 585)

à l'Université de Picardie Jules Verne

par

Monika Trimoska

pour obtenir le grade de Docteur de l'Université de Picardie Jules Verne

Combinatorics in Algebraic and Logical Cryptanalysis

Soutenue le 14 janvier 2021 après avis des rapporteurs, devant le jury d'examen :

Antoine Joux, Professeur	Président
Pierrick Gaudry, Directeur de Recherche	Rapporteur
Laurent Simon, Professeur	Rapporteur
Martin R. Albrecht, Professeur	Examinateur
Laure Brisoux Devendeville, Maître de Conférences	Examinateur
Gilles Dequen, Professeur	Directeur de thèse
Sorina Ionica, Maître de Conférences	Co-encadrant

Cette thèse a été effecutée dans le cadre du projet CASSPair. Le projet CASSPair est cofinancé par l'Union européenne avec le Fonds européen de développement régional.

To my parents

Acknowledgements

The greatest thank you goes out to my supervisors Gilles Dequen and Sorina Ionica, that have worked with me on crypto problems for the past five years. This work would not have been possible without their guidance and creative ideas. They have been very mindful about letting me become an independent researcher while, at the same time, making sure my work stays in the right direction. Any success I hopefully have in the future will also be a result of the skills that they have transferred to me. Gilles, thank you, first, for giving me that initial idea and inspiring me to go into research when my dream was to be a Java architect. For having advice on matters that go beyond the science, teaching me how to sort which things are important and not to worry about those that are not. It was a pleasure to work with you and I hope that we have a lot of brainstorming sessions in front of us. Sorina, thank you for always insisting I do my best work and being an example of the academic that I wish to become. For working till late in the evening with me on submissions, for being kind and patient when explaining technical matters that I still was not very good at, and for thinking (and acting) in advance about what is best for my professional development. Your support is, and has always been, greatly appreciated.

I big thank you to all of the other professors from the CS department that have played a huge role in my development, both by their strong encouragements and by enthusiastically showing me numerous hard problems to think about. Some of them have later become my colleagues at MIS, where the day always starts with joy and laughter during our morning coffee break. On that note, a shout out to Juliette for making administration problems magically disappear. The other PhD students, who have become my friends are the reason why I have enjoyed coming to work every day. A big thanks to those that were here before me, Clement, Jordan, Romuald, and Richardson, for all their help at the beginning and throughout my thesis. I would also like to mention Fabien, Olivier, Clémence, Sébastien, and Pierre for the fun we have at the office and the way we help each other, as well as our neighbors from the PR team. I also extend my thanks to all my friends in Amiens and Skopje for being there for me and for their encouragement.

I am very grateful to the reviewers for taking the time to review my work and for asking such thoughtful questions during my defense. A big thank you to Pierrick Gaudry, for reading the manuscript with such attention to detail and for giving valuable insight on the subject of elliptic curves and index calculus attacks. Secondly, I thank Laurent Simon, for giving his honest feedback and praise that was indispensable to reassure us in our counter-intuitive findings on CDCL techniques. I should also like to extend my thanks to the examiners Antoine Joux and Martin R. Albrecht for their valuable comments, and finally, to Laure Brisoux Devendeville, who I have known for a very long time and was particularly happy to have as an examiner for this thesis. Special thanks to Antoine Joux for coming in person for my thesis defense in spite of the traveling difficulties that arise due to the current health crisis.

I thank my parents for their support, in every meaning of the word, and for always putting me first. I am lucky to have parents that are such different people and thus, to be able to inherit the best of both. My pursuit for excellence in education comes from my

mother, as well as my privilege to pursue it, since she has made sure I start off with all the skills I need, early on. My courage for taking on bigger challenges than I think I can handle comes from my father, for setting such an example and encouraging me to do so as well. This thesis is dedicated to them. I also wish to make an honorable mention to their respective spouses, and, of course, to my younger brother and sister to whom I wish all the happiness and success in the world. To my grandparents and my other grandmother, I am very grateful for being the best cheerleaders anyone can have. I often feel like they are even more excited than I am about my accomplishments.

My final thanks go to Aleksandar, to whom I owe my happiness. Your never-ending support makes life enjoyable and smooth, which in turn lets me be my best creative self. For this, and a list of reasons that can never be exhaustive, you play a huge hidden part in all of my accomplishments.

Contents

Résumé **8**

Glossary **14**

1 Introduction **16**

2 Boolean Polynomial Systems **21**
2.1 Algebraic solving techniques 22
2.1.1 Gröbner basis algorithms 22
2.1.2 Linearization and the XL family 26
2.1.3 Exhaustive search 27
2.1.4 Hybrid methods 27
2.1.5 Algorithms for sparse systems 28
2.2 Applications in cryptography 28

I SAT as a tool **30**

3 The Satisfiability problem **31**
3.1 Preliminaries 31
3.2 Solving techniques 33
3.3 Applications in cryptography 36
3.3.1 Deriving a SAT model from a Boolean polynomial system 37

4 The WDSat Solver **41**
4.1 Core algorithm 42
4.2 Three reasoning modules 44
4.2.1 CNF module 44
4.2.2 XORSET module 47
4.2.3 XORGAUSS module 49
4.3 Extending the XORGAUSS module 53
4.4 Complexity discussion 58

II Cryptographic applications **60**

5 ECDLP Preliminaries **61**
5.1 Applications in cryptography 63
5.1.1 Diffie-Hellman key exchange 63
5.1.2 ElGamal encryption 64
5.1.3 Pairing-based cryptography 65
5.2 Known attacks 66

5.2.1 Generic attacks . . . 66
5.2.2 Attacks on specific families . . . 67

6 Parallel Collision Search 69
6.1 Background . . . 70
6.1.1 Solving discrete logarithms. . . . 71
6.1.2 Many collision applications : the multi-user setting . . . 72
6.1.3 Many collision applications : meet-in-the-middle attacks . . . 74
6.1.4 Computational model and data structure. . . . 74
6.2 Time complexity . . . 75
6.2.1 Finding one collision . . . 75
6.2.2 Finding many collisions . . . 77
6.3 The multi-user setting . . . 80
6.4 Our approach for the data structure . . . 83
6.4.1 Radix tree structure . . . 83
6.4.2 Packed Radix-Tree-List . . . 85
6.5 Implementation and benchmarks . . . 87
6.5.1 PRTL implementation . . . 87
6.5.2 PRTL vs. hash table. . . . 88
6.5.3 ECDLP implementation details and scalability. . . . 91
6.5.4 Multi-collision search computation. . . . 92

7 Index Calculus 94
7.1 Background . . . 95
7.1.1 Classical index calculus . . . 95
7.1.2 Index calculus for ECDLP . . . 98
7.2 Model description . . . 100
7.2.1 The Algebraic Model . . . 101
7.2.2 The CNF-XOR Model . . . 102
7.2.3 The CNF Model . . . 103
7.3 Symmetry breaking technique . . . 104
7.4 Branching order . . . 105
7.4.1 MVC and summation polynomials . . . 108
7.5 Complexity analysis . . . 110
7.5.1 The third summation polynomial . . . 111
7.6 Experimental results . . . 112
7.6.1 Whole Point Decomposition Phase Computation . . . 117

8 Other applications of logical cryptanalysis 119
8.1 The Trivium stream cipher . . . 119
8.1.1 Model generation and experimental results . . . 120
8.1.2 Complexity discussion . . . 121
8.2 The MQ problem . . . 122
8.2.1 Generation of the MQ model . . . 123
8.2.2 Complexity analysis . . . 124
8.2.3 Experimental results and perspectives . . . 124

9 Conclusion 128
9.1 Open questions and extendibility . . . 129

Résumé

La cryptanalyse a pour objet de vérifier, par la recherche d'affaiblisssements potentiels, si un cryptosystème est suffisamment sécurisé pour être utilisé dans un contexte de développement expérimental, d'entreprise ou encore grand public. A ce titre, la cryptanalyse se concentre également sur la bonne compréhension de la complexité calculatoire d'une attaque cryptographique permettant de corréler la longueur de la clé secrète à la garantie que l'effort cryptanalytique pour la découvrir, même partiellement, ne puisse se faire en un temps raisonnable. Ainsi, les recommandations concernant les exigences de longueur de clé minimale données par diverses organisations universitaires et gouvernementales sont basées sur la complexité des attaques connues.

Les attaques que nous décrivont dans cette thèse reposent sur des approches combinatoires, relevant notamment de la théorie des graphes et de la satisfaction sous contraintes. Ainsi, nous présentons plusieurs exemples de modélisation de problème cryptographique en instances équivalentes de problèmes de satisfaction sous contraintes. Les méthodes employées pour leur résolution sont dédiées. Bien que cette approche puisse être générique, notre objectif principal concerne l'étude du problème du logarithme discret sur courbes elliptiques (ECDLP).

Définition 1 *Soit E une courbe elliptique définie sur un corps fini $\mathbb{F}_q$, avec $q = p^n$ et n un nombre premier et soit $P \in E(\mathbb{F}_q)$. Étant donné un Q multiple de P, le problème du logarithme discret sur courbes elliptiques consiste à trouver x tel que $xP = Q$.*

La sécurité des systèmes cryptographiques basé sur des courbes elliptiques est sous-tendue par la difficulté intrinsèque de ce problème. Nous étudions des attaques sur ces systèmes dans deux catégories distinctes : attaques génériques, attaques dans le cas spécifique des courbes définies sur un corps de caractéristique 2.

Recherche de collisions Pour le cas général, nous nous concentrons sur l'algorithme Parallel Collision Search (PCS), proposé par van Oorschot et Wiener [vOW99]. Cet algorithme consiste à modéliser une fonction aléatoire dont les valeurs sont des points sur la courbe elliptique. Une suite des points trouvés de cette façon est appelée chemin aléatoire. Lorsqu'un même point de la courbe est trouvé par deux chemins aléatoires distincts, le problème du logarithme discret est alors résolu pour l'instance considérée, correspondant par exemple, à une clé de chiffrement secrète. La parallélisation de cette attaque, dans le cadre du modèle à mémoire partagée, se fait de façon naturelle, où chaque thread construit, au sein de sa section parallèle, son propre chemin aléatoire.

Les contributions que nous avons proposées dans ce contexte sont les suivantes.

- *Proposer une structure de données adaptée.* Pour mener l'attaque PCS, il est nécessaire de stocker une portion des points trouvés. Afin de proposer une structure de stockage qui optimise la complexité de cet algorithme, nous avons défini les propriétés nécessaires suivantes : permettre un accès à la mémoire multi-threadé limitant les défauts de cache ainsi qu'une recherche rapide d'un point dans la structure.

Dans la littérature, ces questions ne sont pas traitées en détails. Communément, lorsque ce point est abordé, il est fait usage d'une table de hachage. Notre contribution est fondée sur le principe des "radix tree" tirant également parti des listes chainées. L'hybridation de ces deux principes offre une occupation optimale de la mémoire. Cette structure originale est nommée Packed Radix-Tree-List (PRTL).

- *Analyser la complexité de recherche de collisions.* Nous approfondissons l'étude et la compréhension de la complexité en temps de l'algorithme de van Oorschot et Wiener. Les travaux réalisés dans le cadre de l'attaque PCS sont exploitables pour d'autres approches se destinant à la recherche de collisions. A titre d'exemple on retrouve l'algorithme Meet-In-The-Middle exploitant les vulnérabilités des systèmes cryptographiques symétriques (e.g 3-DES). La résolution du problème du logarithme discret constitue un autre exemple, notamment dans le cas où un grand nombre d'utilisateurs utilisent les mêmes paramètres publiques, couramment appelé le cas multi-user [KS01; FJM14a]. Ainsi, contrairement au PCS qui nécessite une seule collision, l'attaque Meet-In-The-Middle ou l'attaque multi-user nécessite l'accès à un grand nombre de collisions. Dans ce cas, le besoin de ressources mémoire devient plus important et induit que le choix de la structure de stockage est alors déterminant. Au final, nous revisitons la preuve de la complexité en temps pour les algorithmes multi-collision et proposons l'élimination de certaines heuristiques.

  Théorème 1 *Soit S un ensemble de N élements et $f : S \to S$ une fonction aléatoire. On note θ la proportion de points distingués (des points ayant une propriété définie) dans S. Le temps d'exécution attendu pour trouver m collisions de la fonction f avec une contrainte mémoire de w mots est:*

$$\frac{1}{L}\left(\frac{w}{\theta} + (m - \frac{w^2}{2\theta^2 N})\frac{\theta N}{w} + \frac{2m}{\theta}\right).$$

Calcul d'index La méthode du calcul d'index permet d'exploiter certaines faiblesses algébriques des courbes disposant de propriétés particulières. Nos travaux se concentrent sur le cas des courbes elliptiques binaires. Au cœur de l'attaque de ces courbes réside la résolution des systèmes polynomiaux en caractéristique 2. Ainsi, la première phase du calcul d'index, phase de recherche de relations, consiste à résoudre des systèmes d'équations obtenus à partir de polynômes de Semaev, dont les zéros représentent des coordonnées de points. La résolution de ces systèmes répond au problème de décomposition de points et ainsi, elle permet de générer des relations algébriques entre certains points de la courbe. Les relations sont utilisées dans la deuxième phase dite phase d'algèbre linéaire, nous permettant de déduire le logarithme discret pour une instance donnée.

La phase de recherche de relations est abordée par la cryptanalyse algébrique. Cette approche de cryptanalyse consiste à modéliser un cryptosystème sous la forme d'équations polynomiales, amenant la sécurité de celui-ci à la difficulté de la résolution du système associé. Les attaques algébriques se déroulent en deux étapes principales. La première étape consiste en une modélisation (ou encodage), étape déterminante pour la suite de l'attaque, consistant, en seconde étape, en la résolution du système. La méthode de résolution, représentant l'attaque à proprement parlé, est adaptée et spécifique au modèle construit. Cette spécialisation de la résolution est un gage d'efficacité. Cela s'illustre par exemple par le fait qu'il existe toute une famille d'algorithmes dédiés à la résolution de systèmes polynomiaux creux, qui ne sont pas bien adaptés à la résolution de systèmes aléatoires.

La complexité de la résolution d'un système d'équations polynomiales dépend fortement du nombre de variables, noté n, du nombre d'équations, noté m et, selon les cas, du rapport entre ces deux paramètres. Une autre mesure cruciale de la difficulté est le degré maximal des polynômes. Pour les algorithmes qui utilisent les techniques reposant sur les bases de Gröbner, les estimations de complexité sont basées sur le degré de régularité.

Dans cette thèse, nous étudions le cas spécifique de systèmes d'équations polynomiales en caractéristique 2. Ces systèmes ont la particularité que les coefficients sont dans $\mathbb{F}_2$.

Exemple 1 Ce qui suit est un exemple de système d'équations booléennes en trois variables.

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

Il existe des nombreuses méthodes algébriques pour la résolution de ces systèmes. Les plus courants sont les algorithmes utilisant les techniques de bases de Gröbner [BCLA83; Buc06], notamment les algorithmes F4 [Fau99] et F5 de Faugère [Fau02]. Pour une description détaillée de ces algorithmes, le lecteur peut se reporter à [BFS15; Bar04; Spa12; Vit11; Sva14]. D'autres méthodes de résolution algébriques comprennent la méthode de linéarisation et la famille d'algorithmes XL [KS99; CKPS00], la recherche exhaustive [BCC+10] et les méthodes hybrides [YC04; YC05; BFP09; BFSS13; JV17].

Pour tout système d'équations booléennes, il existe une formule propositionnelle équivalente. Pour transformer un système polynomial en caractéristique 2 en une formule sous forme normale algébrique (ANF), il est nécessaire de remplacer toute multiplication par l'opérateur logique ET ($\wedge$), et toute addition par l'opérateur XOR ($\oplus$). Les éléments 0 et 1 dans $\mathbb{F}_2$ correspondent alors respectivement à $\bot$ et $\top$. Une formule ANF peut alors être transformée en forme normale conjonctive (CNF) par des méthodes classiques. Cette modélisation du problème sous sa forme CNF nous permet de bénéficier des progrès colossaux réalisés ces dernières années dans la résolution pratique du problème SAT. Ainsi, nous exploitons la puissance des solveurs SAT de dernière génération afin de dédier une partie des calculs fastidieux d'une cryptanalyse à un outil automatique. Ce type d'attaques, dont le formalisme utilisé relève de la logique propositionnelle, forment la base de la cryptanalyse logique, qui a été décrite pour la première fois au début des années 2000 dans les travaux de Fabio Massacci [MM00]. Cette première proposition utilise le (désormais obsolète) Data Encryption Standard (DES) pour montrer une approche de modélisation d'une attaque cryptanalytique sous la forme d'un problème SAT. Depuis, les solveurs SAT sont devenus des outils puissants, de référence, dans l'analyse des schémas cryptographiques symétriques. Ils ont été utilisés avec succès pour attaquer des cryptosystèmes à clé secrète tels que Crypto1, Bivium, Trivium, Grain, HiTag2 [MCP07; LJN14; HJ12a; SNC09; Soo10a]. Leur utilisation dans le cadre des cryptosystèmes à clé publique a cependant rarement été envisagée. Les travaux de Galbraith et Gebregiyorgis [GG14], explorant la possibilité de remplacer les méthodes de bases de Gröbner avec des solveurs SAT génériques dans l'attaque de calcul d'index sur les courbes elliptiques binaires, en sont un exemple. Les solveurs CDCL, tels que MiniSat [SE05] et Glucose [AS09], se sont révélés être des outils puissants pour résoudre les formules CNF. Ils ne sont toutefois pas adaptés pour gérer les clauses XOR et supposent donc que de telles contraintes de parité soient traduites sous forme CNF. Cela induit à la fois lors de la modélisation et lors de la gestion des clauses résultantes une lourdeur de traitement pénalisant la résolution. Ainsi, des travaux récents se sont concentrés sur

le couplage de solveurs CDCL avec un module de raisonnement XOR. Ces techniques peuvent, de plus, être améliorées par la méthode d'élimination gaussienne, comme dans les travaux de Soos *et al.* (dont le résultat est le solveur CryptoMiniSat) [SNC09; Soo10a], Han et Jiang [HJ12a], Laitinen *et al.* [LJN14; LJN11]. Les solveurs SAT adaptés à la résolution XOR utilisent généralement des formules sous une forme spécifique, qui est une conjonction de clauses OR et de clauses XOR. On parle alors d'une forme CNF-XOR.

Dans le cadre de l'attaque de calcul d'index, premièrement, nous avons modélisé le problème de décomposition de points sous la forme d'une formule logique et nous l'avons défini comme une instance du problème SAT. En ajout de cela, nous avons développé un solveur SAT dédié à ce problème spécifique, que nous avons nommé WDSat. Au final, nous comparons expérimentalement les performances de résolution des instances SAT issues de l'attaque de calcul d'index avec le solveur WDSat, avec d'autres solveurs SAT génériques de la littérature mais aussi avec la méthode des bases de Gröbner.

Le solveur WDSat Notre solveur est basé sur l'algorithme Davis-Putnam-Logemann-Loveland (DPLL) [DLL62]. L'algorithme DPLL procède, grâce au développement d'un arbre binaire de recherche, à l'énumération implicite de l'ensemble des modèles résultant de l'affectation des variables de l'instance aux différentes valeurs de vérité. Dans le pire des cas, la hauteur de cet arbre est donc égale au nombre de variables présentes dans la formule considérée. Au cours de la résolution, et après chaque affectation de variable, la formule est simplifiée par un processus de raisonnement automatique nommé propagation unitaire. Lorsqu'un conflit survient, une procédure de *backtracking* est lancée et la valeur de vérité opposée est affectée au littéral. Si la valeur de vérité opposée entraîne un conflit également, nous revenons à une supposition antérieure ou nous concluons que la formule n'est pas satisfaisable.

WDSat est construit autour de 3 modules de raisonnement. Le premier module raisonne sur la partie de la formule étant sous forme CNF. Ce module applique des méthodes de propagation unitaire classiques. Le deuxième module, nommé XORSET, applique le raisonnement de parité sur la partie XOR. Finalement, le troisième module, nommé XORGAUSS, applique une élimination gaussienne, également sur la partie XOR. Par contre, nous avons identifié que dans certains cas, l'élimination gaussienne sur les instances SAT ne fonctionne pas aussi bien que l'élimination gaussienne sur les systèmes algébriques. Plus précisément, nous avons mis en évidence une propriété d'annulation qui est présente dans les méthodes de résolution algébriques mais qui n'est pas appliquée dans les implémentations actuelles d'élimination gaussienne au sein des solveurs SAT. Pour corriger cet oubli dans notre solveur, nous avons développé une extension du module XORGAUSS, nommé XG-EXT. A ce jour, WDSat a été utilisé dans le cadre de trois attaques cryptographiques différentes.

1. **Calcul d'index.** Notre solveur WDSat a été initialement conçu pour résoudre des instances dérivées du problème de décomposition de points. Par conséquent, il comprend deux extensions qui, à ce jour, ont donné de meilleurs temps d'exécution uniquement pour les instances dérivées de ce problème. La première contribution vise à éliminer les solutions symétriques sans agrandir le modèle SAT et sans introduire de coût de calcul supplémentaire. La seconde contribution sur ce sujet est centrée autour d'une technique originale de prétraitement basée sur le problème de couverture minimum par sommets (ou problème du transverse minimum) en théorie des graphes. Plus précisément, cette technique est utilisée avant la phase de résolution. Par conséquent, elle peut être considérée comme un prétraitement qui détermine, de manière statique, l'ordre des variables de branchement. Pour les

instances ANF, nous considérons qu'un ordre optimal de variables de branchement est celui qui conduira le plus rapidement possible à un système polynomial linéaire.

La complexité dans le pire des cas de notre solveur, pour résoudre l'étape de décomposition de points de l'attaque de calcul d'index dans $\mathbb{F}_{2^n}$, en utilisant le $(m+1)^{\text{ième}}$ polynôme de Semaev et une base de factorisation définie par un espace vectoriel de dimension l, est

$$O(\frac{2^{ml}}{m!}),$$

dans le cas où $m \geq 3$. Dans le cas où $m = 2$, la complexité est donnée par $O(2^l)$.

Pour étayer nos affirmations, nous avons conduit des expériences utilisant les courbes elliptiques de Koblitz [Kob92a] définies sur $\mathbb{F}_{2^n}$, où n est un nombre premier. Les temps d'exécution expérimentaux montrent que notre approche de résolution utilisant WDSAT est significativement plus rapide que les méthodes algébriques actuelles basées sur le calcul de bases de Gröbner. De plus, notre solveur surpasse d'autres solveurs SAT couramment utilisés, pour ce problème spécifique.

2. **Trivium.** WDSAT a été utilisé pour effectuer une attaque logique sur le système de chiffrement de flux Trivium [DC06]. L'attaque de ce système peut être réduite à la résolution d'un système d'équations algébriques booléennes. Les résultats de nos travaux expérimentaux montrent que la version XG-EXT couplée à notre technique de prétraitement (MVC) améliore les performances de WDSAT pour les instances dérivées du chiffrement Trivium. Cependant, WDSAT est dépassé par CRYPTOMINISAT pour la résolution de ces instances, à la fois en termes de temps d'exécution et de nombre de conflits. À notre avis, cette différence est due au fait que l'approche CDCL sur laquelle s'appuie CRYPTOMINISAT est mieux adaptée pour ces instances que l'approche DPLL de base utilisée par WDSAT. En particulier, l'algorithme de backtracking avec apprentissage de clauses implique des techniques de saut arrière, ce qui entraîne moins de conflits.

3. **Le problème MQ.** Les cryptosystèmes à clé publique multivariée (MPKC) sont des cryptosystèmes dont les clés publiques sont un ensemble de polynômes multivariés et leur sécurité est généralement basée sur la difficulté de résoudre un système d'équations polynomiales de degré supérieur à 1. Un système quadratique multivarié (MQ) est un système d'équations polynomiales multivariées qui se compose uniquement de polynômes quadratiques. Ce cas spécifique de système est le plus couramment utilisé dans la cryptographie multivariée. Le problème MQ appartient à la classe NP et il est considéré difficile (en moyenne) même pour les ordinateurs quantiques. Par conséquent, MQ est le problème sous-jacent de l'une des grandes familles de candidats NIST pour le processus de standardisation de la cryptographie post-quantique en cours [Nat].

   Pour construire des systèmes MQ, nous avons pris comme référence le challenge MQ initié en [YDH+15]. Nous avons expérimenté avec des paramètres m et n différents. Tout d'abord, nous avons observé que la meilleure version de WDSAT est celle qui utilise la technique XG-EXT. Ensuite, nous avons pu constater que CRYPTOMINISAT donne de meilleurs résultats lorsque l'élimination gaussienne est activée, cependant, WDSAT dépasse CRYPTOMINISAT pour ces instances. Le détenteur actuel du record pour ce type d'instances du challenge MQ est l'algorithme hybride de Joux et Vitse [JV17]. Nous n'avons pas implémenté cet algorithme pour avoir des résultats

expérimentaux comparatifs, mais selon les temps d'exécution annoncés dans [Tak], notre solveur serait dépassé par cet algorithme.

Nous espérons que le matériel présenté dans cette thèse encouragera la communauté cryptographique à utiliser les solveurs SAT pour la cryptanalyse sur $\mathbb{F}_2$. Pour des corps plus grands, d'autres techniques de programmation par contraintes peuvent être envisagées.

Glossary

AND The AND bitwise binary operator.

ANF Algebraic Normal Form.

CDCL The Conflict-driven clause learning algorithm.

CMS The CryptoMiniSat SAT solver.

CNF Conjunctive Normal Form.

DH Diffie-Hellman.

DLP Discrete logarithm problem.

DPLL The Davis-Putnam-Logemann-Loveland algorithm.

EC Equivalence class.

ECDH Elliptic curve Diffie-Hellman.

ECDHE Elliptic curve Diffie-Hellman ephemeral.

ECDL Elliptic curve discrete logarithm.

ECDLP Elliptic curve discrete logarithm problem.

GE Gaussian elimination.

Glucose The Glucose SAT solver.

MQ The Multivariate Quadratic polynomial problem.

MVC Minimal Vertex Cover.

Magma The Magma computational algebra system.

MiniSat The MiniSat SAT solver.

OR The OR bitwise binary operator.

PCS The Parallel Collision Search algorithm.

PDP Point Decomposition Problem.

PRTL The Packed Radix-Tree-List structure proposed in this thesis.

SAT The satisfiability problem

UNSAT Unsatisfiable.

WDSat The name of the SAT solver presented in this thesis. Stands for Weil Descent SAT solving.

XG An abbreviation for the XORGAUSS module.

XG-ext An extention of the XORGAUSS module of the WDSAT solver.

XOR The Exclusive OR bitwise binary operator.

XORGAUSS The XORGAUSS module of the WDSAT solver.

XORSET The XORSET module of the WDSAT solver.

Chapter 1

Introduction

Solving hard problems has always been at the center of scientific research. However, categorizing a problem as *hard* is very broad and can have a different meaning depending on the topic of interest. In the field of computational complexity theory, the hardness of a problem is assessed by the computational cost associated with its resolution, in terms of resources such as time and memory. Time (resp. memory) complexity quantifies the amount of time (resp. memory) required to run the algorithm, with respect to the size of the input. To put this into perspective, we can look at some examples of practical problems. The problem of finding an occurrence of an element in a list has linear complexity, as in the worst case, the algorithm has to traverse the list from beginning to end to answer with certainty if the element is present or not. However, when the input list is sorted, the so-called *dichotomic* search can solve this problem in logarithmic time. For other problems, one traversal of the input is not enough. For example, to delete all duplicates from a list, the list has to be traversed once for each element, which gives us an algorithm with quadratic complexity. All of these problems fall into the *polynomial* time complexity class (P) and as such, they are considered to be practical (*easy*) problems. The time complexity of algorithms in the P class is noted $O(n^c)$, where n is the size of the input and c is a constant.

Outside of the P class, we have, for example, the brute force algorithm for breaking a cryptographic key. This straightforward solution consists in testing all possible combinations and has a complexity of $O(2^n)$, where n is the bit-size of the key. Thus, the running time of this algorithm grows twice for every bit added to the input size. We consider this algorithm to be in the *exponential* time complexity class (EXP or EXPTIME).

Table 1.1 shows a comparison of how the running time grows with the size of the input n for all examples mentioned thus far. For simplicity, we consider that treating one unit of the input takes 0.01 seconds. This exercise is a textbook practice that makes understanding the impact of exponential growth easier. Now, let us imagine that the 0.01

Table 1.1: Comparing P class and EXP class complexities.

	$log_2 n$	n	n^2	2^n
$n = 2$	0.01	0.02	0.04	0.04
$n = 10$	0.03	0.1	1	10.24
$n = 20$	0.06	0.2	4	10485
$n = 45$	0.13	0.45	20.25	$351843 \cdot 10^6$

value is actually the thickness of a piece of paper in millimeters and that this piece of paper is infinitely big and can be folded an infinite number of times. The last row, holding $n = 45$, is actually the number of times that the paper needs to be folded to (almost) cover the distance between the earth and the moon. The human mind has a hard time grasping how big exponential growth is, which is what makes this and other exponential growth examples so popular.

The example of a brute force algorithm for finding a cryptographic key is actually in a subclass of exponential algorithms, called the nondeterministic polynomial time (NP) class. For problems in this class, verifying if a given output is a correct solution can be done in polynomial time. However, we do not know of a polynomial-time algorithm for finding a solution or deciding if a solution exists. We talk about a problem being *solvable* or *decidable* interchangeably, as we can show that one implies the other and vise versa. The implication from solvable to decidable is pretty straightforward. If a polynomial-time algorithm to find a solution exists, then after execution, this algorithm can answer whether the solution was found or not. However, the inverse is also true. If there exists an oracle that can decide, in polynomial time, if a given instance of the problem admits a solution, then this oracle can also be used to find it. The technique goes as follows. First, we ask the oracle if a solution exists. If the answer is positive, then one entry from the solution is chosen arbitrarily. We can assess whether the choice that was made is correct by asking the oracle if a solution still exists. If we are given a negative answer, we will have to try another guess. In the worst case, we would have to check all possibilities for one single entry. This process is repeated n times, as n is the size of the output solution. In any case, the number of queries to the oracle is a polynomial function, thus this algorithm for finding a solution has polynomial time complexity. The name of the NP class comes from the more formal definition which states that all problems in this class can be solved in polynomial time using a *nondeterministic* Turing machine.

The Boolean satisfiability problem, abbreviated as SAT, is an eminent problem in the NP class. SAT is the problem of determining whether there exists an interpretation that satisfies a given propositional formula. Historically, this is the first problem to be proven to be the hardest problem in NP. A problem A is considered to be harder than B, if we can use a solution for problem A to solve problem B. In this case, we say that problem B *reduces* to problem A. According to the Cook-Levin Theorem [Coo71; Lev73] from the early 1970s, all problems in NP can be reduced to the SAT problem and the transformation can be completed in polynomial time. As a result, SAT is said to be an NP-complete problem. Later, 21 more problems were proven to be NP-complete [Kar72] and nowadays there are hundreds. To prove that a problem in NP is NP-complete, it is enough to prove that any one of the members in the NP-complete class can be reduced to this problem.

To complete our summary of basic complexity classes, we mention the NP-hard class, a class of problems that are at least as hard as the hardest problems in NP. Problems in this class are not necessarily decision problems and those that are, may or may not verify a solution in polynomial time. A prominent example in this class is the halting problem, which is the problem of deciding whether an arbitrary computer program with a given input will eventually finish running or will run infinitely.

Polynomial-time decision and solving algorithms for NP problems have never been found, however the nonexistence of these algorithms remains unproven. This raises the question of whether all problems in NP are also in P, which is one of the most important open questions in theoretical computer science. The importance of this question is enhanced by the consequences that proving P=NP would have on today's society. On a positive note, we would, for instance, be able to efficiently solve the vehicle routing

problem or optimal scheduling and thus, save a significant amount of resources. Important advances in biotechnology and the science of DNA would be made as well, as most problems in protein structure prediction are NP. However, P=NP would also invalidate most of cryptography, as various NP problems are fundamental to constructing existing cryptosystems. Solving the underlying NP problem is equivalent to breaking the cryptosystem or violating some of its core principles: encryption, authentication, integrity, and nonrepudiation. Consequently, these problems are the main focus of cryptanalysis.

Academic research in cryptanalysis is focused on deciding whether a cryptosystem is secure enough to be used in the real world. In addition, a good understanding of the complexity of a cryptographic attack allows us to determine the secret key length, making sure that no cryptanalytic effort can find the key in a feasible amount of time. Recommendations for minimum key length requirements given by various academic and governmental organizations [Blu18] are based on the complexity of known attacks. However, for cryptographic purposes, we have to deviate from the classical notion of complexity that focuses on the worst case and turn our attention to problems that are hard on average. Indeed, there exist problems that are hard in the worst case, but that admit easily solvable instances. For example, the 3-coloring problem in graph theory is in the NP-complete class, but becomes polynomial for graphs with specific characteristics, such as dense graphs, perfect graphs [GLS84], AT-free graphs [Sta10] and others. The security of public-key cryptosystems can be threatened by this phenomenon, as was the case with one of the earliest public-key cryptosystems, the Merkle–Hellman knapsack cryptosystem [MH06]. This cryptosystem is based on the subset sum problem, another problem that is known to be NP-complete but has a significant subset of polynomial instances. Consequently, it was broken by Shamir [Sha83], a few years after its publication. Today, it is well known that to be able to construct a public-key cryptosystem, we need to find a problem for which we are able to generate hard instances in polynomial time. Also, the generation of an instance needs to come with a corresponding solution. To find such problems in cryptography, we turn to one-way functions. A function f is considered to be a one-way function if it is easy to compute, but the problem of inverting f is hard. Such functions are the primal focus of many cryptanalytic efforts.

The work presented in this thesis focuses on the cryptanalysis of several cryptographic schemes by attacking their underlying problem. The greatest portion of our work is directed at the elliptic curve discrete logarithm problem (ECDLP). This problem[1] is of great interest, not only because it represents a one-way function, but also because of its *random self-reductibility* property [AFK87]. According to this property, every instance of this problem can be transformed into a purely random instance. This serves to prove that, if there exist hard instances of the ECDLP, the problem is practically hard for random instances. The ECDLP is defined as follows.

Definition 1.0.1 *Let E be an elliptic curve over a finite field $\mathbb{F}_q$, where $q = p^n$ and p is prime and let $P \in E(\mathbb{F}_q)$. Given a multiple Q of P, the elliptic curve discrete logarithm problem is to find x such that $xP = Q$.*

We study both the general-group case and the case of elliptic curves over binary extension fields. In the family of generic attacks, our focus is on the Parallel Collision Search algorithm (PCS), first introduced by van Oorschot and Wiener [vOW99] and based on Pollard's rho algorithm [Pol78]. The complexity of this attack is in the square root of the size of the field, or more precisely in the square root of its largest prime factor [PH78]. To date, there are no attacks with a better complexity for the general case. However, when there is more information on the group structure, this information can be exploited to

[1] Also, the DLP problem in general.

develop a more efficient attack. A prominent example is the index calculus attack which constitutes our second line of our research. Historically, the index calculus attack was conceived for integer factorization. Its greatest success, however, comes from its efficiency in solving the discrete log in the multiplicative group of finite fields. The subexponential complexity of the index calculus attack for these groups triggered the use of elliptic curves in cryptography. The switch to elliptic curves for constructing secure cryptosystems, first suggested by Miller [Mil85] and Koblitz [Kob87] in 1985, became inevitable due to the need for exceedingly large key sizes when multiplicative groups of finite fields are used. Ever since, there have been many efforts to adapt the index calculus attack for the ECDLP. The first contribution that made this line of research possible was Semaev's solution for decomposing a point on an elliptic curve [Sem04]. Later, Gaudry [Gau09] and Diem [Die11] independently proposed a way to put Semaev's work into practice and develop an index calculus algorithm for elliptic curves over extension fields. The Gaudry-Diem algorithm has a better asymptotic complexity than generic methods for solving the DLP , using the $(m+1)^{\text{th}}$ summation polynomial, for the case of elliptic curves over $\mathbb{F}_{q^n}$, when q is large, $n \geq 3$ is small and $\log q > cm$ for some constant c. In our work, we are particularly interested in the special case of elliptic curves over $\mathbb{F}_{2^n}$, where n is prime.

We extended our research outside of the realm of the DLP, by analyzing the security of two other cryptosystems built upon finite fields of characteristic 2: Trivium and multivariate public key cryptosystems (MPKC) over $\mathbb{F}_2$. Trivium is a stream cipher that uses keystream generators comprised of Nonlinear-Feedback Shift Registers (NLFSR). It was designed in 2005 by C. De Cannière and B. Preneel [DC06] as a submission for the eSTREAM project and is to this day a common target of cryptanalytic attacks. MPKCs are cryptosystems whose public keys are a set of multivariate polynomials and their security is commonly based on the multivariate quadratic polynomial (MQ) problem defined as follows.

Definition 1.0.2 *Given m multivariate quadratic polynomials $f_1, \ldots, f_m$ of n variables over a finite field $\mathbb{F}$, find a tuple $\boldsymbol{w} = (w_1, \ldots, w_n)$ in $\mathbb{F}^n$, such that $f_1(\boldsymbol{w}) = \cdots = f_m(\boldsymbol{w}) = 0$.*

This problem is in the NP class and it is believed to be hard (on average) even on quantum computers. Consequently, MQ is the underlying problem of one of the families of NIST candidates for the ongoing Post-Quantum Cryptography Standardization Process [Nat]. Even though most of the MQ-based encryption constructions have been broken, some digital signature schemes remain robust. For instance, Rainbow [DS05] is one of three Round 3 finalists in this category.

What unifies most problems studied in this thesis is that their resolution can be reduced to the problem of solving a nonlinear multivariate Boolean polynomial system. This approach is called algebraic cryptanalysis and is usually tackled by using Gröbner basis algorithms [Fau99], exhaustive search [BCC+13] or hybrid methods [BFP09] for solving. However, for multivariate polynomial systems over finite fields of characteristic 2, classic algebraic methods can be replaced by SAT solving techniques. This approach, first introduced by Massacci in 2000 [MM00], is commonly known as logical cryptanalysis and is at the core of our cryptanalysis efforts. One could say that even though we initially presented SAT as a problem, in our work SAT it is more of a tool for solving other problems. In other words, we detect cryptographic problems whose resolution can be aided by SAT solvers and then we look for an efficient modelization and devise SAT solving techniques adapted for the problem at hand. To this end, we first developed a built-from-scratch SAT solver dedicated to solving the *point decomposition* step of the index calculus attack. Then, our solver was adapted for and tested against Boolean polynomial systems derived from other cryptographic problems.

When we consider the underlying cryptosystem as a black-box, the worst-case complexity of our solver is exponential in the number of variables in the Boolean polynomial system from which the SAT model was derived. This is a significant difference compared to other complete SAT solvers whose worst-case complexity is exponential in the number of variables in the SAT model itself. Looking at the approach for deriving a SAT model (detailed in Chapter 3), it is evident that the number of variables in the Boolean polynomial system is always lesser than or equal to the number of variables in the SAT model derivied from that system[2]. We also propose a preprocessing technique that gives a finer analysis of the complexity even in the case where we have no information about the underlying cryptosystem. On the other hand, when we know the structure of the underlying cryptosystem, we can determine a more precise bound on the worst-case complexity. Notably, we have the example of solving the point decomposition step of the index calculus attack in $\mathbb{F}_{2^n}$ using the $(m+1)^{\text{th}}$ summation polynomial and a factor base defined by a vector space of dimension l. The worst-case complexity of our solver for this problem is

$$O(\frac{2^{ml}}{m!}),$$

when $m \geq 3$ and for the case of $m = 2$, we have

$$O(2^l).$$

We observe experimentally that the average-case complexity deviates only by a factor of two and only in the case of systems that have a solution. For these attacks, our SAT solver outperforms all other approaches presented in this thesis, including the best currently available implementation of Gröbner bases (F4 [Fau99] in MAGMA [BCP97]) and state-of-the-art SAT solvers: MINISAT [ES04], GLUCOSE [AS09], MAPLELCMDISTCHRONOBT [NR18], CADICAL [Bie] and CRYPTOMINISAT [SNC09].

In our work on the generic PCS algorithm, we explore both one-collision (i.e. discrete logs) and multi-collision applications, such as meet-in-middle attacks the discrete log problem in the multi-user setting. Our contributions include:

- presenting a formula for the expected runtime to find any given number of collisions, with and without a memory constraint,
- proposing a simple storage structure, inspired by radix trees, which saves space and provides fast look-up and insertion,
- providing benchmarks that show the linear parallel performance of the attack on elliptic curves discrete logarithms,
- improved running times for multi-collision applications with a memory constraint.

[2]We have equality only in the case where the Boolean polynomial system is linear.

Chapter 2

Boolean Polynomial Systems

Polynomial systems of equations over finite fields have many interesting applications in cryptography, as well as in other fields. An important application is their use in Multivariate Cryptography (MC). A core principal for this application is that evaluating a polynomial at a vector is easy, whereas solving a system of polynomial equations is hard, resulting in a one-way function. The second application that is significant for our work is their use in algebraic cryptanalysis. As previously stated, algebraic cryptanalysis denotes any technique which reduces a cryptographic attack to the problem of solving a multivariate polynomial system. Algebraic attacks proceed in two main steps. First, there is the modeling, or encoding step. This step is crucial for the success of the attack and strongly influences the next step, which is the solving process. Indeed, choosing the solving techniques based on the particularity of the model results in better attacks. For instance, there is a whole family of algorithms dedicated to solving sparse polynomial systems, that are not well suited for arbitrary systems.

The hardness of solving a multivariate polynomial system of equations strongly depends on the number of variables, denoted n, the number of equations, denoted m and the ratio between these two parameters. Another crucial measure of difficulty is the maximum degree of polynomial. For algorithms that use Gröbner basis techniques, complexity estimates are based on the degree of regularity, which is a notion that will be defined later.

This chapter is dedicated to the special case of a multivariate Boolean polynomial system, which, recall, is a system of polynomials in several variables and whose coefficients are in $\mathbb{F}_2$. The following example shows a Boolean polynomial system of three equations in the variables $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$:

$$\begin{aligned} &\mathbf{x}_1 + \mathbf{x}_2 \cdot \mathbf{x}_3 = 0 \\ &\mathbf{x}_1 \cdot \mathbf{x}_2 + \mathbf{x}_2 + \mathbf{x}_3 = 0 \\ &\mathbf{x}_1 + \mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 = 0. \end{aligned} \tag{2.1}$$

Our research efforts go towards the goal of fully replacing the solving step of an algebraic cryptanalysis attack by a SAT solver. In this chapter, first, we present state-of-the-art algebraic solving techniques. Gröbner basis algorithms are described in more detail than other approaches, as they are to be compared to our SAT-based approach. Secondly, we present most notable applications of algebraic attacks in cryptography.

Notation. For simplicity, in the remainder of this dissertation we will omit the multiplication operator $\cdot$ whenever its use in monomials is implicit.

2.1 Algebraic solving techniques

2.1.1 Gröbner basis algorithms

Gröbner bases are extensively used in cryptanalysis as they are a powerful tool for solving multivariate systems of polynomial equations. Finding a solution of a polynomial system of equations using Gröbner bases is done in two steps. First, the reduced Gröbner basis is found, then the solution is extracted from the set of polynomials in the Gröbner basis. There exist many other applications for Gröbner basis algorithms, however, as our interest lies in polynomial-system solving, we will present them from this point of view. In this section, we give an elementary introduction to the notion of Gröbner basis. For further details the reader can refer to [CLO07].

Definition 2.1.1 *Let K be a field and let $K[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ be the ring of polynomials in n variables over K. An ideal I, generated by the set of polynomials $\{f_1, \ldots, f_s\}$, is a subset of $K[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$, such that*

$$I = \{\sum_{i=1}^{s} g_i f_i \mid g_i \in K[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n].\}$$

Suppose that our objective is to determine whether the system of equations $f_1 = f_2 = \cdots = f_s = 0$ has a solution over the algebraic closure $\bar{K}$. Usually, we also need to find the solutions, if there are any. The basic idea behind the Gröbner basis approach is that the original generating set of the ideal I can be replaced by a different generating set that is either simpler of has some properties that allow us to extract the solution more easily. The new generating set with convenient properties is sometimes referred to as a *basis* of the ideal. Two classical techniques for solving a polynomial system of equations that use this reasoning are the Gaussian elimination for the case of linear polynomials, and Euclid's algorithm for the case of polynomials in one variable.

Example 2.1.1 Let us consider the following linear polynomials in $\mathbb{R}[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3]$

$$\begin{aligned} f_1 &= \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 - 8 \\ f_2 &= 2\mathbf{x}_1 + 3\mathbf{x}_2 - \mathbf{x}_3 - 17 \\ f_3 &= -\mathbf{x}_2 + 5\mathbf{x}_3 - 1, \end{aligned}$$

and the associated matrix

$$\begin{pmatrix} 1 & 1 & 1 & -8 \\ 2 & 3 & -1 & -17 \\ 0 & -1 & 5 & -1 \end{pmatrix}.$$

The most efficient technique for solving the system of equations $f_1 = f_2 = f_3 = 0$ is to use Gaussian elimination to obtain the reduced row echelon form of this matrix:

$$\begin{pmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & -4 \\ 0 & 0 & 1 & -1 \end{pmatrix}.$$

The new basis of the ideal $I =< f_1, f_2, f_3 >$ that we obtained through Gaussian elimination is:

$$f_1' = \mathbf{x}_1 - 3$$
$$f_2' = \mathbf{x}_2 - 4$$
$$f_3' = \mathbf{x}_3 - 1.$$

This new set of polynomials allows us to easily compute the solution.

Example 2.1.2 Let us consider the following polynomials in one variable:

$$f_1 = \mathbf{x}^3 - \mathbf{x}^2 - 4\mathbf{x} - 6$$
$$f_2 = \mathbf{x}^2 - 2\mathbf{x} - 3.$$

This is an example of the specific case of polynomials in one variable, and thus, we can use Euclid's algorithm to find the solution to the system of equations $f_1(\mathbf{x}) = f_2(\mathbf{x}) = 0$. Using Euclid's algorithm, we find that the greatest common divisor of f_1 and f_2 is $f' = \mathbf{x} - 1$. More specifically, we have that $f_1 = (\mathbf{x}^2 + 2\mathbf{x} + 2)f'$ and $f_2 = (\mathbf{x} + 1)f'$. Since f' divides both f_1 and f_2, we conclude that another generating set of the ideal $I =< f_1, f_2 >$ is $\{f'\}$. Computing the solution to $f'(\mathbf{x}) = 0$ is trivial.

These two examples show techniques for finding a basis that can be applied respectively to the special cases of linear polynomials and polynomials in one variable. The idea behind Gröbner basis algorithms is to combine key features of Gaussian elimination and Euclid's algorithm that can be applied to the general case. For a reader that is accustomed to linear algebra, it might be interesting to (informally) draw parallels between linear and nonlinear polynomial systems. From Definition 2.1.1, we can see that an ideal is the set of all possible linear combinations of polynomials f_i, with polynomials $g_i \in K[\mathbf{x}_1, \ldots, \mathbf{x}_n]$ as coefficients. Hence, the notion of an ideal is analogous to the notion of a vector space from linear algebra. The set of solutions to a linear system is a linear subspace of K^n, whereas the set of solutions to a nonlinear system is a geometric object called an *algebraic variety* of K^n. The algorithm that transforms a linear system of equations into an upper triangular system, will be replaced by an algorithm that finds a Gröbner basis of an ideal. The analogue of Euclidean division is *reduction*, but to define this, as well as further notions, we first need to define *monomial ordering*.

Definition 2.1.2 *A monomial ordering $\prec$ is a relation on the set of monomials in $K[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$, such that*

- *The relation $\prec$ is a total ordering;*
- *The relation is compatible with the product, i.e. for a triplet of monomials m_1, m_2, m_3, if $m_1 \prec m_2$, then $m_1 m_3 \prec m_2 m_3$;*
- *Every nonempty subset of monomials has a smallest element;*

We define three of the various monomial orderings that are of interest in computational algebra. Let $\mathbf{x}^\alpha = \mathbf{x}_1^{\alpha_1} \ldots \mathbf{x}_n^{\alpha_n}$ denote a monomial and let $|\alpha| = \alpha_1 + \cdots + \alpha_n$ denote the monomial degree.

Lexicographic. The lexicographic (*lex*) order is similar to the usual alphabetical order. We have that $\mathbf{x}^\alpha \prec_{lex} \mathbf{x}^\beta$ if the left-most nonzero entry of $\alpha - \beta$ is strictly negative.

Example 2.1.3 The *lex* order of all degree 2 monomials in the variables $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ is

$$1 \prec \mathbf{x}_3 \prec \mathbf{x}_3^2 \prec \mathbf{x}_2 \prec \mathbf{x}_2\mathbf{x}_3 \prec \mathbf{x}_2^2 \prec \mathbf{x}_1 \prec \mathbf{x}_1\mathbf{x}_3 \prec \mathbf{x}_1\mathbf{x}_2 \prec \mathbf{x}_1^2.$$

Graded lexicographic. With the graded lexicographic (*glex*) order, we first sort by the total degree and if there is an equality, the lexicographic order applies, i.e. $\mathbf{x}^\alpha \prec_{glex} \mathbf{x}^\beta$ if $|\alpha| < |\beta|$ or $|\alpha| = |\beta|$ and $\mathbf{x}^\alpha \prec_{lex} \mathbf{x}^\beta$.

Example 2.1.4 The *glex* order of all degree 2 monomials in the variables $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ is

$$1 \prec \mathbf{x}_3 \prec \mathbf{x}_2 \prec \mathbf{x}_1 \prec \mathbf{x}_3^2 \prec \mathbf{x}_2\mathbf{x}_3 \prec \mathbf{x}_2^2 \prec \mathbf{x}_1\mathbf{x}_3 \prec\prec \mathbf{x}_1\mathbf{x}_2 \prec \mathbf{x}_1^2.$$

Graded reverse lexicographic. With the graded reverse lexicographic (*grevlex*) order, we first sort by the total degree and if there is an equality, we use a reversed lexicographic order, i.e. $\mathbf{x}^\alpha \prec_{grevlex} \mathbf{x}^\beta$ if $|\alpha| < |\beta|$ or $|\alpha| = |\beta|$ and right-most nonzero entry of $\alpha - \beta$ is strictly positive.

Example 2.1.5 The *grevlex* order of all degree 2 monomials in the variables $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ is

$$1 \prec \mathbf{x}_3 \prec \mathbf{x}_2 \prec \mathbf{x}_1 \prec \mathbf{x}_3^2 \prec \mathbf{x}_2\mathbf{x}_3 \prec \mathbf{x}_1\mathbf{x}_3 \prec \mathbf{x}_2^2 \prec \mathbf{x}_1\mathbf{x}_2 \prec \mathbf{x}_1^2.$$

The leading term of a polynomial f is defined with respect to any monomial ordering and the reduction process depends on a given monomial ordering. Given a polynomial f and a set of polynomials $B = \{f_1, \ldots, f_s\}$, the reduction process returns a polynomial r such that

$$f = a_i f_1 + \cdots + a_s f_s + r,$$

where $a_i \in K[\mathbf{x}_1, \ldots, \mathbf{x}_n]$, for $1 \leq i \leq s$ and the leading monomials of the f_i polynomials do not divide the leading monomial of r. We say that f *reduces* to r by B.

Definition 2.1.3 *A Gröbner basis of an ideal $I \subset K[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ for a given monomial ordering is a finite set $B \subset I$ such that any $f \in I$ reduces to 0 by B. The basis is called reduced when the f_i polynomials all have leading coefficient 1 and when none of the f_i polynomials involves a monomial which reduces by $B \setminus \{f_i\}$.*

The notion of a Gröbner basis was first introduced in Buchberger's PhD thesis in 1965 [BCLA83; Buc06], along with a means to test whether a set of polynomials is a Gröbner basis. These findings led to the first algorithm for computing a Gröbner basis from an arbitrary basis of an ideal, called Buchberger's algorithm. Ever since, there have been significant efforts to improve Buchberger's algorithm. Initially, there were Buchberger's own improvements in [Buc79; Buc85]. Then, it was observed by Lazard [Laz83] that performing row reduction without column pivoting on the Macaulay's matrix [Mac16] is equivalent to performing Buchberger's algorithm.

Definition 2.1.4 *Given a set of polynomials $\{f_1, \ldots, f_s\} \in K[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$, each of degree d_i for $1 \leq i \leq s$, then the Macaulay matrix of degree d is the matrix containing the coefficients of*

$$M(d) = \begin{pmatrix} f_1 \\ \boldsymbol{x}_1 f_1 \\ \vdots \\ \boldsymbol{x}_n^{d-d_1} f_1 \\ f_2 \\ \boldsymbol{x}_1 f_2 \\ \vdots \\ \boldsymbol{x}_n^{d-d_s} f_s \end{pmatrix}$$

where each polynomial f_i is multiplied with all monomials from degree 0 up to $d - d_i$ for all i, such that $1 \leq i \leq s$.

Example 2.1.6 Let us consider the following polynomials in variables $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$

$$\begin{aligned} f_1 &= 3\mathbf{x}_1^2 - \mathbf{x}_2^2 + 5\mathbf{x}_1\mathbf{x}_3 + \mathbf{x}_2 - 22 \\ f_2 &= 2\mathbf{x}_1 + \mathbf{x}_2 - \mathbf{x}_3 - 1 \\ f_3 &= 2\mathbf{x}_1\mathbf{x}_2 + \mathbf{x}_2 + 8\mathbf{x}_3. \end{aligned}$$

The Macaulay matrix of degree 2 associated to this system, with respect to the *grevlex* order $\mathbf{x}_1 > \mathbf{x}_2 > \mathbf{x}_3$ is

$$M(2) = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ \mathbf{x}_1 f_2 \\ \mathbf{x}_2 f_2 \\ \mathbf{x}_3 f_2 \end{array} \begin{array}{c} \begin{array}{cccccccccc} \mathbf{x}_1^2 & \mathbf{x}_1\mathbf{x}_2 & \mathbf{x}_2^2 & \mathbf{x}_1\mathbf{x}_3 & \mathbf{x}_2\mathbf{x}_3 & \mathbf{x}_3^2 & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & 1 \end{array} \\ \begin{pmatrix} 3 & 0 & -1 & 5 & 0 & 0 & 0 & 1 & 0 & -22 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & -1 & -1 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 0 \\ 2 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & -1 & 0 & 0 & -1 & 0 \end{pmatrix} \end{array}.$$

Lazard's discoveries on the use of linear algebra for computing Gröbner bases led to the conception of the F4 [Fau99] and F5 [Fau02] algorithms by Faugère. For a detailed description of these algorithms, the reader can refer to [BFS15; Bar04; Spa12; Vit11; Sva14].

The application of Gröbner basis that is the main focus of our work is solving a polynomial system of equations. For solving the system of equations $f_1 = f_1 = \cdots = f_s = 0$, it is most convenient to obtain a Gröbner basis of $I =< f_1, \ldots, f_s >$ with respect to the lexicographic order, as this order allows us to use elimination theory. For $1 \leq k \leq n$, the k^{th} elimination ideal is the ideal $I_k = I \cap K[\mathbf{x}_k, \ldots, \mathbf{x}_n]$. Finding elimination ideals results in a system that has a shape similar to the triangular shape of a linear system obtained from a Gaussian elimination.

Example 2.1.7 Let us consider the ideal

$$I = (\mathbf{x}_1, \mathbf{x}_2 - 1, \mathbf{x}_3 - 4)(\mathbf{x}_1 - 2, \mathbf{x}_2 + 1, \mathbf{x}_3 - 1),$$

given by 9 generators. The Gröbner basis of I with respect to the lexicographic order $\mathbf{x}_1 > \mathbf{x}_2 > \mathbf{x}_3$ is

$$G_{lex} = \{\mathbf{x}_1 + \mathbf{x}_2 - 1, \mathbf{x}_2^2 - 1, \mathbf{x}_2\mathbf{x}_3 + 2\mathbf{x}_2 - 3\mathbf{x}_3 + 6, 3\mathbf{x}_2 - 2\mathbf{x}_3 + 5, \mathbf{x}_3^2 - 5\mathbf{x}_3 + 4\}.$$

The last equation $\mathbf{x}_3^2 - 5\mathbf{x}_3 + 4 = (\mathbf{x}_3 - 1)(\mathbf{x}_3 - 4)$ contains only $\mathbf{x}_3$ and thus, it is trivial to compute $\mathbf{x}_3 = 1$ or $\mathbf{x}_3 = 4$. The second to last equation contains only $\mathbf{x}_2$ and $\mathbf{x}_3$ and thus, after the substitution of $\mathbf{x}_3$ it is trivial to compute $\mathbf{x}_2 = -1$ or, respectively, $\mathbf{x}_2 = 1$. We continue this process until all solutions are found.

Computing a Gröbner basis with respect to the lexicographic order is very hard in practice. Consequently, other orders need to be considered. The most common practice is to first compute the Gröbner basis with respect to the graded reverse lexicographic order, and then use the FGLM algorithm [FGLM93] to change the order of the basis to the lexicographic one. The FGLM algorithm takes as input a Gröbner basis of a zero-dimensional ideal in the ring of polynomials over a field with respect to a monomial order and returns a Gröbner basis of the ideal with respect to a different (specified) ordering.

Similar to SAT solvers, Gröbner basis implementations have been optimized with great care, sometimes even through hand-optimizations of the compiled code. For this reason, it is recommended that these implementations be used, instead of implementing the algorithms from scratch. Best currently available implementation of F4 is in MAGMA [BCP97] and F5 is implemented in the FGb library [Fau10]. For a competitive open source implementation, we reference the SAGE project [S+].

The complexity of computing a Gröbner basis is strongly linked to the cost of the linear algebra on the largest Macauley matrix that will appear during the computation. It is exponential in the *degree of regularity*, denoted D_{reg}, which is the maximal degree of all computed polynomials. Knowing D_{reg}, one can establish an upper bound on the cost of computing a Gröbner basis [BFS15; Laz83; Mac02]:

$$\binom{n + D_{reg}}{D_{reg}}^w,$$

where w is the exponent in matrix multiplication complexity. For Boolean systems, this estimate simplifies to

$$\binom{n}{D_{reg}}^w.$$

Unfortunately, the degree of regularity is difficult to compute for general polynomial systems. It is, in fact, as difficult as computing a Gröbner basis. However, the degree of regularity can be computed efficiently for regular [Mac02] and semi-regular [BFSY05; Bar04] equations.

Gröbner basis algorithms have exponential theoretical complexity, but they have been shown efficient in practical applications, where the system to be solved is derived from a concrete cryptosystem and has some structure. Recall that, in cryptanalysis, the average-case complexity is more significant than the worse-case complexity. On the other hand, Gröbner bases computations have huge memory requirements, which can sometimes be the bottleneck of the resolution.

2.1.2 Linearization and the XL family

The linearization method of a non-linear system consists in replacing all monomials of degree greater than one by an additional variable. For instance, linearization of the system in Equation (2.1) results in a linear system of three equations in the variables $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$:

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

The derived linear system is consistent with all the solutions of the initial system. However, other solutions may have been introduced due to the loss of information. Full linearization only works if the number of (linearly independent) equations is as big as the number of monomials in the system. In this case, the derived linear system is not underdetermined and thus, it either has one solution, which must be consistent with the initial system, or it has no solution, which means that the initial system is inconsistent as well.

When we are not in this favorable case of extremely overdetermined systems, naturally, one would try to derive new equations that express the link between the original variables and the substitution variables. This is the idea behind the family of

XL algorithms. The extended linearization (XL) technique was introduced by Courtois *et al.* in 2000 [CKPS00], inspired by the then recently proposed relinearization algorithm [KS99]. Initially, the XL algorithm was thought to be subexponential if m exceeds n by a small number, until this claim was disputed [Die04]. Nevertheless, there are many variants and analysis of this algorithm in the literature [YC04; YC05; CP02; CL05], as well as hybrid methods that use the XL technique [JV17].

Interestingly, the XL method also produces a Gröbner basis and bears many similarities to algorithms in the F4/F5 family. Consequently, these two families of algorithms have similar complexities [AFI+04; YCY13].

2.1.3 Exhaustive search

A straightforward method for solving multivariate polynomial systems is the exhaustive search method, which consists in enumerating all possible values for the n variables and testing whether they are consistent with all m equations in the system. According to the analysis in [Bar04] for solving the worst-case ($m = n$) quadratic systems over $\mathbb{F}_q$, a Gröbner basis algorithm, such as the F5, has better asymptotic complexity than the exhaustive search for $q \geq 20$. Consequently, Gröbner basis algorithms are expected to be slower than enumeration algorithms for the $\mathbb{F}_2$ case.

Currently, the fastest enumeration method is the algorithm of Bouillaguet *et al.*, proposed in [BCC+10], resulting in the LibFES open source library [Bou16]. Initially, this algorithm was implemented for CPU and GPU and was able to solve 48 quadratic equations in 48 variables in 21 minutes. Later, an FPGA implementation with a lesser energy consumption was proposed in [BCC+13]. Using this algorithm, solving quadratic systems over $\mathbb{F}_2$ require $4 \log n \cdot 2^n$ operations. Note that, this complexity does not depend on the number of equations in the system.

A quantum version of the exhaustive search method can be developed using Grover's algorithm [Gro96]. According to the analysis in [SW16] on pure enumeration for solving Boolean multivariate quadratic systems, the expected running time of Grover's algorithm on a large quantum computer is $O(2^{n/2})$.

2.1.4 Hybrid methods

A prominent line of research is combining exhaustive search with Gröbner basis or XL methods. This is commonly known as the hybrid method. The main idea behind this method is to guess a portion of the variables and then compute several Gröbner bases on smaller systems. The number of variables that are fixed beforehand is known as the *trade-off* parameter. It is well known that the cost of a Gröbner basis computation decreases when the number of equations is greater than the number of variables, as overdetermined systems have lower degrees of regularity. The hybrid approach yields a better time complexity than a pure Gröbner basis algorithm when the cost of Gröbner bases computations is decreased significantly enough to compensate for the cost of exhaustive search on the fixed variables.

In the family of hybrid methods there are many state-of-the-art algorithms for solving multivariate polynomial systems. The FXL algorithm [YC04; YC05] is a derivative of the XL method. For a complexity analysis of this algorithm, see [YCC04]. The hybrid approach, proposed in 2009 [BFP09], is best adapted for systems over $\mathbb{F}_q$ when $q > 2$. It has been further studied in [BFP12]. For the special case of $q = 2$, there is the BooleanSolve algorithm [BFSS13], which is a combination of exhaustive search and sparse linear algebra. Finally, we have Joux-Vitse's hybrid algorithm proposed in 2017 [JV17]. This algorithm

currently holds the record for solving instances from the MQ challenge [YDH+15]. At the time of writing, this hybrid algorithms is not open source, however, there is an open source GPU implementation by Niederhagen *et al.* [NNY18]. The solving algorithms mentioned in this paragraph surpass pure enumeration algorithms and are considered state-of-the-art for solving multivariate polynomial systems.

2.1.5 Algorithms for sparse systems

When solving sparse polynomial systems, it is a common practice to concentrate directly on the set of solutions. The model that we consider *sparse* is one where only a small subset of variables appear in each equation. Systems derived from reduced round block or stream ciphers are often likely to fall in this category, as variables from one round are usually related to variables from the previous and the following round. For example, the Bivium [Rad06] and Trivium [DC06] stream ciphers each have 6 or fewer variables in the equations of their algebraic cryptanalysis model.

Algorithms dedicated to this type of systems are in the Agreeing-Gluing algorithm family [RS06; RS07; Sem08; Sem16]. A prominent example in this category is the Raddum-Semaev algorithm [RS06], which has been used in one of the first attacks on Trivium [Rad06]. The idea behind it is to consider each equation separately and eliminate all solutions that are invalidating the equation. What is unique about this algorithm is that the equations are represented as vertices in a graph and the agreeing method is represented as message-passing on a graph. Later, Raddum and Semaev introduced a generalized technique with a different representation [RS07]. With this technique, the equations are represented as a system of Multiple Right Hand Sides linear equations, hence the name MRHS.

2.2 Applications in cryptography

Algebraic cryptanalysis applies to a variety of cryptosystems, ranging from symmetric blockciphers like the Advanced Encryption Standard (AES) and stream ciphers like Trivium [DC06], to public-key cryptosystems, where the most prominent example is Hidden Field Equations (HFE) [Pat96a].

HFE is a public-key cryptosystem based on the problem of solving a multivariate polynomial system of equations over a finite field. The first HFE challenge, proposed by Patarin [Pat96b], was broken by Faugère and Joux in 2003 [FJ03]. The challenge consisted of 80 quadratic equations in 80 variables over $\mathbb{F}_2$. The computation was performed using the F5 algorithm implemented in C and took two days and four hours on a computer with a 1 Ghz processor and 4 GB of RAM. The huge impact of this computation lies in its demonstration that the algebraic systems arising from HFE do not behave as random systems. The challenge was solved for the second time by Steel in 2004 [Ste04]. The latter computation used the F4 implementation in Magma 2.11 and took 25.4 hours on a 750 MHz processor using 15 GB of memory.

Another public-key cryptosystem that has been a target of algebraic attacks is the McEliece cryptosystem [McE78], based on the hardness of decoding a random linear code. McEliece was introduced in 1978, alongside RSA and Diffie-Hellman, but was not considered as a standard due to the huge public key size. However, the cryptosystem is now considered for building post-quantum cryptography, as, contrary to RSA and Diffie-Hellman, it is not theoretically vulnerable to attacks on a quantum computer. There are recent developments in algebraic attacks on code-based cryptosystems. A new approach for solving the Rank Metric Decoding problem with Gröbner basis techniques was

introduced at Eurocrypt 2020 [BBB+20].

Lastly, algebraic cryptanalysis is one of the most common approaches to attacking cryptosystems based on the Multivariate Quadratic problem (MQ). Cryptographic schemes in this family include, for instance, the UOV-based [KPG99] signatures Rainbow [DS05] and LUOV [BP17] and encryption schemes, such as ABC [TDTD13; DPW14], ZHFE [PBD14] and QUAD [BGP06].

Part I

SAT as a tool

Chapter 3

The Satisfiability problem

The SAT problem is part of the wider domain of constraint programming. The idea behind constraint programming is to use a formal language to describe a problem at a high level. This allows us to solve the problem using the power of solvers that have already been developed and whose efficiency has immensely improved due to years of research. Thus, more often than not, this approach allows us to solve the problem more efficiently than we would with a novel, dedicated algorithm. To this end, the problem at hand needs to be described in terms of *variables* and *constraints*. The third component that forms the language of constraint programming is a finite set of potential values for the variables, called the set of *domains*. Hence, *solving* the problem can be defined as finding, for all variables, an assignment of values from the domain that satisfies all constraints. Sometimes there is an objective function that needs to be minimized and thus, we are looking for an optimal solution instead of just any solution. In this case, we have a *constraint optimization problem*.

When the domain for all variables is the Boolean set $\{\text{TRUE}, \text{FALSE}\}$, and the constraints are described as a propositional formula in a standard form, we have described a SAT problem. Describing a problem at a high level as a SAT problem is referred to as SAT modelization or a SAT encoding. In this dissertation, we use the former wording.

SAT is used as a tool for solving various problems starting from automated deduction, which is historically the motivation behind developing modern SAT solving techniques [Bib07]. Other applications include hardware and software verification, planning, scheduling, optimal control, protocol design, routing, bioinformatics, combinatorial problems, and many more. In recent years, SAT solvers have been proven to be a powerful tool in cryptanalysis, making XOR constraints a popular area of research. The appeal behind the use of SAT solvers is evident when we look at the progress that has been made to their efficiency over the last 20 years. They have gone from treating 100 variables with 200 constraints in the early 90s to a million variables with 20 million constraints nowadays. Propositions for novel solvers or extensions to state-of-the-art solvers are breaking records in every biennial SAT competition [vMF].

3.1 Preliminaries

Propositional variables can take two possible truth values: TRUE and FALSE ($\top$ and $\bot$). Propositional formulas are built upon binary operators and the unary operator $\neg$, the negation. The negation operator changes a value TRUE to FALSE and vice versa. Table 3.1 represents a *truth table* giving definitions of five common binary operators in the following order: logical conjunction (AND), logical disjunction (OR), logical implication, logical equivalence and exclusive disjunction (XOR).

Table 3.1: Truth table of common binary operators

x_1	x_2	$\wedge$	$\vee$	$\Rightarrow$	$\Leftrightarrow$	$\oplus$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$

State-of-the-art SAT solvers are not adapted to handle all of the operators described in Table 3.1. They use a standard form, known as conjunctive normal form (CNF), containing only the negation, logical AND, and logical OR. Before we give a precise definition of a CNF formula, we need to define other notions from the SAT terminology.

- 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 ;
- 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.

The use of CNF was proposed by Davis and Putnam in 1958 [DP60], along with the essentials for modern SAT solving techniques. For every propositional formula, there is an equivalent CNF formula that can be obtained using some of the following transformation techniques.

- Double negations are eliminated: $\neg\neg x$ is replaced by x ;
- A negation can only appear in literals. If this is not the case, the negation can be moved inwards using De Morgan's law: $\neg(x_1 \wedge x_2)$ is replaced by $\neg x_1 \vee \neg x_2$ and $\neg(x_1 \vee x_2)$ is replaced by $\neg x_1 \wedge \neg x_2$;
- Implications are eliminated: $x_1 \Rightarrow x_2$ is replaced by $\neg x_1 \vee x_2$;
- Equivalences can be transformed to two-way implications and are then eliminated using the previous step: $(x_1 \Leftrightarrow x_2)$ is replaced by $(x_1 \Rightarrow x_2) \wedge (x_2 \Rightarrow x_1)$;
- The truth table of the XOR operator holds the exact opposite values of the truth table of the equivalence, thus: $(x_1 \oplus x_2)$ is replaced by $\neg(x_1 \Leftrightarrow x_2)$;
- As per the distributive law for $\vee$ over $\wedge$: $x_1 \vee (x_2 \wedge x_3)$ is replaced by $(x_1 \vee x_2) \wedge (x_1 \vee x_3)$;

The XOR operator is commonly used in cryptography and thus, it is the central focus of our work. Another transformation technique will be presented in Chapter 2 that can be applied to bigger-size XOR-*clauses*. A XOR-clause is an exclusive disjunction ($\oplus$) of literals. (e.g. $x_1 \oplus \neg x_2 \oplus x_3$), that is said to be satisfied (resp. falsified) if an even (resp. odd) number of its literals is satisfied. Due to the rising interest in the XOR operator and XOR-handling techniques inspired from algebra, there exist SAT solvers that are adapted to read and handle formulas containing XOR-clauses.

3.2 Solving techniques

A straightforward method for solving the SAT problem is to complete the truth table associated with 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 that can satisfy the clause, it must be set to TRUE and therefore *propagated*. The described method is called *unit propagation*. Unit propagation is classified as an inference technique because it is used to infer new assignments. Inference techniques are also referred to as simplification techniques as they reduce the number of variables or clauses in the formula. The reader can refer to [BHvMW09] for more details.

A *conflict* occurs when there 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, it has to be undone. This is commonly known as *backtracking*.

Example 3.2.1 For instance, these two atomic operations can be illustrated with the following example built of a set of 5 clauses numbered C_1 to C_5:

$$C_1 : \neg x_1 \vee x_2 \vee \neg x_4$$
$$C_2 : x_1 \vee x_3 \vee x_4$$
$$C_3 : x_1 \vee \neg x_3$$
$$C_4 : x_1 \vee x_3$$
$$C_5 : x_2 \vee x_4$$

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

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

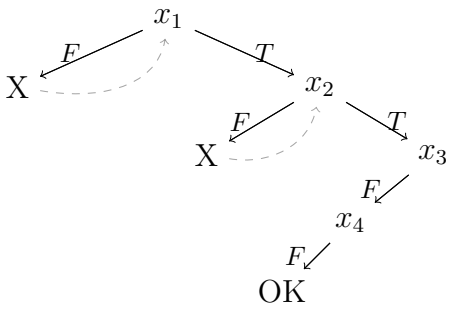


Figure 3.1: Binary search tree constructed with the DPLL algorithm.

The basic backtracking search with unit propagation that we describe gives the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DLL62], which is a state-of-the-art complete SAT solving technique. DPLL works by trying to assign a truth value to each variable in the CNF formula, recursively building a binary search tree of height equivalent (at worst) to the number of variables. After each variable assignment, the formula is simplified by unit propagation. When a *conflict* occurs, a backtracking procedure is launched and the opposite truth value is assigned to the last assigned literal. If the opposite truth value results in conflict as well, we backtrack to an earlier assumption or conclude that the formula is *unsatisfiable* - when there are no earlier assumptions left. The number of conflicts is a good measure for the time complexity of a SAT problem solved using a DPLL-based solver. If the complete search tree is built, the worst-case complexity is $O(2^v)$, where v is the number of variables in the formula. Figure 3.1 illustrates the binary search tree resulting from the resolution of Example 3.2.1.

A common variation of the DPLL is the conflict-driven clause learning (CDCL) algorithm [MSS96; MSS97]. In this variation, each encountered conflict is described as a new clause which is *learnt* (added to the formula). These algorithms use conflict analysis techniques, and their branching heuristics, responsible for determining the order of branching variables, are conflict-driven [MMZ+01]. In other words, branching heuristics receive feedback from the backtrack search to choose the branching order dynamically. CDCL methods are responsible for rendering SAT solvers so efficient in practice, since their invention in the mid-90s. Most of the modern SAT solvers are CDCL-based, including all solvers used to obtain experimental results in this thesis.

The CDCL algorithm keeps the DPLL core that results in a binary search tree, but with two main differences:

- There is a *conflict analysis* procedure that is executed each time a conflict occurs and before the backtracking procedure.
- The backtracking procedure can be non-chronological, which means that when a conflict occurs on level d, there is a 'jump' to an upper level that is not necessarily level $d-1$. The exact level is computed by the conflict analysis procedure.

The conflict analysis procedure uses a so-called implication graph to analyze the structure of the unit propagation that has led to a conflict. The implication graph is constructed as follows. Vertices in the graph can be either assigned variables or conflicts. Another information that vertices hold is the level at which the variable was assigned. A directed edge from one vertex to another exists if the variable assignment corresponding to the first vertex has led to the propagation of the variable assignment corresponding to the second vertex. The edges are labeled with the clause that was used for the propagation, known as the *antecendent*. This process defines a directed acyclic graph.

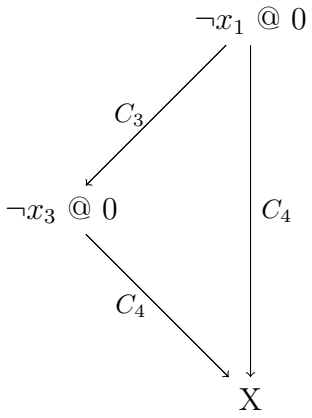


Figure 3.2: Implication graph corresponding to the first conflict in Figure 3.1.

Example 3.2.2 Using the same set of clauses as in Example 3.2.1, we show an implication graph in Figure 3.2. This graph describes the unit propagation steps that led to the first conflict in the DPLL algorithm (see the leftmost branch of the tree in Figure 3.1). We note as '@ d' the level at which the assignment of the variable occurred (whether through an assumption or unit propagation).

We now describe how the implication graph is used for conflict analysis to learn one or multiple new clauses. In this process, there are intermediate learned clauses that will not be added to the formula. We will denote both intermediate and final learned clauses by C_L^i. We start by taking the clause that provoked the conflict. In Example 3.2.2, we take C_4. Then, we take a second clause from one of the clauses that led to an assignment of a variable at the decision level at which the conflict occurred. The second clause is C_3 in our example. These two clauses may be used to learn a new clause C_L^1. Learned clauses are derived using the *resolution operator*. This operator takes two clauses that have exactly one variable x_i such that one clause contains the literal x_i and the other contains $\neg x_i$. If there is no such an occurrence, the process of learning clauses will be stopped. In our example, we have the literal x_3 in clause C_4 and the literal $\neg x_3$ in clause C_3. Thus, the clause resulting from the application of the resolution operator on clauses C_3 and C_4 is the clause $C_L^1 : x_1$. Since there are no more clauses that have participated in unit propagation at this decision level, we can end this process and keep the clause C_L^1. Another criterion for ending the conflict analysis is if the latest learned clause contains exactly one literal that was assigned at the current decision level. This literal is known as a Unit Implication Point (UIP) and applying this rule allows us to obtain smaller clauses. UIPs are also useful for backtracking. As per the Chaff proposition [ZMMM01], it is common practice to always take the first UIP and decide on the backtracking level according to the learned clauses. Namely, for each literal in the learned clause, we find the level at which the corresponding variable was assigned, and then we choose the highest level among them. In our example, x_1 is the only literal and thus we backtrack to level 0 without making a 'jump'.

Example 3.2.3 Continuing the resolution of the formula from Example 3.2.1, we encounter the next conflict at level 1. The implication graph for this conflict is illustrated in Figure 3.3. The conflict analysis goes as follows. First, we detect C_5 as the clause that was found unsatisfied through unit propagation. To find this clause using the graph, we look at the label on an edge that is pointing to the conflict. Then, the only other clause that participated in this decision level is C_1. To find such clauses, we look for edges pointing to a vertex labeled as '@ d', where d is the decision level at which the conflict

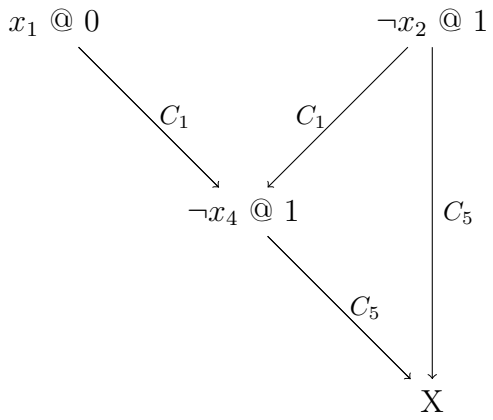


Figure 3.3: Implication graph corresponding to the first conflict in Figure 3.1.

occurred. Here, d is 1. By applying the resolution operator on clauses $C_5 : (x_2 \vee x_4)$ and $C_1 : (\neg x_1 \vee x_2 \vee \neg x_4)$, we obtain a learned clause $C_L^2 : (\neg x_1 \vee x_2)$. We stop the conflict analysis and we launch a backtracking procedure. The next implication graph has no conflict and we conclude that the formula is satisfiable. For a more elaborate example where the UIP rule is used and a non-chronological backtracking step occurs, see [BHvMW09].

The techniques explained in this section define the basics of the CDCL method. Modern SAT solvers use additional techniques to fix some issues that arise at an implementation level or to render SAT solvers even more efficient. These techniques include search restarts, clause deletion strategies, lazy data structures, and, as mentioned, conflict-driven branching heuristics.

Both DPLL and CDCL are complete SAT solving algorithms, which means that they are guaranteed to terminate with a correct answer on the satisfiability of the CNF formula. Other algorithms in this category are look-ahead based algorithms [BS96; Fre95; Li99; Li00; DD04]. On the other hand, incomplete algorithms usually have a fixed resource limit and will either find a satisfiable solution or report failure without declaring the formula to be unsatisfiable [SLM92; SKC93]. They are generally based on stochastic local search methods and are most efficient for random SAT instances or specific problems for which complete methods do not scale well. In this chapter, we have only detailed solving methods that are used in our work or in the solvers that we use for comparing experimental results. For more details on the methods that are not explained here the reader can refer to [BHvMW09] or to their respective citations.

3.3 Applications in cryptography

The first proposition of logical cryptanalysis [MM00] uses the (now obsolete) Data Encryption Standard (DES) to demonstrate how to model a cryptanalysis attack as a SAT problem. After these first results, 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 Crypto1, Bivium, Trivium, Grain, HiTag2 [MCP07; LJN14; HJ12a; SNC09; Soo10a]. However, their use in public key cryptosystems has rarely been considered. A prominent example is the work of Galbraith and Gebregiyorgis [GG14], where they explore the possibility of replacing available Gröbner basis implementations with generic SAT solvers in the index calculus attack on binary elliptic curves.

State-of-the-art CDCL solvers, such as MiniSat [SE05] and Glucose [AS09], have been shown to be a powerful device 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) [SNC09; Soo10a], Han and Jiang [HJ12a], Laitinen *et al.* [LJN14; LJN11]. XOR-enabled SAT solvers usually use formulas in a specific form, which is a conjunction of OR-clauses and XOR-clauses. We refer to this as a CNF-XOR form and it will be detailed in the next chapter.

All attacks mentioned thus far are derived from an algebraic cryptanalysis attack. For some cryptosystems, such as the Advanced Encryption Standard (AES), leading attacks are found in the field of differential cryptanalysis. In this case, other constraint programming (CP) techniques are used to aid the attack, as the derived constraints are not Boolean. A first example in the field of differential cryptanalysis is given by the work of Gerault *et al.* [GMS17; GLMS18; GLMS20] who showed how to use CP for solving the optimal related-key differential characteristic problem. We also note the work of Lui *et al.* [LCM+17; LCM18], in which a CP model is used to aid the Tolerant Algebraic Side-Channel Analysis, which is a combination of algebraic and side-channel analysis.

3.3.1 Deriving a SAT model from a Boolean polynomial system

SAT solvers are powerful tools that can be used for solving a variety of problems, including the problem of solving multivariate polynomial systems. This is the most common use of SAT solvers in cryptography. Even though a general knowledge of the solving techniques is recommended, a SAT solver can be used as a black-box. To this end, there is a uniform representation for the input formula that is given to a SAT solver, known as a CNF formula (see Section 3.1). XOR-enabled SAT solvers can also read a formula in a so-called CNF-XOR form. Both these models are presented in this section, along with standard techniques for deriving these models from a Boolean polynomial system.

ANF to CNF-XOR

In Boolean algebra, the closest form to a Boolean polynomial is the algebraic normal form (ANF). In this form, a *term* is either one variable or multiple variables ANDed together. The TRUE/FALSE constants are also considered terms. One or more terms XORed together create an ANF. The operator $\neg$ is not allowed. A Boolean polynomial system can be rewritten as a conjunction of logical formulas in ANF as follows: multiplication in $\mathbb{F}_2$ ($\cdot$) becomes the logical AND operation ($\wedge$) and addition in $\mathbb{F}_2$ ($+$) becomes the logical XOR ($\oplus$). The elements 0 and 1 in $\mathbb{F}_2$ correspond to $\bot$ and $\top$, respectively. Consequently, solving a multivariate Boolean polynomial system is equivalent to solving a conjunction of logical formulas in ANF form. To date, few SAT solvers are adapted to tackle formulas in ANF. A common approach is to transform the ANF form in a CNF-XOR form, which is a conjunction of CNF and XOR clauses. In order to do this, every conjunction of two or more literals $x_1 \wedge x_2 \wedge \ldots \wedge x_k$ has to be replaced by an additional and equivalent variable $x_{1,\ldots,k}$ such that

$$x_{1,\ldots,k} \Leftrightarrow x_1 \wedge \ldots \wedge x_k$$

This equivalence can be rewritten in CNF using a three-step transformation, described in Figure 3.4.

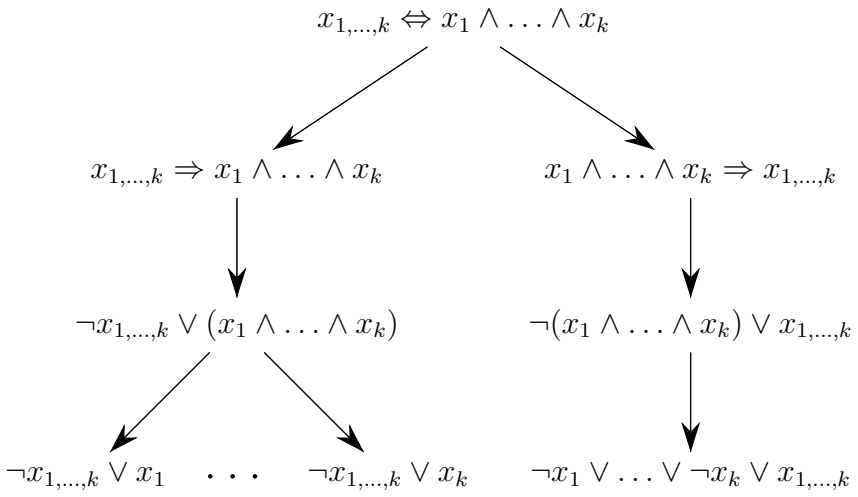


Figure 3.4: A three-step transformation to CNF constraints.

We obtain the following CNF formula:

$$\begin{aligned}
&(\neg x_{1,\ldots,k} \vee x_1) \wedge \\
&(\neg x_{1,\ldots,k} \vee x_2) \wedge \\
&\ldots \\
&(\neg x_{1,\ldots,k} \vee x_k) \wedge \\
&(\neg x_1 \vee \neg x_2 \vee \ldots \vee \neg x_k \vee x_{1,\ldots,k}).
\end{aligned} \tag{3.1}$$

When we substitute all occurrences of conjunctions in an XOR clause by an additional variable, we obtain a formula in CNF-XOR form. This is the form used in the CryptoMiniSat solver [SNC09], which is an extension of the MiniSat solver [ES04] specifically designed to work on cryptographic problems.

Example 3.3.1 Let us consider the Boolean polynomial system:

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

One additional variable $x_{2,3}$ needs to be introduced to substitute the monomial $\mathbf{x}_2\mathbf{x}_3$. The corresponding CNF-XOR form for this Boolean system is a conjunction of the following clauses:

$$\begin{aligned}
&\neg x_{2,3} \vee x_2 \\
&\neg x_{2,3} \vee x_3 \\
&\neg x_2 \vee \neg x_3 \vee x_{2,3} \\
&x_1 \oplus x_{2,3} \oplus x_5 \oplus x_6 \\
&x_3 \oplus x_5 \oplus x_6 \oplus \top.
\end{aligned}$$

Complexity discussion

The transformation that we described results in a system composed of two distinct parts. We will refer to these as the CNF part and the XOR part of the system. The substitution of non-unary terms makes this transformation similar to the linearization method. Unlike linearization, this transformation method does not result in loss of information. Indeed, the derived CNF-XOR formula is equivalent to the initial ANF formula.

The DPLL and other complete SAT solving methods can be viewed as an elimination procedure on the domain of possible solutions. For CNF-XOR formulas, this elimination procedure needs to consider constraints from both parts of the model simultaneously. This is easier said than done, as the two parts have a different structure and thus, require different solving techniques. The XOR part of the formula is essentially a linear polynomial system and can be solved in polynomial time using techniques like Gaussian elimination. Another way to view or represent the XOR part is as the Macaulay matrix [Mac16] of the initial Boolean polynomial system. Surprisingly, the CNF part of this formula can also be solved in polynomial time due to its specific structure. Indeed, the CNF formula is comprised only of *Horn* clauses [Hor51] and solving this formula is a HORNSAT problem. A Horn clause is a clause that contains at most one positive literal and any number of negative literals. The HORNSAT problem is solvable in polynomial time [DG84] using the following technique. First, the unit propagation rule is applied for all clauses (if any) composed of a single literal l. Recall, as per unit propagation, all clauses containing l are removed (because they are satisfied) and all occurrences of $\neg l$ are removed from other clauses. If there are new unary clauses introduced as a result of unit propagation, the process is repeated. When there are no unary clauses left, all non assigned literals are assigned to FALSE. Note that, a Horn clause remains a Horn clause when a literal is removed. When there are no unary clauses in the system, all of the remaining Horn clauses contain at least one negative literal. Thus, assigning all remaining literals to FALSE is sure to give us a solution that satisfies all clauses. This is not as surprising when we recall that each block of clauses of the form (3.1), actually describes the multiplication rules. For more details on Horn-satisfiability, the reader can refer to [BHvMW09], for example.

Even though finding one solution to the HORNSAT problem is easy, finding the entire set of solutions is as hard as the general SAT problem. We encounter a similar problem with the XOR part. Unless we are in the favorable case of very overdetermined systems, there are many solutions to the linearized XOR part and thus, enumerating them all becomes a hard problem. In conclusion, finding a solution to the CNF and to the XOR part separately is easy, however, to date, there is no polynomial-time algorithm that looks for solutions in the intersection.

Open question 1 Models derived from cryptographic attacks introduce a fundamental question in SAT theory, which is how to combine two SAT problems with polynomial complexity and in which complexity class would such an algorithm be placed.

CNF-XOR to CNF

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

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

$$\begin{aligned} &(x_1 \vee \neg x_2 \vee \neg x_3) \wedge \\ &(\neg x_1 \vee x_2 \vee \neg x_3) \wedge \\ &(\neg x_1 \vee \neg x_2 \vee x_3) \wedge \\ &(x_1 \vee x_2 \vee x_3) \end{aligned} \tag{3.2}$$

Similarly, a XOR-clause of size k can be transformed into 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:

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

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.

The advantage of this model is its standard form, due to which it can be solved using any off-the-shelf SAT solver. On the other hand, this model is not optimal for solving formulas arising from cryptographic problems. The XOR operator is extensively used in cryptography and the pure CNF model does not allow for exploiting powerful XOR-reasoning techniques such as Gaussian Elimination and parity reasoning.

Chapter 4

The WDSat Solver

SAT solvers are powerful tools for solving many practical problems. As we explained previously, it is usually better to use off-the-shelf SAT solvers instead of writing our own algorithms, as these SAT solvers have been optimized over the years. However, in this work, we built a SAT solver from scratch. The motivation behind this endeavor is the need that we have found for a special purpose SAT solver for non-sparse Boolean polynomial systems. Constraint programming solvers have been successfully used in differential attacks on AES [GLMS18; GLMS20], but, similarly to Gröbner bases, are not optimal for Boolean constraints. General-purpose SAT solvers such as MiniSat and Glucose are better suited for Boolean systems. It is common that cryptanalysts turn to these tools as they are efficient and easy to use. However, these solvers do not handle XOR constraints efficiently due to their input form. Indeed, to use these solvers, one needs to derive a CNF model and, as we explained in Chapter 3.3.1, transforming XOR constraints into OR constraints can often result in huge formulas. Plus, through this transformation process, we lose some of the structure of the system and we can no longer use efficient XOR-handling techniques. Nevertheless, MiniSat has been used many times to aid a cryptographic attack with promising results. Recently, MiniSat was used for performing an index calculus attack on elliptic curves in characteristic two [GG14]. In their work, Galbraith and Gebregiyorgis observe experimentally that the use of SAT solvers may potentially enable larger factor bases to be considered for point decomposition. This result was one of the main motivations for our work, as we detected some improvements that can be made to the model, as well as the solver, for this particular attack. The model that we propose for this attack is presented in Chapter 7. As our solver was initially conceived with this model in mind, we have named it WDSat, a SAT solver for systems arising from a Weil Descent. Other extensions to the solver that have, thus far, shown an improvement only for the index calculus model, are presented in Chapter 7 as well.

There exist SAT solvers that are XOR-enabled and use XOR-handling techniques inspired from algebra [SNC09; CSCM19]. Moreover, there is extensive research on integrating XOR-handling in SAT solvers [BM00; Che09; HJ12b; LJN11; Li00]. However, we have detected certain limitations of the Gaussian Elimination (GE) technique in XOR-enabled SAT solvers that need to be addressed. Specifically, we have found a canceling property that is present in algebraic resolution methods but is overseen in current SAT-based GE implementations. Section 4.3 is dedicated to a solution to overcome these limitations, developed as an extension of WDSat.

Our solver is highly customizable, as we find that, in most cases, it will be used by cryptanalysts that have a good understanding of the model that needs to be solved. In the following chapters, we give configuration recommendations for each model that we propose. The contributions of this work are presented in [TID20b].

4.1 Core algorithm

Our WDSAT solver is based on the DPLL algorithm [DLL62], described in Section 3.2. The code for the WDSAT solver is written in C and is publicly available [TID20d]. The solver implements three reasoning modules. These include the module for reasoning on the CNF part of the formula and the so-called XORSET and XORGAUSS (XG) modules designed for reasoning on XOR constraints. The CNF module is designed to perform classic unit propagation on OR-clauses. The XORSET module performs the operation equivalent to unit propagation, but adapted for XOR-clauses. Practically, this consists in checking the parity of the current interpretation and propagating the unassigned literal. Finally, the XG module is designed to perform GE on the XOR constraints dynamically. We also implement an XG extension, described in Section 4.3.

As per the DPLL paradigm, the WDSAT solver assigns a truth value to each variable in a formula F, recursively building a binary search tree. After each assignment, either the formula is simplified and other truth values are inferred or a conflict occurs. In the case of a conflict, the last assignment has to be undone for each module via a backtracking procedure. This technique, which is at the core of our solver, is illustrated in Algorithm 4.1 as a recursive SOLVE function.

Algorithm 4.1 Function SOLVE(F) : DPLL algorithm.

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 next x.
5: (contradiction, F') ← ASSIGN(F, ¬x).
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') ← ASSIGN(F, x).
16: if contradiction then
17:    BACKTRACK().
18:    return FALSE.
19: end if
20: return SOLVE(F').
```

Algorithm 4.1 uses the ASSIGN function, detailed in Algorithm 4.2. The purpose of the ASSIGN function is to syncronize all three modules in the following manner. First, the truth value is assigned in the CNF module, and truth values of other variables are propagated. Next, the truth value of the initial variable, as well as the propagated ones are assigned in the XORSET module. If the XOR-adapted unit propagation discovers new truth values, they are assigned in the CNF module, going back to step one. We go back and forth with this process until the two modules are synchronized and there are no

more propagations left. Finally, the list of all inferred literals is transferred to the XG module. If the XG module finds new XOR-implied literals, the list is sent to the CNF module and the process is restarted. If a conflict occurs in any of the reasoning modules, the ASSIGN function fails and a backtracking procedure is launched. We briefly detail the other functions used in the pseudo-code. There is a SET_IN function for each module which takes as input a list of literals and a propositional formula F and sets all literals in this list to TRUE in the corresponding modules. Through this assignment, the function also infers truth values of other literals, according to the specific rules in different modules. Finally, the LAST_ASSIGNED function in each module returns the list of literals that were assigned during the last call to the respective SET_IN function.

Algorithm 4.2 Function ASSIGN(F, x) : Assigning a truth value to a literal x in a formula F, simplifying F and inferring truth values for other literals.

Input: The propositional formula F, a literal x
Output: FALSE if a conflict is reached, TRUE and a simplified F otherwise

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

Input forms

Our solver can read formulas in both forms described in Chapter 3: ANF and CNF-XOR. Reading a formula in ANF comes with two advantages. The first one is linked to branching rules and the second is that this form allows us to use the extension of the XG module.

Branching rules

Reading a formula in ANF, the solver can store the information of which variables comprise the initial system, as opposed to variables that are added to substitute a conjunction. In the remaining of this dissertation, we will refer to these variables as *unary* variables. Variables that are introduced as a substitution of a conjunction will be referred to as non-unary variables, or informally, monomials. Assigning truth values to all unary variables

will necessarily propagate all other variables. Thus, only unary variables are considered in the binary search. Conflict-driven branching heuristics can not be used in WDSAT, as the solver does not perform conflict analysis. This choice is further discussed in Section 4.2.1 of this chapter.

We have developed a heuristic branching technique specific to SAT instances derived from Boolean polynomial systems. This technique, inspired by the Minimal Vertex Cover problem from graph theory, is currently used only during preprocessing to provide us with a predetermined branching order. Thus, the solver does not use heuristics to decide on the order of branching variables dynamically, but the order can be specified by the user. This feature is to be used if the user has more information on the system or if the preprocessing technique was applied. Examples of both uses are given in Chapter 7.

4.2 Three reasoning modules

In this section, we give a description and implementation details of the three modules that make up the WDSAT solver. Each module has its own propagation stack, called the *CNF_propagation_stack*, the *XORSET_propagation_stack* and the *XG_propagation_stack*. These stacks are used for communication between the modules. In fact, their content is provided as input to the respective SET_IN functions by the enveloping Algorithm 4.2. For simplicity, we consider that these stacks and all other data structures relative to the modules are included in the structure F, simply referred to as the propositional formula.

4.2.1 CNF module

In this module, responsible for unit propagation on OR-clauses, we transform the OR-clauses into lists of implications. The unit propagation rule can be informally stated as follows. If all except one literal in the clause are assigned and none of them satisfies the clause, then the remaining literal is implied. According to this rule, from a binary clause $(x_1 \vee x_2)$, we derive two implications: $\neg x_1$ implies x_2 and $\neg x_2$ implies x_1. At the implementation level, lists of implications are stored in an array indexed by both negative and positive literals. This idea for fast unit propagation is presented in [HDvZvM04] and used in the March SAT solver.

Example 4.2.1 Let us consider the following set of 10 clauses numbered C_1 to C_{10}.

$$
\begin{aligned}
C_1 &: \neg x_{1,2} \vee x_1 \\
C_2 &: \neg x_{1,2} \vee x_2 \\
C_3 &: \neg x_1 \vee \neg x_2 \vee x_{1,2} \\
C_4 &: \neg x_{1,3} \vee x_1 \\
C_5 &: \neg x_{1,3} \vee x_3 \\
C_6 &: \neg x_1 \vee \neg x_3 \vee x_{1,3} \\
C_7 &: \neg x_{1,3,4} \vee x_1 \\
C_8 &: \neg x_{1,3,4} \vee x_3 \\
C_9 &: \neg x_{1,3,4} \vee x_4 \\
C_{10} &: \neg x_1 \vee \neg x_3 \vee \neg x_4 \vee x_{1,3,4}
\end{aligned}
$$

The lists of binary implications derived from these clauses are illustrated in Figure 4.1. Intuitively, we can see that these clauses are part of a model describing a Boolean polynomial system that contains the monomials $\mathbf{x}_1\mathbf{x}_2$, $\mathbf{x}_1\mathbf{x}_3$ and $\mathbf{x}_1\mathbf{x}_3\mathbf{x}_4$.

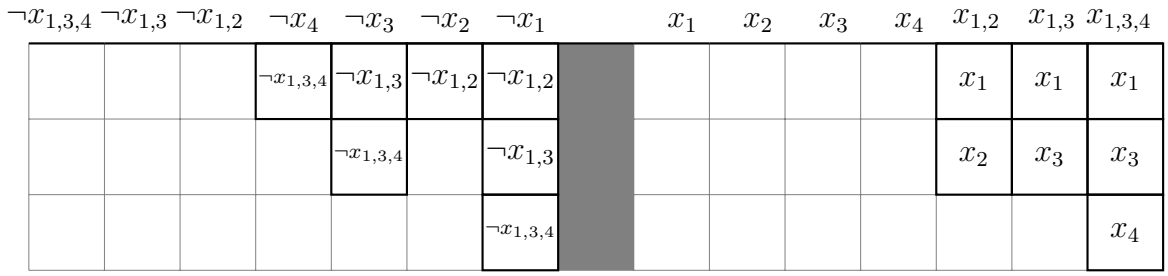


Figure 4.1: Lists of binary implications.

Binary implication data structures, such as the one in Figure 4.1, are used by Algorithm 4.3, which propagates binary implications.

Algorithm 4.3 Function ADD_BINARY_IMPLICATIONS(l, *binary_implications*) : Function for adding implied literals to the *CNF_propagation_stack*.

Input: A literal l and an array holding lists of binary implications *binary_implications*
Output: The *CNF_propagation_stack* is modified.

```
1: for each l_implied in binary_implications[l] do
2:     add l_implied to CNF_propagation_stack.
3: end for
```

For clauses of size bigger than two, we store implications in a similar way, the difference being that instead of a single literal, implications are tuples of literals. For instance, from a ternary clause $(x_1 \vee x_2 \vee x_3)$ we derive three implication: $\neg x_1$ implies $(x_2 \vee x_3)$, $\neg x_2$ implies $(x_1 \vee x_3)$ and $\neg x_3$ implies $(x_1 \vee x_2)$. See figures 4.2 and 4.3 that show the lists of ternary and quaternary implications derived from Example 4.2.1. Now, the way we search for implied literals needs to be adapted accordingly. The function ADD_TERNARY_IMPLICATIONS is shown in Algorithm 4.4. For simplicity, we assume that when we assign a literal $\neg x$ to TRUE, it is the same as assigning x to FALSE and vice versa. Also, checking whether $\neg x$ is equal to TRUE is the same as checking whether x is equal to FALSE. In the algorithm, this is illustrated through the *assignment* array.

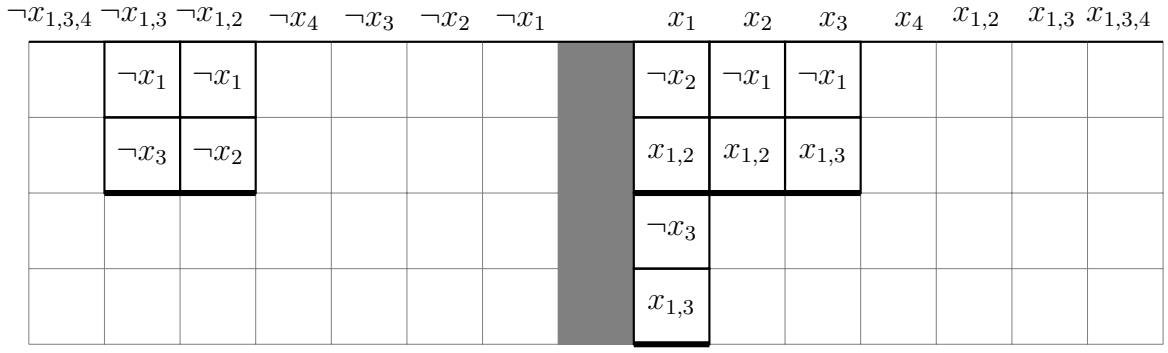


Figure 4.2: Lists of ternary implications. There is a delimiter between the tuples.

Even though these techniques can be generalized for variable-size clauses, this CNF module was devised for formulas containing relatively short OR-clauses. In this work, we are looking at problems that can be reduced to solving Boolean polynomial systems of small degrees. Indeed, all models considered in this thesis are systems of degree three at most. Thus, the size of the derived OR-clauses is at most four in all of our models.

Finally, Algorithm 4.5 shows the SET_IN_CNF function that was used in Algorithm 4.2

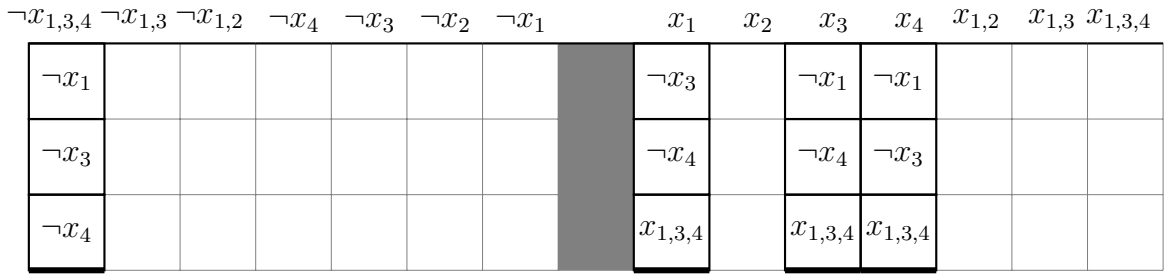


Figure 4.3: Lists of quaternary implications. There is a delimiter between the tuples.

Algorithm 4.4 Function ADD_TERNARY_IMPLICATIONS(l, *ternary_implications*) : Function for adding implied literals to the *CNF_propagation_stack*.

Input: A literal l and an array holding lists of ternary implications *ternary_implications*
Output: The *CNF_propagation_stack* is modified.

```
1: for each tuple in ternary_implications[l] do
2:     if (assignment[tuple[0]] ≠ TRUE) and (assignment[tuple[1]] ≠ TRUE) then
3:         if assignment[tuple[0]] = FALSE then
4:             add tuple[1] to CNF_propagation_stack.
5:         end if
6:         if assignment[tuple[1]] = FALSE then
7:             add tuple[0] to CNF_propagation_stack.
8:         end if
9:     end if
10: end for
```

as part of the synchronization between the models. Note that, if there exist N-ary implications with $N > 3$, they should be treated after line 10.

Algorithm 4.5 Function SET_IN_CNF(*to_set*, F) : Function that sets a list of literals to TRUE.

Input: A list of literals that need to be set to TRUE, the propositional formula F
Output: TRUE if formula is satisfiable, FALSE otherwise.

```
1: CNF_propagation_stack ← to_set.
2: while CNF_propagation_stack is not empty do
3:     l ← top element from CNF_propagation_stack.
4:     if assignment[l] ≠ TRUE then
5:         if assignment[l] = FALSE then
6:             return FALSE.
7:         end if
8:         assignment[l] ← TRUE.
9:         ADD_BINARY_IMPLICATIONS(l, binary_implications)
10:        ADD_TERNARY_IMPLICATIONS(l, ternary_implications)
11:    end if
12: end while
13: return TRUE.
```

What about clause learning?

Our solver is not CDCL-based. Intuitively, we do not believe that clause learning can speedup the solving process, as the CNF part of the models that we consider only serves

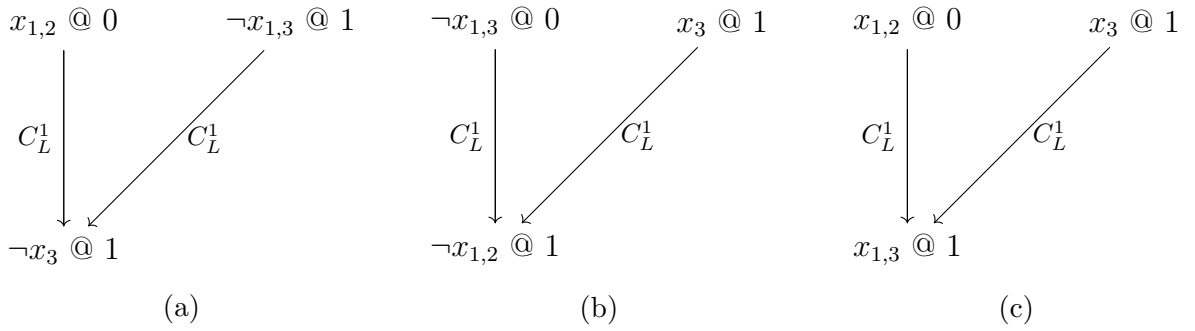


Figure 4.4: Implication graphs for the three cases where C_L^1 is used to imply a variable.

to describe the laws of multiplication in $\mathbb{F}_2$. Let us use Example 4.2.1 to analyse possible outcomes of the CDCL technique. This will allow us to check whether there is a case where a learned clause can help us propagate an assignment sooner than we would without applying CDCL techniques.

Recall that learned clauses are derived using the resolution operator on two other clauses. Applying this operator results in a *new* clause only if there is exactly one literal in the first clause, whose negation appears in the second clause. A first step in our analysis would be to find couples of clauses that satisfy this property and thus can be used to learn a new clause. Let us call a *block*, a subset of clauses that describe one monomial from the initial Boolean polynomial system. For instance, clauses C_1 to C_3 constitute a block, as well as clauses C_4 to C_6 and clauses C_7 to C_{10}. Then, two clauses from the same block can never satisfy the property that we need for the resolution operator. Indeed, the last clause in the block always has two literals that are the negation of the literals in one of the binary clauses. This leaves us with the possibility to combine clauses from two different blocks. This can be done, but only in the case where two blocks share a variable. For example, applying the resolution operator on clauses C_1 and C_6 results in a learned clause $C_L^1 : (\neg x_{1,2} \vee x_{1,3} \vee \neg x_3)$. This learned clause can be used to imply a variable in one of three cases, whose implication graphs are shown in Figure 4.4.

However, in all three cases the variables would have been implied even without the existence of the learned clause. Figure 4.5 shows alternative implication graphs for the three cases. We see that, in all cases, the same variable is implied using only clauses that are already in the formula before clause-learning techniques are used.

This example enumerates all possible outcomes of applying the resolution operator on clauses that are present in the initial formula. We have also tried combining learned clauses with each other or with original clauses. To date, we did not succeed in finding an example where a learned clause implies a variable that would not have been implied at the same decision level without clause learning techniques.

Open question 2 We have not yet explored the possibility of using clauses learned from conflicts in the XOR part of the CNF-XOR formula. It may be of interest to implement CDCL techniques in WDSAT for this case.

4.2.2 XORSET module

XORSET is a simple module for parity reasoning. In other words, this module performs unit propagation on XOR-clauses. The unit propagation rule can be informally defined as follows. When all except one literal in an XOR-clause are assigned, the remaining literal is given a truth value according to parity reasoning. Recall that an XOR-clause is satisfied if there is an odd number of literals that are set to TRUE. Equivalently, a conflict occurs

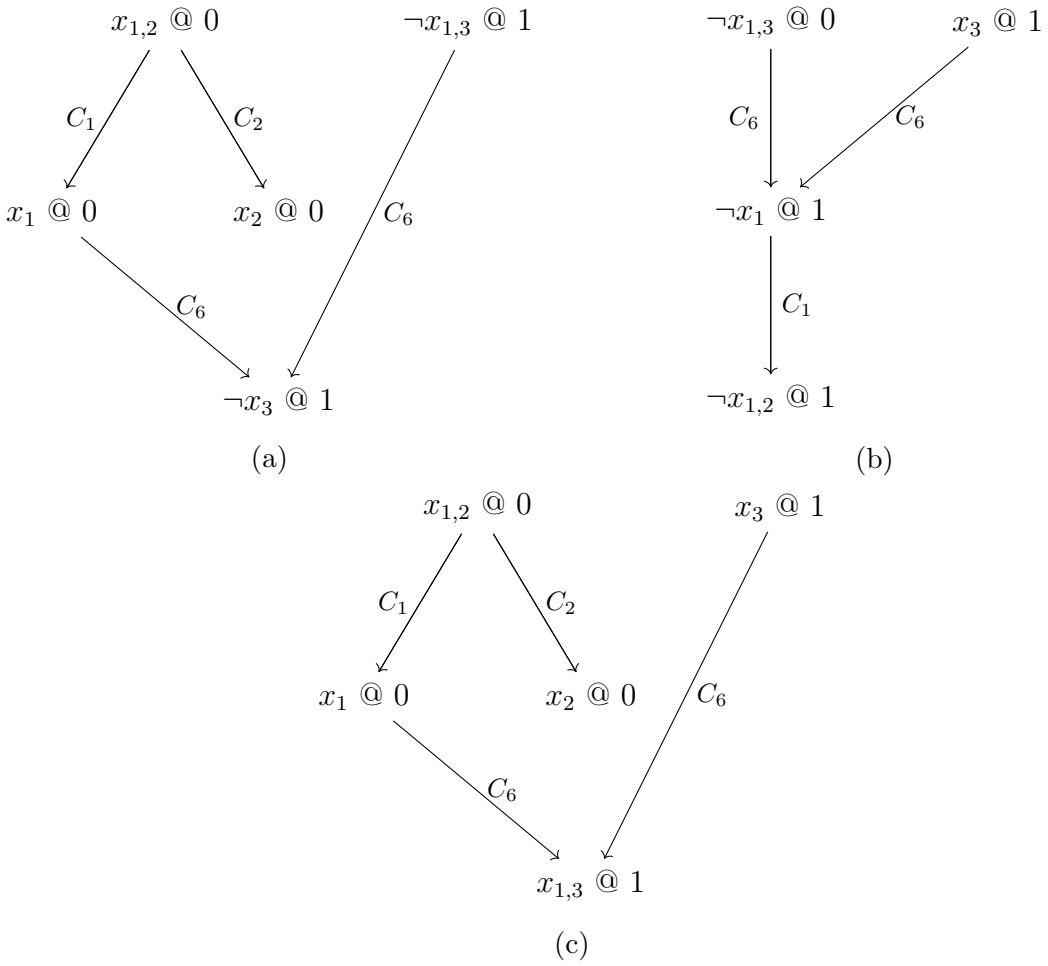


Figure 4.5: Implication graphs without learned clauses.

when all literals in an XOR-clause are assigned and an even number of them are set to TRUE.

During the solving process, we count the number of literals in a clause that are set to TRUE, and respectively the ones that are set to FALSE. In order to do this efficiently, we need to have quick access to the occurrences of each literal. At the implementation level, we manage a structure called *xorset_index* that keeps this information. This structure is an array indexed by both positive and negative literals that contains lists of clauses in which a literal appears. This is a classical technique for implementing basic XOR reasoning in a SAT solver.

Example 4.2.2 Let us consider the following set of four XOR-clauses numbered C_1^x to C_4^x.

$$C_1^x : \neg x_2 \oplus x_3$$
$$C_2^x : x_1 \oplus \neg x_2 \oplus x_4$$
$$C_3^x : x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$C_4^x : \neg x_2 \oplus \neg x_3 \oplus \neg x_4$$

Figure 4.6 shows the *xorset_index* structure corresponding to this formula.

$\neg x_4$	$\neg x_3$	$\neg x_2$	$\neg x_1$		x_1	x_2	x_3	x_4
C_4^x	C_4^x	C_1^x			C_2^x	C_3^x	C_1^x	C_2^x
		C_2^x			C_3^x		C_3^x	C_3^x
		C_4^x						

Figure 4.6: The *xorset_index* structure.

The SET_IN function for this module is detailed in Algorithm 4.6. In this algorithm, *degree_T* (resp. *degree_F*) is an array that holds counters of the number of literals set to TRUE (resp. FALSE) for each clause. Similarly, the array named *size* holds the size of each clause. Finally, we suppose that there is a function FIND_UNSET_VARIABLE() that takes a clause number and finds one variable from the clause that is not assigned with a truth value yet.

4.2.3 XORGAUSS module

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

$$x_1 \Leftrightarrow (x_2 \oplus x_3 \oplus ... \oplus x_n \oplus \top). \tag{4.1}$$

We can use this rule to define all equivalence classes. Thus, the initialization process of the XG module consists in performing the following steps for each XOR-clause : (i) put the

Algorithm 4.6 Function SET_IN_XORSET(to_set, F) : Function that sets a list of literals to TRUE.

Input: A list of literals that need to be set to TRUE, the propositional formula F
Output: TRUE if formula is satisfiable, FALSE otherwise.

```
1: XORSET_propagation_stack ← to_set.
2: while XORSET_propagation_stack is not empty do
3:     l ← top element from XORSET_propagation_stack.
4:     if assignment[l] ≠ TRUE then
5:         if assignment[l] = FALSE then
6:             return FALSE.
7:         end if
8:         assignment[l] ← TRUE.
9:         for each C in xorset_index[l] do
10:            increment degree_T[C]
11:            if degree_T[C] + degree_F[C] = size[C] − 1 then
12:                x ← FIND_UNSET_VARIABLE(C)
13:                if degree_T[C] is even then
14:                    add x to XORSET_propagation_stack.
15:                else
16:                    add ¬x to XORSET_propagation_stack.
17:                end if
18:            end if
19:        end for
20:        for each C in xorset_index[¬l] do
21:            increment degree_F[C]
22:            if degree_T[C] + degree_F[C] = size[C] − 1 then
23:                x ← FIND_UNSET_VARIABLE(C)
24:                if degree_T[C] is even then
25:                    add x to XORSET_propagation_stack.
26:                else
27:                    add ¬x to XORSET_propagation_stack.
28:                end if
29:            end if
30:        end for
31:    end if
32: end while
33: return TRUE.
```

clause in normal form, (ii) transform the clause into an EC and (iii) replace all occurrences of its representative in the system with the right side of the equivalence. Applying this transformation, we obtain a simplified system having the following property: a representative of an EC will never be present in another EC. Hence, the set of representatives and the set of literals on the right side of XOR-clauses are disjoint.

Example 4.2.3 Let us consider the following set of three XOR-clauses.

$$x_1 \oplus x_4 \oplus x_5 \oplus x_6$$
$$x_1 \oplus x_2 \oplus \neg x_4$$
$$x_2 \oplus x_3 \oplus \neg x_6$$

The steps of the initialization process of this formula are shown in Table 4.1. The left

column shows the set of equivalence classes that grows with each step. The right column shows the set of remaining XOR-clauses. We consider that all clauses are already put in normal form. This set becomes smaller as each clause is transformed into an equivalence class.

Table 4.1: Equivalence classes initialization steps.

Set of equivalence classes	Set of XOR-clauses
$\emptyset$	$x_1 \oplus x_4 \oplus x_5 \oplus x_6$ $x_1 \oplus x_2 \oplus x_4 \oplus \top$ $x_2 \oplus x_3 \oplus x_6 \oplus \top$
$x_1 \Leftrightarrow x_4 \oplus x_5 \oplus x_6 \oplus \top$	$x_2 \oplus x_5 \oplus x_6$ $x_2 \oplus x_3 \oplus x_6 \oplus \top$
$x_1 \Leftrightarrow x_4 \oplus x_5 \oplus x_6 \oplus \top$ $x_2 \Leftrightarrow x_5 \oplus x_6 \oplus \top$	$x_3 \oplus x_5$
$x_1 \Leftrightarrow x_4 \oplus x_5 \oplus x_6 \oplus \top$ $x_2 \Leftrightarrow x_5 \oplus x_6 \oplus \top$ $x_3 \Leftrightarrow x_5 \oplus \top$	$\emptyset$

We denote by k the number of variables in an XOR-CNF formula. At the implementation level, XOR-clauses are represented as $(k+1)$-bit vectors: a bit for every variable and one for a $\top$, $\bot$ constant. If a variable is present in the clause, the corresponding bit is set to 1, otherwise it is set to 0. Clauses are stored in an array indexed by the representatives. For a compact representation of the $(k+1)$-bit vector we used an array of $\lceil (k+1)/64 \rceil$ integers. This array is the core structure of the XG module and it will be referred to as the *EC* structure. For an example of the *EC* structure, see Figure 4.7 that illustrates the set of equivalence classes that we obtained through the transformation in Table 4.1. In this illustration, each line represents one equivalency and is labeled with the representative. The columns are colored in gray if and only if the corresponding variable belongs to the right side of the equivalency. The constant is referenced in the first column. The following is a detailed explanation of how this *EC* structure is used for performing

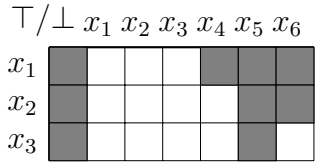


Figure 4.7: The *EC* structure.

GE and propagation of literals.

Let R be the set of representatives and C be the set of clauses. R and C hold the right-hand side and the left-hand side of all equations of type (4.1) respectively. We denote by C_x the XOR-clause in C that is equivalent to x. In other words, C_x is the right-hand side of the EC that has x as representative. Finally, we denote by $var(C_x)$ the set of literals (plus a $\top/\bot$ constant) in the clause C_x and $C[x_1/x_2]$ denotes the following substitution of clauses: for all $C_i \in C$ containing x_1, $C_i \leftarrow C_i \oplus x_1 \oplus x_2$, i.e. x_1 is replaced by x_2 in C_i. When we replace a literal x_1 by a clause C_{x_2}, we adopt a similar notation: $C[x_1/C_{x_2}]$.

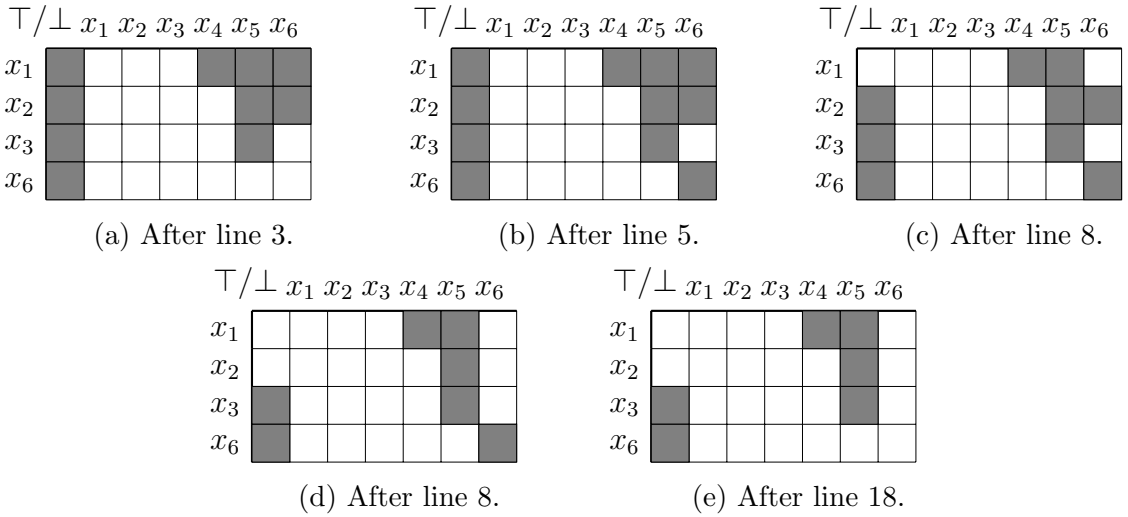
Thus, assigning a literal x_1 to $\top$ leads to using one of the rules in Table 4.2, depending on whether x_1 belongs to R or not. In both cases, propagation occurs when : $\exists\ x_i \neq x_1\ s.t.\ var(C_{x_i}) = \top/\bot$. Conflict occurs when one constraint leads to the propagation of x_i to $\top$ and another constraint leads to the propagation of x_i to $\bot$.

Table 4.2 presents inference rules for performing GE in the XG module of WDSat. Applying these rules allows us to maintain the property of the system which states that a representative of an EC will never be present in another EC. For clarity of the notation, the first column of this table contains the premises, the second one contains the conclusion and the third one is an update on the set R which has to be performed when the inference rule is used.

Table 4.2: Inference rules for the substitution of x_1 by a TRUE/FALSE constant.

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

Our chosen representation of equivalence classes allows us to implement these inference rules only by XOR-ing bit-vectors and flipping the clause constant. The first rule, whose pseudo-code is given in Algorithm 4.7, corresponds to the case where x_1 is not a representative. In a bit-vector from the *EC* structure, individual bits can be set to 0, set to 1 or their value can be checked. We distinguish variable bits from the constant bit. Other operations that modify the *EC* structure are FLIP_CONSTANT, used simply to inverse the value of the constant bit, and the operator $\oplus$ that denotes the XOR-ing of two bit-vectors. Lines in Algorithm 4.7 that contain operations that modify the *EC* structure are in bold. For a better understanding of the infer algorithm, we provide an execution example in Figure 4.8. In this example, we show the contents of the *EC* structure after the execution of each line in bold.



(a) After line 3. (b) After line 5. (c) After line 8.

(d) After line 8. (e) After line 18.

Figure 4.8: Setting x_6 to TRUE. Stream of changes on the *EC* structure after execution of the respective lines of Algorithm 4.7.

The infer function corresponding to the second inference rule, where x_1 is in the set

Algorithm 4.7 Function INFER_NON_REPRESENTATIVE(ul, tv, F) : Function that applies the first inference rule to the EC structure.

Input: Propositional variable ul, truth value tv, the propositional formula F
Output: The EC structure and the $XG_propagation_stack$ are modified.

```
1: add ul to R.
2: if tv =TRUE then
3:     FLIP_CONSTANT(EC[ul]).
4: end if
5: set ul to 1 in EC[ul].
6: for each r in R do
7:     if ul is set to 1 in EC[r] then
8:         EC[r] ← EC[r] ⊕ EC[ul].
9:         if all variable bits in EC[r] are set to 0 then
10:            if the constant bit in EC[r] is set to 1 then
11:                add r to XG_propagation_stack.
12:            else
13:                add ¬r to XG_propagation_stack.
14:            end if
15:        end if
16:    end if
17: end for
18: set ul to 0 in EC[ul].
```

of representatives, is detailed in Algorithm 4.8. In this code, we use a RESET_VECTOR function that simply sets all the bits in a given bit-vector to 0. The execution example for this algorithm is given in Figure 4.9.

Finally, we link everything together in the SET_IN function of the XG module, detailed in Algorithm 4.9. In this algorithm, we use the GET_PROPOSITIONAL_VARIABLE function that extracts the propositional variable from a literal and the GET_TRUTH_VALUE function that tells us whether l is a positive or a negative literal. For instance, GET_PROPOSITIONAL_VARIABLE($\neg x_1$) would return x_1 and GET_TRUTH_VALUE($\neg x_1$) would return FALSE.

4.3 Extending the XORGAUSS module

In this section, we show how we extend our XG module. First, we present the motivation for this work by giving an example of a case where GE in SAT solvers has certain limitations compared to algebraic GE. Secondly, we propose a solution to overcome these limitations and we implement it in our solver to develop the XORGAUSS-EXTENDED method (XG-EXT in short). To introduce new rules for this method, we use the same notation as in Section 4.2.3.

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

Algorithm 4.8 Function INFER_REPRESENTATIVE(ul, tv, F) : Function that applies the second inference rule to the EC structure.

Input: Propositional variable ul, truth value tv, the propositional formula F
Output: The EC structure and the $XG_propagation_stack$ are modified.

```
1: new_r ← CHOOSE_NEW_REPRESENTATIVE(EC[ul]).
2: add new_r to R.
3: EC[new_r] ← EC[new_r] ⊕ EC[ul].
4: RESET_VECTOR(EC[ul]).
5: if tv = ⊤ then
6:     FLIP_CONSTANT(EC[ul]).
7:     FLIP_CONSTANT(EC[new_r]).
8: end if
9: for each r in R do
10:    if new_r is set to 1 in EC[r] then
11:        EC[r] ← EC[r] ⊕ EC[new_r].
12:        if all variable bits in EC[r] are set to 0 then
13:            if the constant bit in EC[r] is set to 1 then
14:                add r to XG_propagation_stack.
15:            else
16:                add ¬r to XG_propagation_stack.
17:            end if
18:        end if
19:    end if
20: end for
21: set new_r to 0 in EC[new_r].
22: if all variable bits in EC[new_r] are set to 0 then
23:    if the constant bit in EC[new_r] is set to 1 then
24:        add new_r to XG_propagation_stack.
25:    else
26:        add ¬new_r to XG_propagation_stack.
27:    end if
28: end if
```

Example 4.3.1 We will use the following Boolean polynomial system of three equations to demonstrate a case where a cancellation of a term is missed by an XOR-enabled SAT solver.

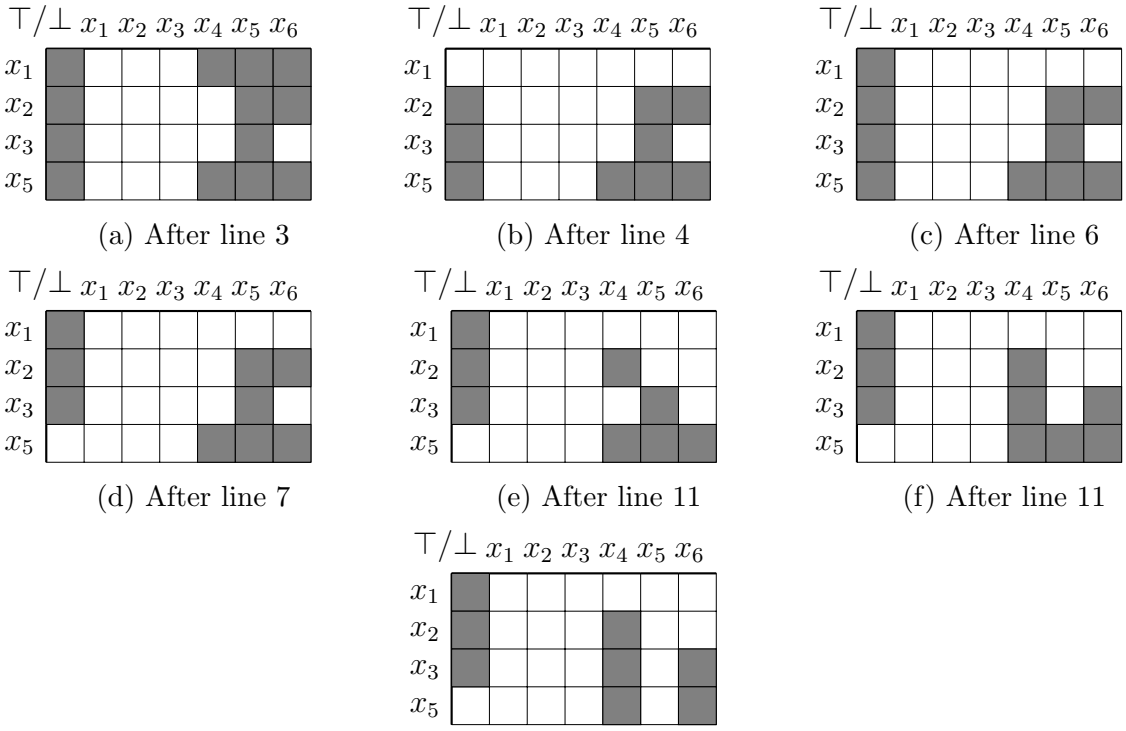
$$\begin{aligned} &\mathbf{x}_1 + \mathbf{x}_2\mathbf{x}_3 + \mathbf{x}_5 + \mathbf{x}_6 + 1 = 0 \\ &\mathbf{x}_3 + \mathbf{x}_5 + \mathbf{x}_6 = 0. \end{aligned} \tag{4.2}$$

Let us consider that in this system, we try to assign the value of 1 to $\mathbf{x}_2$. As the monomial $\mathbf{x}_2\mathbf{x}_3$ will be equal to 1 only if both terms $\mathbf{x}_2$ and $\mathbf{x}_3$ are equal to 1, the value of this monomial now depends only on the value of x_3. We get the following result:

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

After XOR-ing the two equations, we infer that $\mathbf{x}_1 = 1$.

The system in Equation (4.2) was already used in Chapter 3 as an example of transforming a Boolean polynomial system into a SAT model in CNF-XOR form. We recall here the derived CNF-XOR formula.



(a) After line 3 (b) After line 4 (c) After line 6

(d) After line 7 (e) After line 11 (f) After line 11

(g) After line 21

Figure 4.9: Setting x_1 to TRUE (x_5 is chosen as the new representative). Stream of changes on the EC structure after execution of the respective lines of Algorithm 4.8.

Algorithm 4.9 Function SET_IN_XG(to_set, F) : Function that sets a list of literals to TRUE.

Input: A list of literals that need to be set to TRUE, the propositional formula F
Output: TRUE if formula is satisfiable, FALSE otherwise.

```
1:  XG_propagation_stack ← to_set.
2:  while XG_propagation_stack is not empty do
3:      l ← top element from XG_propagation_stack.
4:      if assignment[l] ≠ TRUE then
5:          if assignment[l] = FALSE then
6:              return FALSE.
7:          end if
8:          assignment[l] ← TRUE.
9:          ul ← GET_PROPOSITIONAL_VARIABLE(l).
10:         tv ← GET_TRUTH_VALUE(l).
11:         if x1 ∈ R then
12:             INFER_REPRESENTATIVE(ul, tv, F).
13:         else
14:             INFER_NON_REPRESENTATIVE(ul, tv, F).
15:         end if
16:     end if
17: end while
18: return TRUE.
```

$$\begin{aligned} &\neg x_{2,3} \vee x_2 \\ &\neg x_{2,3} \vee x_3 \\ &\neg x_2 \vee \neg x_3 \vee x_{2,3} \\ &x_1 \oplus x_{2,3} \oplus x_5 \oplus x_6 \\ &x_3 \oplus x_5 \oplus x_6 \oplus \top. \end{aligned} \tag{4.3}$$

When we assign x_2 to $\top$ in the corresponding CNF-XOR clause in Equation (4.3), as per unit propagation rules, we get the following result:

$$\begin{aligned} &\neg x_{2,3} \vee x_3 \\ &\neg x_3 \vee x_{2,3} \\ &x_1 \oplus x_{2,3} \oplus x_5 \oplus x_6 \\ &x_3 \oplus x_5 \oplus x_6 \oplus \top. \end{aligned}$$

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

Note that $(\neg x_{2,3} \vee x_3) \wedge (\neg x_3 \vee x_{2,3})$ rewrites as $x_{2,3} \Leftrightarrow x_3$, but if the solver does not syntactically search for this type of occurrences regularly, $x_{2,3}$ will not be replaced by x_3. Moreover, this type of search adds an additional computational cost to the resolution.

Omissions as the one detailed in Example 4.3.1 can occur every time a variable is set to TRUE. As a result, we define the following inference rule with the goal to improve the performance of XOR-enabled SAT solvers:

$$\frac{x_1 \quad x_{1,2} \Leftrightarrow (x_1 \wedge x_2)}{x_{1,2} \Leftrightarrow x_2}. \tag{4.4}$$

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

$$\frac{x_1 \quad x_{1,\ldots,d} \Leftrightarrow (x_1 \wedge x_2 \wedge \ldots \wedge x_d)}{x_{1,\ldots,d} \Leftrightarrow (x_2 \wedge \ldots \wedge x_d)}. \tag{4.5}$$

Even though these rules are standard in Boolean logic, they are presently not implemented in XOR-enabled SAT solvers. Recall that when a solver takes as input an instance in CNF-XOR form, the second premise is lost or has to be inferred by syntactic search. To have knowledge of the second premise, the solver needs to read the instance in ANF. This is one of the reasons why we defined a new ANF input format for SAT solvers. Even though our solver can read formulas in CNF-XOR form, the extension of the XG can not be used in that case.

The following is a detailed explanation of how the rule in Equation (4.4) is applied in our implementation. Recall that the XG module has the following property: a representative of an EC will never be present in another EC. This property will be maintained in the XG-EXT method as well. Using the conclusion in Equation (4.4), we derive in Table 4.3 six inference rules that allow us to perform the substitution of a variable x_1 by a variable x_2 while maintaining the unicity-of-representatives property. Applying one of the inference rules in Table 4.3 can result in conflict or it can propagate a newly discovered truth value. Note that $var(C_{x_1} \oplus C_{x_2})$ is given by the symmetric difference $(var(C_{x_1}) \cup var(C_{x_2})) \setminus (var(C_{x_2}) \cap var(C_{x_1}))$.

Table 4.3: Inference rules for the substitution of x_1 by x_2.

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

This extension of the XG module is implemented as part of the SET_IN_XG function used in the ASSIGN algorithm. The adapted function SET_IN_XG is detailed in Algorithm 4.10 and the lines that were added in comparison to Algorithm 4.9 are in bold. Note our use of the *list_monomials* and *degree_monomials* structures. *list_monomials* is an array indexed by unary variables that contains the list of monomials in which a unary variable occurs. For instance, the monomial $\mathbf{x_1x_2}$ (or the conjunction $x_1 \wedge x_2$ in ANF) belongs both to the list of monomials of x_1 and the list of monomials of x_2. *degree_monomials* is an array indexed by non-unary variables and holds, what we informally call, the 'degree' of each variable. This degree is equal, at first, to the number of unary variables that compose the monomial. When each of the composing variables is set to TRUE, the degree decrements and serves as an indicator of when a monomial needs to be substituted by a unary variable. These structures are initialized when the solver reads the ANF formula.

Open question 3 Note that, in our current implementation, we only substitute monomials with unary variables. The case of substituting monomials by lower degree monomials should also be considered. We have not yet done empirical estimations on whether this

Algorithm 4.10 Function SET_IN_XG(to_set, F) : Function that sets a list of literals to TRUE.

Input: A list of literals that need to be set to TRUE, the propositional formula F
Output: TRUE if formula is satisfiable, FALSE otherwise.

```
1:  XG_propagation_stack ← to_set.
2:  while XG_propagation_stack is not empty do
3:      l ← top element from XG_propagation_stack.
4:      if assignment[l] ≠ TRUE then
5:          if assignment[l] = FALSE then
6:              return FALSE.
7:          end if
8:          assignment[l] ← TRUE.
9:          ul ← GET_PROPOSITIONAL_VARIABLE(l).
10:         tv ← GET_TRUTH_VALUE(l).
11:         if x_1 ∈ R then
12:             INFER_REPRESENTATIVE(ul, tv, F).
13:         else
14:             INFER_NON_REPRESENTATIVE(ul, tv, F).
15:         end if
16:         if ul is a unary variable then
17:             for each m in list_monomials[ul] do
18:                 if tv = TRUE then
19:                     decrement degree_monomials[m].
20:                     if degree_monomials[m] = 1 then
21:                         substitute m by ul in F.
22:                     end if
23:                 else
24:                     degree_monomials[m] ← −1.
25:                 end if
26:             end for
27:         end if
28:     end if
29: end while
30: return TRUE.
```

would yield faster running times, but we believe that the unswer to this would be specific to the properties of the model, notably its sparsity. For example, consider the monomial $\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ of degree three. When $\mathbf{x}_1$ is set to 1, $\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ can be substituted by $\mathbf{x}_2\mathbf{x}_3$. Whether computing such a substitution is of interest, depends mostly on the probability that the monomial $\mathbf{x}_2\mathbf{x}_3$ exists in the system and can potentially be canceled out.

4.4 Complexity discussion

The complexity of our solver strongly depends on the model and will be analyzed individually for each cryptosystem that we study in this thesis. When the underlying cryptosystem is considered as a black-box, the worst-case complexity of WDSat is exponential in the number of unary variables. This result is mainly due to our branching policy. In contrast, the complexity of SAT solvers that use the CNF-XOR form is exponential in the number of, both unary, and substitution variables.

Since the solution usually has a balanced distribution over the domain $\{0, 1\}$, every possible assignment of variables has an equal chance to be the right solution. Whether this solution is found at the beginning of our binary search or toward the end is completely arbitrary. Thus, we have no reason to believe that the average complexity of the solver will differ from the worst-case complexity by more than a factor of two. This assessment coincides with our experimental results for cases where at least one solution exists. Note however that, the same reasoning can not be applied for solvers that use the CNF-XOR form. When variables are intertwined by more than XOR constraints, there can be a propagation stream that occurs on each branch and thus the height of the binary search tree can be inherently smaller on average.

Part II

Cryptographic applications

Chapter 5

ECDLP Preliminaries

The discrete logarithm problem (DLP) is one of the most studied hard problems in the last few decades. The huge interest in this problem comes from its versatile nature that has allowed it to be one of the fundamental building blocks of modern public-key cryptography. The DLP can be defined for every multiplicative group G. Specifically, when G is a cyclic group or order N generated by g, and h is an element in G, the DLP consists in finding, an integer x, such that $h = g^x$.

In this work, we are interested in the case where G is the group of points on an elliptic curve. The purpose of this chapter is to provide an explanation of the basic notions needed to understand the cryptographic attacks presented in this thesis. Secondly, we present a quick survey of state-of-the-art attacks, as well as practical uses of the elliptic curve discrete logarithm problem (ECDLP). Neither of these is an exhaustive presentation of the ever-growing literature on this topic.

Let $K = \mathbb{F}_q$ denote a finite field. An elliptic curve over K is given by a Weierstrass equation, i.e. an equation of the form:

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3,$$

with $a_1, a_2, a_3, a_4, a_6 \in K$. We generally write the Weierstrass equation for the elliptic curve using non-homogeneous coordinates $x = X/Z$ and $y = Y/Z$

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

The addition law of two points on an elliptic curve is defined geometrically. Having two points P and Q on the curve E, first, we draw a line through them. According to a special case of Bezout's theorem (see [Har77]), the line intersects the curve E at a third point R. Then we draw a vertical line through R. This line intersects E in another point, and we define this point to be the sum of P and Q. However, when we add a point P to itself, the first step can not be carried out as described, so instead, we draw the tangent line to E at P. Figure 5.1 is an illustration of performing the operation $P + P$ on an elliptic curve in Weierstrass form. Note that, for characteristic different from 2, since the curve is symmetrical on the x-axis, the inverse of a point P is the reflecting point across the x-axis. Since the vertical line through P and $-P$ does not intersect E, we must include an extra point on the curve to define $P + (-P)$. This is called the point *at infinity* and is denoted by $\mathcal{O}$.

The addition law on E has the following properties:

(a) If a line L intersects E at the (not necessarily distinct) points P, Q, R, then $(P + Q) + R = \mathcal{O}$.

(b) $P + \mathcal{O} = P$, for all $P \in E$.

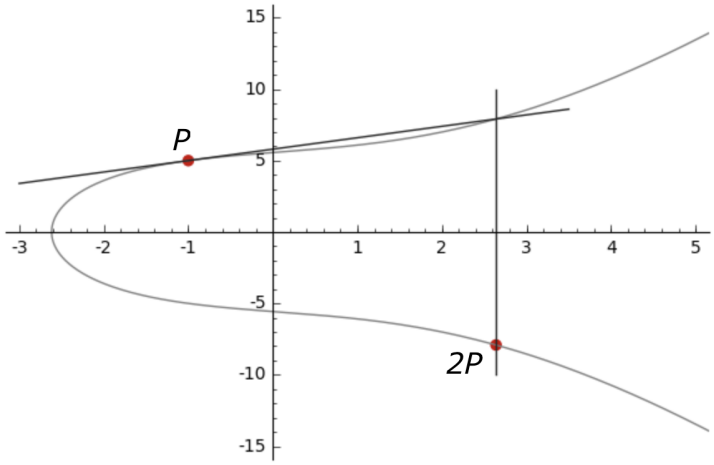


Figure 5.1: Addition law on E. The case of doubling a point.

(c) $P + Q = Q + P$, for all $P, Q \in E$.

(d) Let $P \in E$. There is a point of E, denoted by $-P$, satisfying $P + (-P) = \mathcal{O}$.

(e) $P + (Q + R) = (P + Q) + R$, for all $P, Q, R \in E$.

Let E be an elliptic curve defined over a field of characteristic different from 2 and 3. E can be described as an algebraic curve given in short Weierstrass form

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

We can write down explicitly the formulas for the addition law on E as follows. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, then $P_1 + P_2 = (x_3, y_3) = (\lambda^2 - x_1 - x_2, \lambda(x_3 - x_1) + y_1)$, where

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{when } P_1 \neq P_2 \\ \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{when } P_1 = P_2. \end{cases}$$

For more details, the reader can refer to [Sil86].

The set of points on an elliptic curve, including the point at infinity, forms a finite abelian group. In this group, we consider the Elliptic curve discrete logarithm problem (ECDLP), which is formulated as follows.

Definition 5.0.1 Given points $P, Q \in E(K)$, find an integer x such that $xP = Q$.

Elliptic curves over various fields can be used to define the ECDLP, serving as a one-way function for cryptographic purposes. The main advantage of using elliptic curves is that for well-chosen parameters, generic attacks are best known attacks on the ECDLP. As a result, smaller key sizes can be used for elliptic curve cryptosystems, compared to previously used systems such as RSA or systems based on the discrete log on finite fields. The elliptic curve arithmetic comes with some computational cost, but the reduction in key sizes is sufficient enough to compensate for this cost and make elliptic curve cryptosystems more efficient.

When q is a large prime, we are working with curves over prime fields. When $q = p^n$, curves defined over K are elliptic curves over extension fields. In the special case where

$p = 2$, we say that E is an elliptic curve over a binary extension field or a binary elliptic curve. This is also referred to as the special case of characteristic 2. For binary curves, it is common to adopt the following form

$$E : y^2 + xy = x^3 + ax^2 + 1. \tag{5.1}$$

Note that, since $a \in \{0, 1\}$, there are only two curves defined using Equation (5.1). They are referred to as Koblitz curves [Kob92b] and they are extensively used in cryptography, since they allow fast scalar multiplication and point counting. For security reasons, it is better to choose n prime, but on the other hand, there are computational advantages when n is composite.

5.1 Applications in cryptography

In this section, we detail some of the most notable applications of the discrete log problem. As this is a chapter on ECDLP, we will use the elliptic curve notation, presented in the previous section. Other applications that are not described in this section include, for example, Paillier's encryption [Pai99] which is additively homomorphic, the elliptic curve analog of the Digital Signature Algorithm [NIS92] and zero-knowledge proofs used as authentication protocols, such as the Schnorr's protocol [Sch90].

5.1.1 Diffie-Hellman key exchange

The Diffie-Hellman key exchange, introduced in 1976 [DH76], is the first application that popularized the use of the discrete logarithm in cryptography. The initial proposition of this protocol used $\mathbb{F}_p^*$ and was later generalized for elliptic curve groups. This method covers the need for two communicating parties to agree on a shared secret key, which is the core purpose of public-key cryptography. The shared secret key is to be used for communication via a symmetric cryptosystem.

The Diffie-Hellman (DH) key exchange is considered to be the foundation of public-key cryptography and as such, it has been illustrated many times before. Thus, in this chapter, we will rather use the color metaphor to show an illustration of the protocol in Figure 5.2. Indeed, the set of colors and the operation of color mixing form an abelian semigroup. We also have a one-way function, as mixing colors is easy, but finding the exact original colors from a mixed one is hard[1].

When Alice and Bob want to agree on a shared secret, they need to first agree on public parameters, or more precisely, on a cyclic group and a generator of that group. In the Elliptic curve Diffie-Hellman (ECDH) protocol, they have to choose an elliptic curve E and a point $P \in E$. It is advised that P is of prime order equal to the order of E. These parameters are public. Then, Alice chooses a secret $a \in \mathbb{Z}/N\mathbb{Z}$ and computes the point $A = aP$, which is sent to Bob via a public communication channel. Bob performs the same operation, choosing $b \in \mathbb{Z}/N\mathbb{Z}$, computing $B = bP$, and sending it to Alice. After the exchange, they can both compute $S = bA = aB = abP$. Note that, the fact that the points on E form an abelian group is a crucial component in this protocol.

The elements that an eavesdropper Eve can obtain from the communication channel are A and B. Also, P and E are publicly known. Having $A, P \in E$ and finding the secret a is a discrete log problem. Thus, for the protocol to be considered secure, the discrete log problem must be hard in E. There exist other problems that arise from the DH key exchange. The Computational Diffie-Hellman problem (CDH) is defined as follows. Given

[1] Of course, this does not stand for numerical representations of colors.

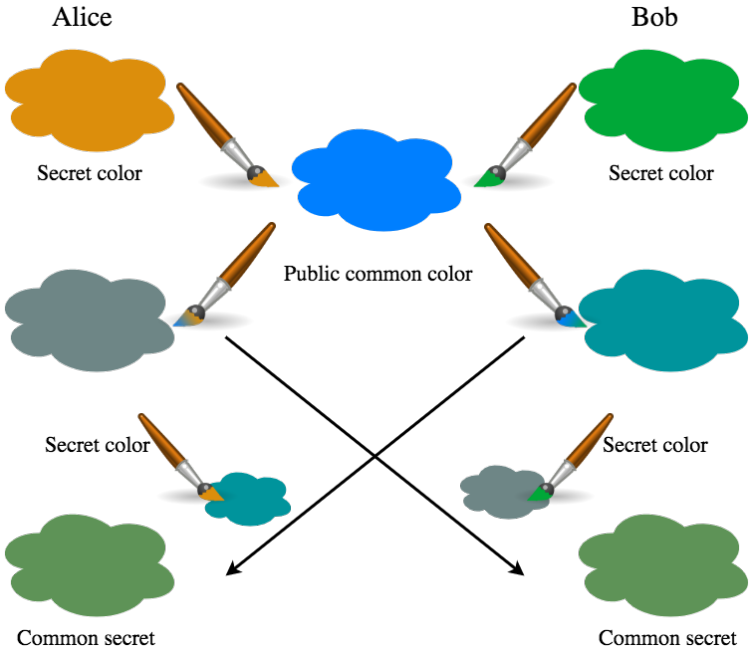


Figure 5.2: Diffie-Hellman key exchange with colors.

P, aP and bP, find abP. Solving this problem would allow an attacker to compute the secret shared key K, without knowing Alice's secret a or Bob's secret b. Then, there is the Decisional Diffie-Hellman problem (DDH). Given P, aP, bP and Q, determine if $Q = abP$. In the general case, solving these problems requires solving the ECDLP.

The ECDH as we have described it is not resistant to a *man-in-the-middle* attack. This is an active attack where Eve intercepts the communication between Alice and Bob and acts as one party for the other. Another requirement that the basic implementation of this protocol does not meet is perfect forward secrecy. In other words, if Eve finally succeeds in finding the secret key S, she will be able to decrypt all past communications. To prevent this detrimental result, the Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) protocol is used in practice. In this case, S is discarded after the communication is done and if Alice and Bob need to communicate in the future, they have to agree upon a new secret key.

5.1.2 ElGamal encryption

In 1984, ElGamal [ElG85] showed how the DH key exchange can be used to construct an encryption scheme that relies on the difficulty of the discrete log. A first notable difference between performing the DH key exchange followed by a symmetric encryption scheme and the ElGamal encryption is that the latter does not require both communicating parties to be online simultaneously. Instead, Alice generates a pair of public and private key, and then she publishes her public key to be used by anyone that wants to send her an encrypted message. The key generation process is the same as the first step in the ECDH key exchange. Alice chooses $a \in \mathbb{Z}/N\mathbb{Z}$, which will be her secret key, and computes $A = aP$ which will be her public key. In practice, Alice does not actually *choose* her secret key, but instead, she computes it using a cryptographically secure pseudorandom number generator. Thus, the security of the protocol depends strongly on the security of this generator. ElGamal also proposed a signature scheme based on the DH key exchange, that starts with this key generation process.

When Bob wants to send a message to Alice, he can just look up her public key and

perform his part of the key agreement. Specifically, he chooses a secret $b \in \mathbb{Z}/N\mathbb{Z}$ and computes $B = bP$ and $S = bA$. He can now use S to encrypt his message using an agreed-upon symmetric cryptosystem, and then send the encrypted message along with B to Alice. This is known as noninteractive ECDH . The ElGamal encryption scheme is similar, but instead of using symmetric encryption, the message is first mapped to a point $M \in E$ using an invertible map that is also agreed-upon. The encrypted message is computed as $C = (M + S)$, with $C \in E$. The decryption process goes as follows. Alice receives (B, C) and uses her secret key to compute $S = aB$. She then computes $M = C + (-S)$ and uses the map inverse to find the message.

5.1.3 Pairing-based cryptography

A significant extension of elliptic curve cryptography is pairing-based cryptography, which uses bilinear pairings to construct protocols. A bilinear pairing is an efficiently-computable map $e : G_1 \times G_2 \to G_T$, where G_1 and G_2 are additive groups and G_T is a multiplicative group. These three groups are of equal prime order N. For simplicity, let us consider the case of symmetric bilinear pairings, which is the case where $G_1 = G_2$. Thus, we simply use G to denote the source group. For cryptographic applications, G is usually the group of points on an elliptic curve and thus we will continue with the notation from previous sections. We consider that e is an admissible bilinear map if we have the following properties.

1. Bilinearity: given $R, S \in G$ and $a, b \in \mathbb{Z}_N^*$, $e(aR, bS) = e(R, S)^{ab} = e(abR, S) = e(R, abS)$.

2. Non-degeneracy: there exist $R, S \in G$, such that $e(R, S) \neq 1_{G_T}$, where 1_{G_T} is the identity of the group G_T.

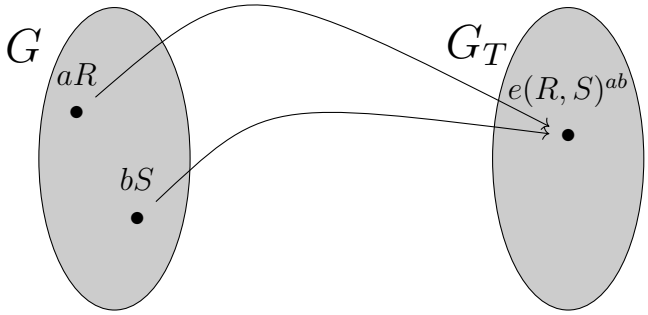


Figure 5.3: A symmetric bilinear pairing.

These primitives find many applications in cryptography, both in protocol construction and in cryptanalysis. As an example, we are going to describe the three-party one-round key exchange. This protocol, proposed by Joux in 2000 [Jou00], is the first key exchange between more than two users that does not require more than one round of interaction between the users. The protocol works as follows. When Alice, Bob, and Charlie need to agree on a shared secret, they respectively choose secret a, b, and c. Then, they respectively compute and send aP, bP, and cP to the other parties. Finally, the following keys are computed.

- Alice: $K_A = e(bP, cP)^a$;
- Bob: $K_B = e(aP, cP)^b$;

- Charlie: $K_C = e(aP, bP)^c$;

As per the properties of bilinear pairings, we have

$$K = K_A = K_B = K_C = e(P, P)^{abc}.$$

Another notable application of pairing-based cryptography is the identity-based encryption scheme [BF01], in which the public key of a user is linked to some identifiable information about the user, such as an email address. Bilinear pairings have also found their use in group and ring signature schemes.

5.2 Known attacks

The goal of this section is to give a general understanding of the state-of-the-art attacks on the ECDLP. We divide known attacks into generic and specialized attacks. We mention some of the most prominent examples in both categories while concentrating on a complexity discussion and on implications that these attacks have on selecting secure elliptic curves.

5.2.1 Generic attacks

We call *generic* an attack that can be applied to every cyclic group G. In this case, we can view G as a black-box, with the group law being the only known element to the attacker. For the attacks mentioned here, we also consider that we can easily compute the order of G in practice, which is usually the case for cyclic groups used in cryptography.

The most straightforward generic attack is the exhaustive search method. For the ECDLP, the exhaustive search consists in choosing, randomly, or in a predetermined order, an element $a \in \mathbb{Z}$ and computing $R = aP$, until we find the case where $R = Q$ and conclude that $x = a$. Among all algorithms mentioned here, this algorithm has the best memory complexity of $O(1)$, and of course, the worst time complexity of $O(N)$, both in the worst and average case.

The Pohlig-Hellman algorithm, introduced in 1978 [PH78] is useful when we know the factorization of N. In fact, factoring the order N is practical for all elliptic curve groups used in cryptography, as their order is small compared to numbers that we can factor in practice. Groups of orders that are factors of N are subgroups of G. The Pohlig-Hellman algorithm allows us to reduce the DLP in G to the DLP in a subgroup of G. The consequence of this reduction is that the DLP of G is only as hard as the DLP of the largest subgroup of prime order of G. The practical implication is that, for strong security, we need to select curves such that N is either prime or has a very large prime factor.

The earliest algorithm that has exponential complexity in the square root of N is the Baby-step giant-step algorithm, proposed by Shanks in 1971. The main idea behind this method is to apply the divide-and-conquer strategy to solving the discrete log. This idea is dependent on the observation that the unknown x can be written as $x = \lambda_0 + \lambda_1 \lfloor \sqrt{N} \rfloor$, where both λ_0 and λ_1 are lesser than $\sqrt{N}$. Then, one can compute and store all pairs (i, P_i), for $0 \leq i < \sqrt{N}$, where $P_i = iP$. These are called baby-steps and require a total of $O(\sqrt{N})$ computations. Similarly, the giant-steps consist in computing pairs (j, Q_j), for $0 \leq j < \sqrt{N}$, where $Q_j = Q - j\lfloor \sqrt{N} \rfloor P$. For efficiency, $\lfloor \sqrt{N} \rfloor P$ can be computed only once at the beginning. As we compute Q_j values, we compare them to the stored P_i values, looking for a match. When we find $Q_j = P_i$, we have than $\lambda_0 = i$ and $\lambda_1 = j$. Thus, we compute the discrete log as $x = i + j\lfloor \sqrt{N} \rfloor$. Computing all baby-steps and giant-steps requires a total of $O(\sqrt{N})$ computations, both in the worst and the average case. Thus,

this algorithm is the one that has the best time complexity in the sequential processing setting. Its main drawback is that we have to store all values from the baby-steps, which makes the memory complexity $O(\sqrt{N})$, as well.

Pollard's rho method [Pol78] offers a solution with a complexity comparable to the Shanks's algorithm, but with negligible memory requirements. Similarly to the baby-step giant-step algorithm, Pollard's algorithm is based on a collision search. This time, the goal is to find, for any point R on E, two different linear combinations of P and Q. Namely, when we have $R = a_1P + b_1Q = a_2P + b_2Q$, x can be computed as

$$x = \frac{a_1 - a_2}{b_2 - b_1}.$$

This method will be explained in greater detail in Chapter 6, as its parallel version is the main point of interest in that chapter. The parallel version of Pollard's rho method is due to van Oorschot and Wiener [vOW99], who proposed an efficient way to parallelize the collision search while compromising on the memory requirements. This algorithm, referred to as the Parallel Collision Search (PCS), is the universal choice for practical discrete log computations. Unlike the baby-step giant-step, Pollard's algorithm and its parallel variant are both probabilistic and the estimate on the time complexity is based on the birthday paradox.

In summary, these three algorithms have a time complexity that is exponential in the square root of N. Coupled with the Pohlig-Hellman algorithm, the complexity of an attack, using these algorithms, is exponential in the square root of p, where p is the largest prime factor of N. Thus, the practical implication of these algorithms is that for b-bit security requirements[2], one needs to use elliptic curves where $p > 2b$. One needs to also consider that the PCS parallelizes perfectly and thus, its time complexity is reduced by a linear factor of L when there are L computing resources available for the attack.

The ECDLP is in the NP class. However, the ECDLP is also in the BQP class, which stands for Bounded-error Quantum Polynomial time. Algorithms in this class are believed to be solvable in polynomial time by a quantum computer, with an error probability of at most $\frac{1}{3}$ for all instances. This theoretical result is due to Shor [Sho97], who proposed an efficient quantum algorithm for the DLP in 1997. This algorithm is in the generic family and thus, the security of elliptic curve cryptography is compromised in the quantum paradigm.

5.2.2 Attacks on specific families

When we know more about the underlying group structure of G, we can find some weaknesses that can be exploited to develop a stronger attack on the DLP in G. In this section, we mention some of these special attacks on elliptic curves. A recurring theme in these attacks is the transfer of the DLP in G to a DLP in another group, where it is easier to solve. This is actually the earliest application of bilinear pairings that were described in Section 5.1.3. In the MOV reduction [MOV93], for example, we use a bilinear pairing $e : G \times G \to G_T$ to transform the discrete log in G, to a discrete log in G_T. Recall that, as per the bilinear property, we have

$$e(P, Q) = e(P, xP) = e(P, P)^x.$$

Thus, x is the discrete log of $e(P, Q)$ to the base $e(P, P)$. This discrete log is easier, as it is computed in G_T, where $G_T \subseteq \mathbb{F}_{q^k}$. The MOV attack uses the Weil pairing and it was later generalized using the Tate pairing [FR94].

[2]Having a b-bit security means that the complexity of the best attack is equivalent to a brute force attack on a b-bit key, i.e. 2^b.

Another family of curves that are susceptible to special attacks are anomalous curves. These are curves over $\mathbb{F}_p$ whose order is equal to p. The DLP on an anomalous curve $E(\mathbb{F}_p)$ can be reduced to the DLP in the additive group of integers modulo p, and as a result, it can be computed in polynomial time with extended Euclid algorithm.

A reduction to the multiplicative group of a finite field ($\mathbb{F}_q^*$) is interesting due to the index calculus algorithm, which is subexponential in this group. Because of its success in $\mathbb{F}_q^*$, there has been tremendous activity in the application of this method for elliptic curve groups. To date, the index calculus is in the family of special attacks, as it can be applied only for elliptic curves over extension fields. Our work is widely concentrated on this attack, and thus, it will be described in detail in Chapter 7.

Considering special attacks, it is clear how important the choice of parameters is for the construction of secure cryptosystems. In practice, we use elliptic curves that are recommended by a standardization authority. For instance, NIST defines 15 standard curves, 10 of which are binary.

Chapter 6

Parallel Collision Search

This chapter is dedicated to our work on the parallel collision search (PCS) algorithm, proposed by van Oorschot and Wiener [vOW99], which is a generic attack on the discrete log that is most commonly used in practice. Unlike other attacks in this thesis, the PCS does not use a SAT solver. Most of the contributions of this work are presented in [TID17].

Let us first introduce the problem and mention some previous work. Given a function $f : S \to S$ on a finite set S, we call collision any pair a, b of elements in S such that $f(a) = f(b)$. Collision search has a broad range of applications in the cryptanalysis of both symmetric and asymmetric ciphers: computing discrete logarithms, finding collisions on hash functions and meet-in-the-middle attacks. Pollard's rho method [Pol78], initially proposed for solving factoring and discrete logs, can be adapted to find collisions for any random mapping f. The parallel collision search algorithm, proposed by van Oorschot and Wiener [vW99], builds on Pollard's rho method, and is expected to have a linear speedup compared to its sequential version. This algorithm computes multiple walks within a parallel processing context and stores some of these points, called distinguished points.

In our work, we revisit the memory complexity of the parallel collision search algorithm, both for applications that need a small number of collisions (i.e. discrete logs) and those needing a large number of collisions, such as meet-in-middle attacks or the multiple DLP. In the case of discrete logarithms, collision search methods are the fastest known attacks in a generic group. In elliptic curve cryptography, subexponential attacks are known for solving the discrete log on curves defined over extension fields, but only generic attacks are known to work in the prime field case. Evaluating the performance of collision search algorithms is thus essential for understanding the security of curve-based cryptosystems. Several record-breaking implementations of this algorithm are available in the literature: over a prime field the current record reaches a discrete log in a 112-bit group on a curve of the form $y^2 = x^3 - 3x + b$ [BKK+12; JWB09]. This computation was performed on a Playstation 3. More recently, Bernstein, Lange and Schwabe [BLS11] reported on an implementation on the same platform and for the same curve, in which the use of the negation map gives a speed-up by a factor $\sqrt{2}$. Over binary fields, the current record is an FPGA implementation breaking a discrete logarithm in a 117-bit group [BEL+]. As for the meet-in-the-middle attack, this generic technique is widely used in cryptanalysis to break block ciphers (double and triple DES, GOST [Iso11]), hash functions [KNW09; MRST09], lattice-based cryptosystems (NTRU [HGSW03; vV16]) and isogeny-based cryptosystems [ACC+18].

Two models of computation can be considered for this algorithm. The first one follows the shared memory paradigm, in which each thread will compute distinguished points and store it in the common memory. The second one is a message-passing model, where the

threads computing points, called the clients, send the distinguished points to a separate process, running on a different machine called the server, who will handle the memory and check for collisions.

First, our contribution is to extend the analysis of the parallel collision search algorithm and present a formula for the expected runtime to find any given number of collisions, with and without a memory constraint. We show how to compute optimal values of θ - the proportion of distinguished points, allowing to minimize the running time of collision search, both in the case of discrete logarithms and meet-in-the-middle attacks. In the case where the available memory is limited, we determine the optimal value of θ, proving that the value conjectured by van Oorschot and Wiener was asymptotically correct. Going further in the analysis, our formulas show that the actual running time of finding-many-collisions algorithm is critically reduced if the number of words w that can be stored in memory is larger. A notable application that requires finding many collisions is the DLP in the multi-user setting. We describe existing algorithms for this setting [KS01; FJM14b] and present some new insights.

Secondly, we focus on the data structure used for the algorithm. To the best of our knowledge, all existing implementations of parallel collision search algorithms use hash tables to organize memory and allow fast look-up operations. In this chapter, we introduce a new structure, called Packed Radix-Tree-List (PRTL), which is inspired by radix trees. We show that the use of this structure leads to a better use of memory in implementations and thus yields improved running times for many-collision applications.

Using the PRTL structure, we have implemented the parallel collision search algorithm for discrete logarithms on elliptic curves defined over prime fields and experimented using a Shared-Memory Parallelism (SMP) system. Our benchmarks demonstrate the performance and scalability of this method. While in the case of a single discrete log, the PRTL variant implementation yields running times similar to those of a hash table approach, our experiments demonstrate that the new data structure gives faster limited-memory multi-collision attacks.

This chapter is organized as follows. Section 6.1 reviews algorithms for solving the discrete logarithm problem and for meet-in-the-middle attacks. In Section 6.2, we revisit the proof for the time complexity of the collision finding algorithm for a small and a large number of collisions. Furthermore, we show how to minimize the runtime, as a function of the proportion of distinguished points. Section 6.3 shows our theoretical findings on the multiple DLP problem. Section 6.4 describes our choice for the data structure and memory complexity estimates. Finally, Section 6.5 details the implementation of our data structure and a comparison with classic hash tables. We also give implementation details of our attack and presents our experimental results.

6.1 Background

In this section, we briefly review Pollard's rho method and the parallel algorithm for searching collisions. Let S be a finite set of cardinality N. In order to look for collisions for a function $f : S \to S$ with Pollard's rho method, the idea is to compute a sequence of elements $x_i = f(x_{i-1})$ starting at some random element x_0. Since S is finite, eventually this sequence begins to cycle and we therefore obtain the desired collision $f(x_k) = f(x_{k+t})$, where x_k is the point in the sequence before the cycle begins and x_{k+t} is the last point on the cycle before getting to x_{k+1} (hence $f(x_k) = f(x_{k+t}) = x_{k+1}$). The sequence defines a walk in the shape of the letter ρ (see Figure 6.1), thus the name of the method. One may show that the expected number of steps taken until the collision is found is $\sqrt{\frac{\pi N}{2}}$, and

therefore that the memory complexity is also $O(\sqrt{\frac{\pi N}{2}})$. This algorithm can be further optimized to constant memory complexity by using Floyd's cycle-finding algorithm [Jou09; Bre80]. We do not further detail memory optimizations here since they are inherently of sequential nature and there is currently no known way to exploit these ideas in a parallel algorithm.

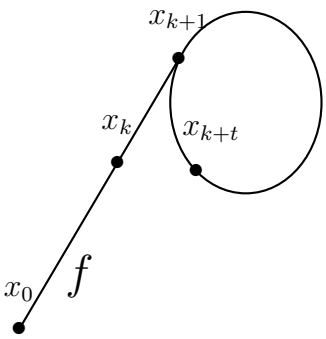


Figure 6.1: Pollard's rho walk.

The parallel algorithm for collision search proposed by van Oorschot and Wiener [vOW99] assigns to each thread the computation of a trail given by points $x_i = f(x_{i-1})$ starting at some point x_0. Only points that belong to a certain subset, called the set of distinguished points, are stored. This set is defined by points having an easily testable property. The proportion of distinguished points in the set is denoted by θ. Whenever a thread computes a distinguished point x_d, it stores it in a common list of tuples (x_0, x_d). If two walks collide, this is identified when they both reached a common distinguished point (see Figure 6.2). We may then re-compute the paths and the points preceding the common point are distinct points that map to the same value.

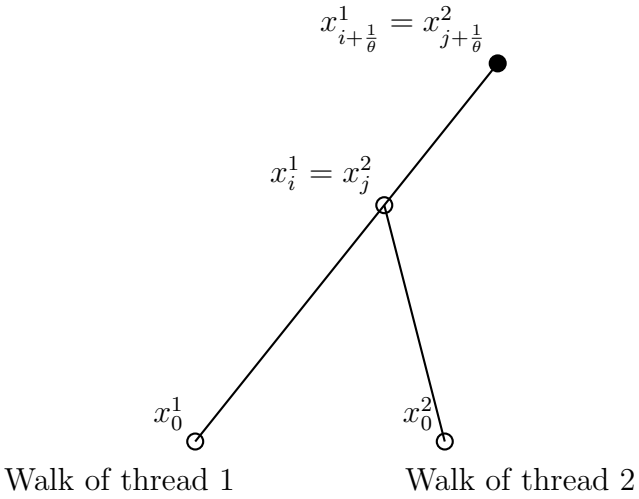


Figure 6.2: Collision search by thread 1 and thread 2 starting from point x_0^1 and point x_0^2 respectively. Nondistinguished points are transparent.

6.1.1 Solving discrete logarithms.

In this subsection, S denotes a cyclic group of order N. We focus on the elliptic curve discrete logarithm problem (ECDLP) in a cyclic group $S = \langle P \rangle$, but the methods described in this thesis apply to every finite cyclic group. We will assume that the curve E is defined

over a finite field $\mathbb{F}_p$, where p is a prime number. Let $Q \in S$ and say we want to solve the discrete logarithm problem $Q = xP$, where $x \in \mathbb{Z}$. To apply the ideas explained above, we define a map $F : S \to S$ which behaves randomly and such that each time we compute $f(R)$ we can easily keep track of integers a and b such that $f(R) = aP + bQ$. Pollard's initial proposal for such a function was

$$f(R) = \begin{cases} R + P & \text{if } R \in S_1 \\ 2R & \text{if } R \in S_2 \\ R + Q & \text{if } R \in S_3, \end{cases} \tag{6.1}$$

where the sets S_i, $i \in \{1, 2, 3\}$ are pairwise disjoint and give a partition of the group S. As a consequence, whenever a collision $f(R) = f(R')$ occurs, we obtain an equality

$$aP + bQ = a'P + b'Q. \tag{6.2}$$

This allows us to recover $x = (a - a')/(b' - b)$, provided that $b' - b$ is not a multiple of N. Starting from R_0, a multiple of P, Pollard's rho [Pol78] method computes a sequence R_i of points where $R_{i+1} = f(R_i)$. Since the group S is finite, this sequence will produce a collision after $\sqrt{\frac{\pi N}{2}}$ iterations on average. To define distinguished points, we take an easily testable property, such as a certain number of trailing bits of their x-coordinate being zero. Whenever a walk computes such a point, this is stored in a common list, together with the corresponding a and b. When two walks collide, this cannot be identified until the common distinguished point is computed. Then the discrete logarithm can be recovered from (6.2).

6.1.2 Many collision applications : the multi-user setting

A first type of application of the van Oorschot and Wiener algorithm computing many collisions is the multi-user setting of both public and secret key schemes. We will concentrate specifically on the case of computing multiple discrete logarithms in a cyclic group $S = \langle P \rangle$ of order N. The ECDLP in the multi-user setting can be defined as follows. Let m be the number of users. Given $Q_i \in S$, for $i \in \{1, \ldots, m\}$, compute each x_i, such that $Q_i = x_i P$. A straightforward method to solve this problem is to compute individually each of the discrete logs. We consider that Q_i is the public key and x_i is the secret key of user i. The application behind this problem is to find the secret keys of multiple users that use the same public parameters. Naturally, the use of shared public parameters raises the question of whether we can exploit the multi-user setting to compute all discrete logs more efficiently than with the straightforward method.

This problem has been considered by Kuhn and Struik [KS01]. They reason that, all points that were collected to solve the first k discrete logs, can be used to solve the $(k+1)^{\text{st}}$ discrete log. We describe the technique in the elliptic curve setting. Thus, we talk about computing the ECDL corresponding to a secret key of a user, or simply, the discrete log of a user.

When we are computing x, such that $Q = xP$, the random map $f : S \to S$ is defined in Equation (6.1). In the multi-user setting, since Q_i is different for each user, a different random map f_i is used for each user. Hence, the collected distinguished points for a user i are of the form $R = a_i P + b_i Q_i$. When the random walks of users i and j collide, we have the following equality

$$a_i P + b_i Q_i = a_j P + b_j Q_j. \tag{6.3}$$

Knowing that $Q_i = x_i P$ and $Q_j = x_j P$, we can substitute Q_i and Q_j and obtain

$$a_i P + b_i x_i P = a_j P + b_j x_j P. \tag{6.4}$$

If x_i is already known, we can recover

$$x_j = \frac{a_i + b_i x_i - a_j}{b_j},$$

provided that b_j is not a multiple of N.

The algorithm starts with all threads computing points using the random map of the first user f_1. When a collision occurs, an equality of the form (6.2) is obtained and x_1 is computed. Then, all threads start computing points for the second user, using the map f_2. From this point on, a discrete log can be computed either by an equality of the form (6.2) or by an equality of the form (6.3). The first equality is obtained in the case where two random walks of the same user collide, whereas the second is obtained when there is a collision on random walks of two different users. Each time a new discrete log is computed, the random map is updated for all threads.

With this method, the number of collisions that are necessary to compute the discrete log of all users is equal to the number of users. The time complexity analysis in [KS01] is based on the problem of finding the number of points we need to choose with replacement out of a set of N points until we have chosen m points at least twice. The expected number of collected points, obtained with this analysis is $\sqrt{2mN}$ for $m \ll \sqrt[4]{N}$. This coincides with our Formula in Equation (6.11) for the expected number of computed distinguished points for finding m collisions, which is obtained using a different analysis and does not have a restriction on m. Kuhn and Struik also consider the problem of solving any one of the discrete log instances. However, they conclude that this problem is computationally as hard as solving one discrete log outside of the multi-user setting.

Another prominent result in the multi-user setting is the work of Fouque *et al.* [FJM14a]. The paper shows both public and secret-key cryptanalytic applications, but the one that we are interested in is the algorithm for recovering the discrete logs of m users. What follows is a description of the proposed algorithmic ideas. First, a distinction is made between the set of distinguished points obtained using the random map of a user whose discrete log is known and one whose discrete log is still unknown, called respectively the public and the private set of distinguished points. Contrary to the method in [KS01], in this algorithm, collisions between points in the private set are also considered, as they represent a correlation between the two discrete logs. More precisely, they are used to construct a graph whose vertices are the public keys and where an edge between Q_i and Q_j is added if there is a collision between the random walks of users i and j. The edge is labeled with the linear relation between x_i and x_j i.e. an equality of the form (6.4). Furthermore, there are random walks computed for users whose discrete log is known. These are not existing users, but rather public keys that we have randomly computed. We will refer to them as infiltrators. Points computed for these users are in the public set of distinguished points. When a giant connected component appears in the random graph, and if one of the infiltrators is in this component, the discrete log of all users in the component is solved. This giant component appears with high probability when the number of edges, and thus the number of computed collisions, is $cm/2$, for a small constant c. For reference, when $c = 4$, there is a giant component whose size is almost exactly $0.98m$. This estimation is obtained using results on random graph processes that can be found in [Bol01]. Then, the birthday paradox is used to estimate the number of computed points needed to find

$cm/2$ collisions. The total time complexity of the algorithm is $\widetilde{\mathcal{O}}(\sqrt{Nm})$[1].

6.1.3 Many collision applications : meet-in-the-middle attacks

A second type of applications concerns meet-in-the-middle attacks, which require finding a collision of the type $f_1(a) = f_2(b)$, where $f_1 : D_1 \to R$ and $f_2 : D_2 \to R$ are two functions with the same co-domain. As explained in [vW99], solving this equation may be formulated as a collision search problem on a single function $f : S \times \{1, 2\} \to S \times \{1, 2\}$, where the solution we need is of the type:

$$f(a, 1) = f(b, 2), \tag{6.5}$$

and S is a set bijective to D_1. This collision is called *the golden collision*. The number of unordered pairs in S are approximately $\frac{N^2}{2}$ and the probability that the two points in a pair map to the same value of f is $\frac{1}{N}$. There are $\frac{N}{2}$ expected collisions for f and there may be several solutions to Equation (6.5). Hence one typically assumes that all collisions are equally likely to occur and that in the worst case, all possible $\frac{N}{2}$ collisions for f are generated before finding the golden one. Because so many collisions are generated, memory complexity can be the bottleneck in meet-in-the-middle attacks and the memory constraint becomes an important factor in determining the running time of the algorithm. We further explain this idea in Section 6.2.

6.1.4 Computational model and data structure.

We consider a CPU implementation of the shared memory variant of the algorithm, where each thread involved in the process performs the same task of finding and storing distinguished points. In this case, the choice of a data structure for allowing efficient lookup and insertion is significant. The most common structure used in the literature is a hash table. In order to make parallel access to memory possible, van Oorschot and Wiener [vOW99] propose the use of the most significant bits of distinguished points. Their idea is to divide the memory into segments, each corresponding to a pattern on the first few bits. Threads read off these first bits and are directed towards the right segment. Each segment is organized as a memory structure on its own.

In recent years, with the development of GPUs and programmable circuits, the client-server model has been widely used for implementing parallel collision search. In this setting, a large number of client chips are communicating with a central memory server over the Internet. For computing discrete logarithms, [BBB+09] gives a comparison between implementations on different architectures in this model. Current record-breaking implementations of ECDLP also rely on this model [BKK+12; JWB09; BEL+].

Except for the need for a structure that allows efficient simultaneous access to memory, all results in this chapter apply to both the client-server and the SMP versions of the PCS algorithm, even though our experimental results are obtained using a CPU implementation following the SMP paradigm.

Notation. In the remainder of this chapter, we denote by θ the proportion of distinguished points in a set S. We denote by N the number of elements of S. We denote by E an elliptic curve defined over a prime finite field $\mathbb{F}_p$ and by $E(\mathbb{F}_p)$ the group of points on E defined over $\mathbb{F}_p$. Whenever the set S is the group $E(\mathbb{F}_p)$, N is the cardinality of this

[1]The $\widetilde{\mathcal{O}}$ notation ignores logarithmic factors.

group. For simplicity, in this case, we assume that N is prime (which is the optimal case in implementations).

6.2 Time complexity

Van Oorschot and Wiener [vOW99] gave formulas for the expected running time of parallel collision search algorithms. In this section, we revisit the steps of their proof and show a careful analysis of the running time both for computing a single collision or multiple collision applications. Our refined formulas indicate that the actual running time of the algorithm depends on the proportion of distinguished points and allow us to determine the optimal choice of θ for actual implementations.

6.2.1 Finding one collision

Van Oorschot and Wiener [vOW99] proved that the runtime for finding one collision is

$$O\left(\frac{1}{L}\sqrt{\frac{\pi N}{2}}\right),$$

with L the number of threads we use. This is obtained by finding the expected number of computed points before a collision occurs and then intuitively dividing the clock time by L when L processors are involved. The proof of the following theorem provides a more rigorous argument for the linear scalability of the algorithm. The proof techniques are similar to that of [vOW99].

Theorem 6.2.1 *Let S be a set with N elements and $f : S \to S$ be a random map. In the parallel collision search algorithm, denote by θ is the proportion of distinguished points and t_c and t_s denote the time for computing and storing a point respectively. The expected running time to find one collision for f is approximated as*

$$T(\theta) = (\frac{1}{L}\sqrt{\frac{\pi N}{2}} + \frac{1}{\theta})t_c + (\frac{\theta}{L}\sqrt{\frac{\pi N}{2}})t_s, \tag{6.6}$$

Proof. We call short path the chain of points computed by a thread between two consecutive distinguished points. The expected number of distinguished points produced after a certain clock time T is $TL\theta$. The probability of not having a collision at $T = 1$, for one thread is

$$1 - \frac{L\theta}{N\theta}.$$

Note that any of the L threads can cause a collision. Thus, the probability for all threads of not finding a collision on any point on the short walk is:

$$(1 - \frac{L}{N})^L,$$

at the moment $T = 1$.

Let X be the number of points calculated per thread before duplication. Hence:

$$P(X > T) = (1 - \frac{L\theta}{N\theta})^L \cdot (1 - \frac{2L\theta}{N\theta})^L \cdot \ldots \cdot (1 - \frac{TL\theta}{N\theta})^L.$$

To do this multiplication we are going to take a shortcut. When x is close to 0, a coarse first-order Taylor approximation for e^x as:

$$e^x \approx 1 + x.$$

Now we can rewrite our expression as:

$$\begin{aligned} P(X > T) &= (e^{-\frac{L}{N}} \cdot e^{-\frac{2L}{N}} \cdot \ldots \cdot e^{-\frac{TL}{N}})^L = (e^{\frac{-(L+2L+\ldots+TL)}{N}})^L = \\ &= (e^{\frac{-T(T+1)L}{2N}})^L = (e^{\frac{-T^2L}{2N}})^L = e^{\frac{-T^2L^2}{2N}}. \end{aligned} \tag{6.7}$$

This gives us the probability

$$P(X > T) = e^{\frac{-T^2L^2}{2N}},$$

thus the expected number of distinguished points found before duplication, is

$$E(X) = \sum_{T=1}^{\infty} T \cdot P(X = T) = \sum_{T=1}^{\infty} T \cdot (P(X > T-1) - P(X > T)) = \sum_{T=0}^{\infty} P(X > T).$$

We approximate

$$E(X) = \sum_{T=0}^{\infty} e^{\frac{-T^2L^2}{2N}} \approx \int_0^{\infty} e^{\frac{-x^2L^2}{2N}} dx \approx \frac{1}{L}\sqrt{\frac{\pi N}{2}}.$$

Since the expected length of a short walk is $\frac{1}{\theta}$, the number of distinguished points before a collision occurs is

$$\frac{\theta}{L}\sqrt{\frac{\pi N}{2}}.$$

However, a collision might occur on any point on the walk and it will not be detected until the walk reaches a distinguished one. We add $\frac{1}{\theta}$ to the number of calculations for the discovery of a collision. Finally, the expected number of calculated points per thread is:

$$\frac{1}{L}\sqrt{\frac{\pi N}{2}} + \frac{1}{\theta}.$$

The two main operations in our algorithm are computing the next point on the random walk and storing a distinguished point. Thus, the time complexity of our algorithm is:

$$T(\theta) = (\frac{1}{L}\sqrt{\frac{\pi N}{2}} + \frac{1}{\theta})t_c + (\frac{\theta}{L}\sqrt{\frac{\pi N}{2}})t_s. \tag{6.8}$$

□

Remark 6.2.1 Note that the analysis above shows that the number of points computed by the algorithm is $O\left(\theta\sqrt{\frac{\pi N}{2}}\right)$. This was proven by van Oorschot and Wiener in the first place.

Remark 6.2.2 In the client-server model, clients do not have access to memory, but they send distinguished points to the server and thus t_s stands for cost of communication on the client-side. We suppose that all the client processors are dedicated to computing points. On the server-side however, the analysis is different. Theorem 6.2.1 and the means of finding the optimal value for θ apply both to the shared memory implementation adopted in our approach, and to the more common distributed client-server model.

As we can see in Equation (6.6), the proportion of distinguished points we choose will influence our time complexity. The optimal value for θ is the one that gives the minimal run complexity. Most importantly, our analysis puts forward the idea that the optimal choice for θ depends essentially on the choices made for the implementation and memory management. From this formula, we easily deduce that if the proportion of distinguished points is too small or too large, the running time of the algorithm increases significantly.

By estimating the ratio t_s/t_c for a given implementation, one can extrapolate the optimal value of θ by computing the zeros of the derivative of the function in Equation (6.6):

$$T'(\theta) = \frac{1}{L}\sqrt{\frac{\pi N}{2}}t_s - \frac{1}{\theta^2}t_c.$$

Figure 6.3 gives timings for our implementation of the attack, using a hash table to store distinguished points. Timings shown in the figure are averaged over 100 runs on a 65-bit curve and support our theoretical findings.

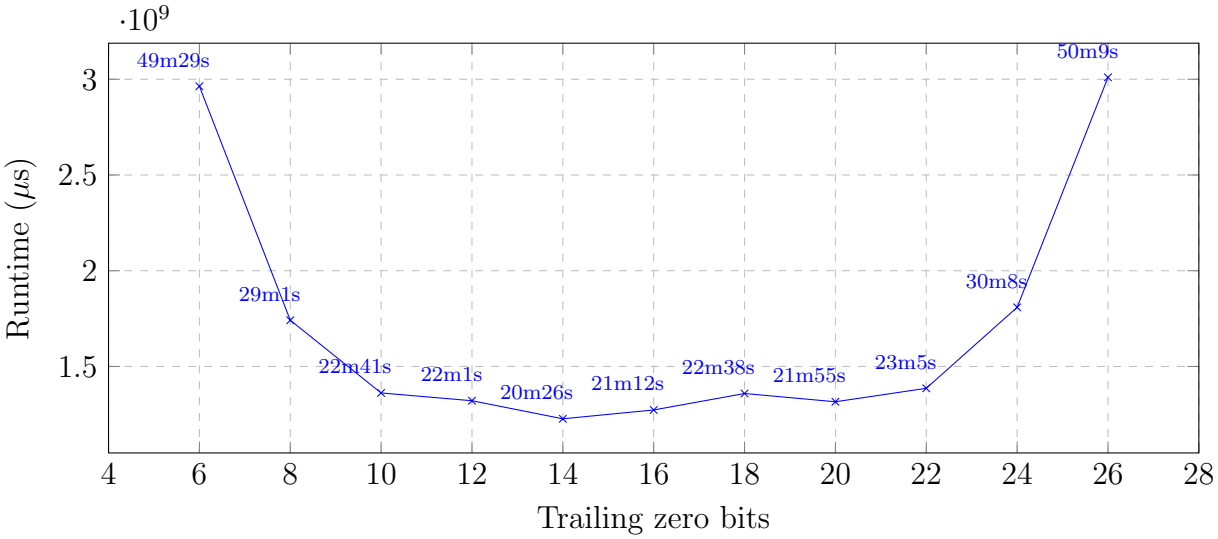


Figure 6.3: Timings of solving ECDLP for different values of θ, 65-bits curve, 28 threads

Note that most recent implementations available in the literature choose the number of trailing bits giving the distinguished point property in a range between $0.178 \log N$ and $0.256 \log N$ (see [BEL+; BLS11; JWB09]). This value was determined by experimenting on curves defined over small size fields. Our theoretical findings confirm that these values were close to optimal, but we suggest that for future record-breaking implementations, the value of θ should be determined as explained above.

6.2.2 Finding many collisions

Using a simplified complexity analysis, van Oorschot and Wiener [vW99] put forward the following heuristic.

Heuristic ([vOW99]). Let $f : S \to S$ a random map and assume that the memory can hold w distinguished points. Then in the meet-in-the-middle attack the (conjectured) optimum proportion of distinguished points is $\theta \sim 2.25\sqrt{\frac{w}{N}}$. Under this assumption, the expected number of iterations required to complete the attack using these parameters is $2.5\frac{N}{L}\sqrt{\frac{N}{w}}$.

This heuristic suggests that in the case of many collisions attacks, a memory data structure allowing to store more distinguished points will yield a better time complexity. To prove the conjectured runtime, we first give a more refined analysis for the running time of a parallel collision search for finding m collisions.

Theorem 6.2.2 *Let S be a set with N elements and $f : S \to S$ a random map. We denote by θ the proportion of distinguished points in S. The expected running time to find m collisions for f with a memory constraint of w words is:*

$$\frac{1}{L}\left(\frac{w}{\theta} + (m - \frac{w^2}{2\theta^2 N})\frac{\theta N}{w} + \frac{2m}{\theta}\right). \tag{6.9}$$

Proof. Let X be the expected number of distinguished points calculated per thread before duplication. Let T_1 be the expected number of distinguished points computed until the first collision was found, and T_i, for any $i > 1$, the expected number of points stored in the memory after the $(i-1)$th collision was found and before the ith collision is found.

As shown in Theorem 6.2.1, the expected number of points stored before finding the first collision is $T_1 = \theta\sqrt{\frac{\pi N}{2}}$. The probability of not having found the second collision after each thread has found and stored T distinguished points is

$$P(X > T) = (1 - \frac{L + T_1}{N\theta})^{\frac{L}{\theta}} \cdot (1 - \frac{2L + T_1}{N\theta})^{\frac{L}{\theta}} \cdot \ldots \cdot (1 - \frac{TL + T_1}{N\theta})^{\frac{L}{\theta}}.$$

As in the proof of Theorem 6.2.1, we approximate this expression by

$$P(X > T) = e^{\frac{-T^2L^2 - 2LT_1T}{2N\theta^2}}.$$

Hence the expected number of distinguished points computed by one thread before the second collision is:

$$\begin{aligned} E(X) &= \sum_{T=0}^{\infty} e^{\frac{-T^2L^2 - 2LT_1T}{2N\theta^2}} \approx \int_0^{\infty} e^{\frac{-x^2L^2 - 2xLT_1}{2N\theta^2}} dx = \\ &= e^{\frac{T_1^2}{2N\theta^2}} \int_0^{\infty} e^{\frac{-(xL+T_1)^2}{2N\theta^2}} dx = \frac{\theta\sqrt{2N}}{L} e^{\frac{T_1^2}{2N\theta^2}} \int_{\frac{T_1}{\theta\sqrt{2N}}}^{\infty} e^{-t^2} dt \\ &= \frac{\theta\sqrt{2N}}{L} e^{\frac{T_1^2}{2N\theta^2}} \left(\frac{\theta\sqrt{2N}e^{-\frac{T_1^2}{2\theta^2 N}}}{2T_1} - \int_{\frac{T_1}{\theta\sqrt{N}}}^{\infty} \frac{e^{-t^2}}{2t^2}\right), \end{aligned}$$

where the last equality is obtained by integration by parts. We denote by

$$U_k = T_1 + T_2 + \ldots T_k.$$

By applying repeatedly the formula above (and neglecting the last integral), we have that $T_k = \frac{\theta^2 N}{LU_{k-1}}$. Therefore we have $U_k = U_{k-1} + \frac{\theta^2 N}{LU_{k-1}}$. By letting $V_k = \frac{LU_k}{\theta\sqrt{N}}$, we obtain a sequence given by the recurrence formula

$$V_k = V_{k-1} + \frac{1}{V_{k-1}}.$$

We will use the Cesaro-Stolz criterion to prove the convergence of this limit. First, we note that this sequence is increasing and tends to ∞. Moreover we have that $V_k^2 =$

$V_{k-1}^2 + 2 + \frac{1}{V_{k-1}^2}$. Hence $\frac{V_k^2 - V_{k-1}^2}{k+1-k} \to 2$ and as per Cesaro-Stolz we have $V_k \sim \sqrt{2k}$. We conclude that

$$U_k \sim \frac{\theta\sqrt{2kN}}{L}. \tag{6.10}$$

Since U_k is the number of distinguished points computed per thread, the total number of stored points is $\theta\sqrt{2kN}$. Hence the memory will fill when $\theta\sqrt{2kN} = w$. This will occur after computing the first $k_w = \frac{w^2}{2\theta^2 N}$ collisions and the expected total time for one thread is $\frac{w}{L\theta}$. When the memory is full, the time to find a collision is $\frac{\theta N}{w}$ (see [vW99] for detailed explanation). Finally, to actually locate the collision, we need to restart the two colliding trails from their start, which requires $2/\theta$ steps on average.

To sum up, the total time to find m collisions is:

$$\frac{1}{L}\left(\frac{w}{\theta} + (m - \frac{w^2}{\theta^2 2N})\frac{\theta N}{w} + \frac{2m}{\theta}\right).$$

□

Remark 6.2.3 According to the formula obtained in Equation 6.10, we see that if the memory is not filled when running the algorithm for finding $\frac{N}{2}$ collisions, as in meet-in-the-middle applications, then we store θN distinguished points, i.e. all distinguished points in S.

Note that the proof of Theorem 6.2.2 relies strongly on our formula for the expected total number of computed distinguished points for finding m collisions, when m is sufficiently large and the memory is not limited:

$$S_m \approx \theta\sqrt{2mN}. \tag{6.11}$$

We confirmed this asymptotic formula experimentally by running a multi-collision algorithm for a curve over a 55-bit prime field. The comparison of our formula with the experimental results is in Table 6.1. Each value in this Table is an average of 100 runs where we set $\theta = 1/2^{13}$.

Collisions	Experimental Avg.	S_k	Collisions	Experimental Avg.	S_k
100	238289	231704	500	530493	518107
1000	750572	732714	2000	1062581	1036215
5000	1681831	1638399	7000	1990671	1938581

Table 6.1: Comparing Formula 6.11 to an experimental average.

Finally, recall that in the meet-in-the-middle attack, one needs to compute $\frac{N}{2}$ collisions. By minimizing the complexity function obtained in Theorem 6.2.2 , we obtain an estimate for the optimal value of θ to take, in order to minimize the running time of the algorithm.

Corollary 6.2.1 *The optimum proportion of distinguished points minimizing the time complexity bound in Theorem 6.2.2 is* $\theta = \frac{\sqrt{w^2+2Nw}}{N}$. *Furthermore, by choosing this value for* θ, *the running time of the parallel collision search algorithm for finding* $\frac{N}{2}$ *collisions is bounded by:*

$$O\left(\frac{N}{L}\sqrt{1 + \frac{2N}{w}}\right). \tag{6.12}$$

Proof. From Theorem 6.2.2, the runtime complexity is given by:

$$T(\theta) = \frac{1}{L}\left(\frac{w}{\theta} + (\frac{N}{2} - \frac{w^2}{\theta^2 2N})\frac{\theta N}{w} + \frac{N}{\theta}\right).$$

By computing the zeros of the derivative:

$$T'(\theta) = \frac{N^2\theta^2 - w^2 - 2Nw}{2Lw\theta^2},$$

we obtain that by taking $\theta = \frac{\sqrt{w^2+2Nw}}{N}$, the time complexity is $O\left(\frac{N}{L}\sqrt{1+\frac{2N}{w}}\right)$. □

This confirms and proves the heuristic findings in [vOW99]. Most importantly, Corollary 6.2.1 suggests that in the case of applications which fill the memory available, the number of distinguished points we can store is an important factor in the running time complexity. More storage space yields a faster algorithm by a constant factor. We propose such a optimization in Section 6.4.

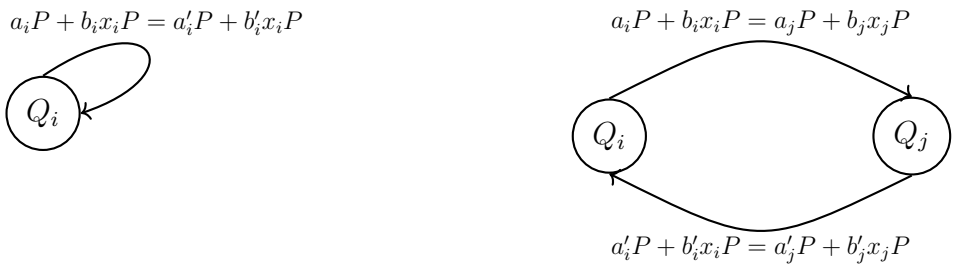
6.3 The multi-user setting

To develop our approach for computing elliptic curve discrete logs in the multi-user setting, we start from the idea in [FJM14a] of using a graph to represent the correlations between discrete logs. Vertices in the graph represent users and are labeled v_i, for $i \in \{1, \ldots, m\}$. As in [FJM14a], an edge represents a collision. However, for demonstrating purposes, the graph that we construct is a directed graph. An edge (v_i, v_j) is considered to be directed from v_i to v_j, if the user i computes a distinguished point that collides with a distinguished point that has previously been stored by user j. In other words, user i is the one that found the collision. In this section, we argue that

- there is no need for infiltrators in the graph;
- exactly one collision per user is required to compute the discrete log of all users;

Both these conditions are met due to our use of cycles in the graph. Recall that, a cycle of a graph is a subset of edges that forms a trail such that the first vertex of the trail is the same as the last vertex of the trail. The length of a cycle is the number of edges in this subset. It can be shown that the occurrence of a cycle in the graph allow us to recover the discrete logs of all users in the cycle. A cycle of length 1 is obtained when there is a collision between two distinguished points found by the same user. In this case, we have an equality of the form (6.2) and the discrete log of the user is computed as in the single-user setting. A length 2 cycle is obtained when we have two distinct collisions between two users, say user i and user j. In this case, we have two linear equations of the form (6.4) with two unknowns, x_i and x_j. These two cases are illustrated in Figure 6.4. It stands in the general case that the labels of all edges in a cycle form a linear system with as many equations as unknowns. Hence, even if none of the discrete logs in a cycle is known they can all be computed by solving the linear system.

Our goal is to construct a graph that allows us to recover all discrete logs. We have demonstrated that, when a cycle is obtained, all discrete logs of users involved in the cycle are recovered. By extension, the discrete log of each user that collides with any of the users forming the cycle is recovered as well. This reasoning leads us to the conclusion that a graph G that allows us to recover all discrete logs needs to have the following property.



(a) Cycle of length 1.

(b) Cycle of length 2.

Figure 6.4: Examples of cycles and their corresponding linear systems.

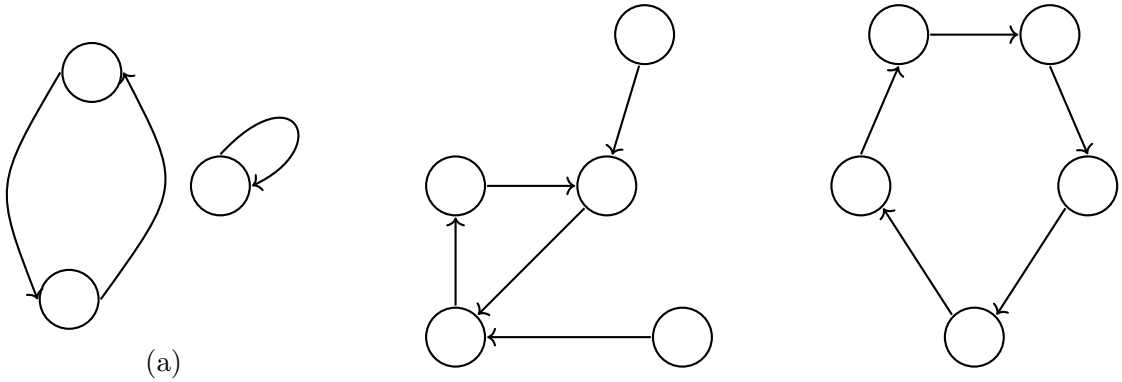


Figure 6.5: Examples of graphs where all connected components contain a cycle.

All connected components in G contain a cycle. Figure 6.5 shows some example topologies of graphs with this property.

Our second claim is that, to compute the discrete log of all users, we need exactly one collision per user. This is not to be confused with "a number of collisions equal to the number of users". The difference is that, with the latter wording, we allow the case where one user finds two collisions and another user finds none. Recall that, in our graph representation, a collision found by a user is represented as a directed edge from the user to another user or to himself. Thus, in order to prove that when each user finds exactly one collision, all discrete logs are recovered, we need to prove the following theorem.

Theorem 6.3.1 *Let $G(V, D)$ be a directed graph. If for each $v_i \in V$, there exists at least one $v_j \in V$, such that $(v_i, v_j) \in D$, then all connected components in G contain a cycle.*

Proof (by contradiction). Suppose that for each $v_i \in V$, there exists at least one $v_j \in V$, such that $(v_i, v_j) \in D$, and suppose that there exists a connected component $G_1(V_1, D_1)$ in G that does not contain a cycle. Let us denote by o the number of vertices in G. The component G_1 contains at least one vertex, say v_1. According to the first supposition, there exists at least one directed edge from v_1 to another vertex. Note that, we can not have $(v_1, v_1) \in D_1$, as this constitutes a cycle. Hence, there are $(o-1)$ choices for the vertex at the end of this edge. Let us denote the choice by v_2, and derive that $(v_1, v_2) \in D_1$. Again, according to the first supposition, there exists at least one directed edge from v_2 to another vertex. As before, this edge can not be (v_2, v_2), as it forms a cycle. However, the edge (v_2, v_1) forms a cycle as well, since we already have $(v_1, v_2) \in D_1$. Thus, we are left with $(o-2)$ choices for this vertex. If we continue with this reasoning, we see that the only way to not contradict the second supposition is to keep adding vertices to G_1, such that for each new v_i, we have $(v_{i-1}, v_i) \in D_1$. When we add the final vertex v_o to G_1, we

already have $(v_{i-1}, v_i) \in D_1$, for $i \in \{1, \ldots, o\}$, and thus a directed edge from v_o to any other vertex forms a cycle in G_1. We conclude that the second supposition is false. □

To sum up, we propose the following method for computing all discrete logs in a multi-user setting.

1. Define a graph where each vertex corresponds to a user.
2. For each user i, compute distinguished points using a random map f_i, until a collision is found. Store the collision as an edge in the graph[2].
3. Use the graph to compute all discrete logs.

The third step requires several depth-first transversals to discover the cycles of all connected components and then, some linear algebra to compute the discrete logs. However, for cryptographic field sizes, the overall cost of this algorithm is dominated by the collision search phase. As per Formula (6.11), the expected number of computations to find m collisions is $\sqrt{2mN}$. When the memory is limited to w stored points, Theorem 6.2.2 applies.

The complexity of our method corresponds to the one in [KS01]. At the implementation level, there is a similar problem that arises in both version. In [KS01], when a collision is found, all threads need to update the random map that they use for computing points and thus the information that a collision has been found needs to be broadcasted to all threads, which adds additional communication cost. This setback can be eliminated by the graph construction method in [FJM14a]. In our version, when a collision is found, only the thread that found the collision needs to update the currently used random map. However, this update still requires some management that can lead to additional communication cost. For instance, all threads need to have a global vision of which users already have a collision, in order to choose the user that should be treated next. Do we have a central authority that keeps track of which users don't have a collision yet? Or do we attribute a list of users to each thread at the beginning, so that when a collision is found for one user, we start looking for the next user in the list? The former solution results in too much communication and with the latter solution, we can have threads that find all collisions in their list faster than other threads, in which case they would stop computing. As a result, we end up with unused computational resources. We think that this practical problem finds a practical solution in multithreading. For m users, we allocate m threads even if we have L processing units with $L < m$. In this case, we leave the time management task to the CPU. Each thread is dedicated to one user since the start and there is no need to update the random map. When some threads finish their job, more time will be afforded to the remaining threads. Note that, in cases where $L > m$ or $L \lesssim m$, this solution can lead to having unused resources and thus, another solution should be found to distribute the work in these cases.

Open question 4 Our version requires less collisions, compared to the version in [FJM14a]. However, whether our version yields faster running times in practice depends on how significant the cost of the cycle search and linear algebra is. An implementation of both approaches and experimental running-time comparisons are needed before we can give a conclusion to our theoretical findings. We set this as a question of future research.

[2]It is unnecessary to use directed edges. We use a directed graph only to simplify the explanation of this method.

6.4 Our approach for the data structure

In this section, we evaluate the memory complexity of parallel collision search algorithms. As explained in Section 6.1, van Oorschot and Wiener's [vOW99] proposed to divide the memory into segments to allow simultaneous access by threads. We revisit this construction, with the goal in mind to minimize the memory consumption as well. Since in Section 6.2 we showed that the time complexity of collision search depends strongly on the available amount of memory, we propose an alternative structure called a Packed Radix-Tree-List, which will be referred to as PRTL . We explain how to choose the densest implementation of this structure for collision search data storing in Section 6.5.

Since the PRTL is inspired by radix trees, we first describe the classic radix tree structure and then we give complexity analysis on why its straightforward implementation is not memory efficient. The PRTL structure has the memory gain of radix tree common prefixes, but avoids the memory loss of manipulating pointers.

6.4.1 Radix tree structure

Each distinguished point from the collision search is represented as a number in a base of our choice, denoted by b. For example, in the case of attacks on the discrete logs on the elliptic curve, we may represent a point by its x-coordinate. The first numerical digit of this number in base b gives the root node in the tree, the next digit is a child and so on. This leads to the construction of an acyclic graph which consists of b connected components (i.e. a forest).

In regard to memory consumption, we take advantage of common prefixes to have a more compact structure. Let c be the length of numbers written in base b that we store in the tree and K the number of distinguished points computed by our algorithm. To estimate the memory complexity of this approach, we give upper and lower bounds for the number of nodes that will be allocated in the radix tree before a collision is found.

Proposition 6.4.1 *The expected number of nodes in the radix tree verifies the following inequalities:*

$$\frac{b}{b-1}K - c - \log_b K - 1 \leq Z(K) \leq (c - \log_b K + \frac{b}{b-1})K. \tag{6.13}$$

Proof. The lower and upper bound in Equation (6.13) are given by the worst-case and best-case scenario for the number of nodes.

Worst-case scenario. In the worst case scenario, for each new word added in this structure we will create as much nodes as possible. This means that the x-coordinates of the added points have the shortest possible common prefix, as shown in Figure 6.6. For the first b points, we will use bc nodes. After that, the first distinguished point that we find will take $c-1$ nodes, since all possibilities for the first letter in the string were created. This case is repeated $(b-1)b$ times, provided that $K > b + (b-1)b$.

More generally, let $k = \lfloor \log_b K \rfloor - 1$. We build the tree by allocating nodes as follows:

- bc nodes for the first b points
- $(b-1)b(c-1)$ for the next $(b-1)b$ points
- $(b-1)b^2(c-2)$ for the next $(b-1)b^2$ points etc.
- $(b-1)b^k(c-k)$ for $(b-1)b^k$ points.

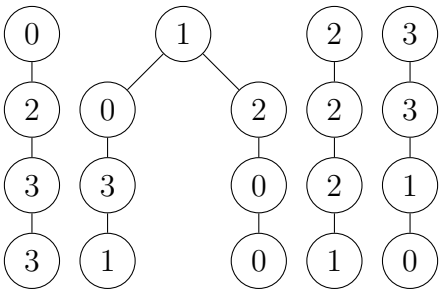


Figure 6.6: Example of one of the worst-case scenarios with parameters $K = 5$ and $b = 4$

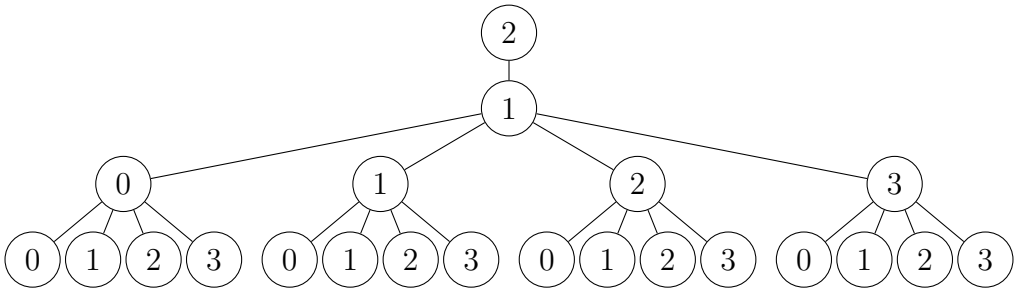


Figure 6.7: Example of one of the best-case scenarios with parameters $K = 16$ and $b = 4$

For each of the remaining $K - (b + \sum_{i=1}^{k}(b-1)b^i)$ points we will need $c-k-1$ nodes. To sum up, the total number of nodes that will bound our worst-case scenario is given by:

$$Z(K) = bc + \sum_{i=1}^{k}(b-1)b^i(c-i) + (K - b - b(b-1)\sum_{i=0}^{k-1} b^i)(c-k-1).$$

We simplify the sums and we approximate by:

$$Z(K) \approx \frac{b}{b-1}b^{k+1} + K(c-k-1).$$

Since $k = \lfloor \log_{10} K \rfloor - 1$, we have that

$$Z(K) \approx \frac{b}{b-1}b^{\lfloor \log_b K \rfloor} + K(c - \lfloor \log_b K \rfloor). \tag{6.14}$$

Best-case scenario. Let K be the number of distinguished points that we need to store and let $k = \lfloor \log_b K \rfloor$. In the best-case scenario, we may assume without loss of generality that each time a new point is added in the structure, the minimal number of nodes is used, i.e. the x-coordinate of the added point has the longest possible common prefix with some other point that was previously stored. For example, for the first point c nodes are allocated, for the next $(b-1)$ nodes, one extra node is allocated and so on, until all subtrees of depth 1, 2 etc. are filled one by one. Figure 6.7 gives an example of how 215 points are stored. If $K > b^{c-1}$, we fill the first tree and start a new one. Let x_i, for $i \in \{0, 1 \ldots, k\}$, denote the i-th digit of K, from right to the left. In full generality, since $c > k$, we use:

- x_k complete subtrees of depth k and a $(x_k{+}1)$-th incomplete tree of depth k;
- the $(x_k{+}1)$-th tree of depth k has x_{k-1} complete subtrees of depth $k-1$ and a $(x_{k-1}{+}1)$-th incomplete tree of depth $k-1$;
- $c-k-1$ extra nodes.

Summing up all nodes, we get the following formula:

$$\begin{aligned} Z(K) &= \sum_{i=0}^{k} x_i \sum_{j=0}^{i} b^j + k + c - k - 1 = \frac{1}{b-1} \sum_{i=0}^{k} x_i (b^{i+1} - 1) + c = \\ &= \frac{b+1}{b} K - c - \frac{1}{b-1} \sum_{i=0}^{k} x_i. \end{aligned}$$

We conclude that:

$$Z(K) \geq \frac{b}{b-1} K - c - k - 1. \tag{6.15}$$

□

Traditionally, nodes in a tree are implemented as arrays of pointers to child nodes. This representation will lead to excessive memory consumption when the data to be stored follows a uniform random distribution, leading to sparsely populated branches and to the average distribution of nodes in the tree being closer to the worst case than to the best case.

The difference between the worst-case value and the best-case value can be approximated as $\Delta \sim K(c - \log_b K)$. Depending on the application, this value may be large. Let us consider the case where a single collision is required for solving the ECDLP. By a theorem of Hasse [Sil86], we know that the number of points on the curve is given by $N = p + 1 - t$, with $|t| \leq 2\sqrt{p}$. Since we assume that N is prime, we approximate $\log N \sim \log p$. Hence an approximation of Δ is:

$$\Delta \sim \theta \sqrt{\frac{\pi N}{2}} (\frac{1}{2} \log_b N - \log_b \sqrt{\frac{\pi}{2}}),$$

which implies that the tree is sparse. In the case of many collisions algorithms, $c \sim \log_b K$ and this standard deviation becomes negligible, resulting into a space-reduced data structure. We show how to handle sparse trees efficiently in Section 6.4.2.

6.4.2 Packed Radix-Tree-List

Starting from the analysis in Section 6.4.1, we look to construct a more efficient memory structure by avoiding the properties of the classic radix tree which make it memory costly for our purposes. Intuitively, we see that the radix tree is dense at the upper levels and sparse at the lower ones. Hence it would be more efficient to construct a radix tree up to certain level and then add the points to linked lists, each list starting from a leaf on the tree. We denote by l be the level up to which we build the radix tree. We call this a Packed Radix-Tree-List[3]. Figure 6.8 illustrates an example of an abstract Radix-Tree-List in base 4.

This idea was considered by Knuth [Knu98, Chapter 6.3] for improving on a table structure called trie, introduced by Fredkin [Fre60]. Knuth considers a forest of radix trees that stop branching at a certain level, whose choice is a trade-off between space and fast access. Indeed, the more we branch, the faster the look-up is, but the more memory we require. He suggests that the mixed strategy yields a faster look-up when we build a tree up to a level where only few keys are possible. Starting from this level a sequential search through a list of the remaining keys is fast.

[3]The 'packed' property is addressed in Section 6.5, where we give implementation details.

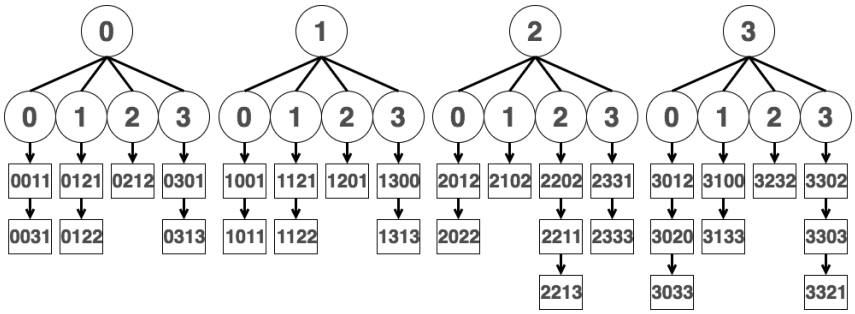


Figure 6.8: Radix-Tree-List structure with $b = 4$ and $l = 2$

K	Value of l	Average nb. of empty lists per run	
		Level l	Level $l + 1$
5 million	18	0	37
7 million	18	0	0.84
10 million	19	0	75

Table 6.2: Verifying experimentally the optimal level.

In our use case, we favor memory optimization to fast lookup, thus we use a different technique to decide on the tree level. First, we look to estimate up to which level the tree is complete for our use case. The number of leaves in a complete radix tree of depth l is b^l. As per the coupon collector's problem, all the linked lists associated with a leaf will contain at least one point when the following inequality is verified:

$$K \geq b^l(\ln b^l + 0.577). \tag{6.16}$$

We consider the highest value of l which satisfies this inequality to be the optimal level, as it allows us to obtain the shortest linked lists while having 100% rate of use of the memory structure. We verified this experimentally by inserting a given number of randomly obtained points of length 65, with $b = 2$, in the PRTL structure. The results are in Table 6.2. We performed 100 runs for each value of K and counted the number of empty lists at the end of each run. None of the 300 runs finished with an empty list in the PRTL structure, which supports the claim that the obtained l is small enough to have at least one point per list. Then, to confirm that l is the highest possible value that achieves this, we reproduced the experiments by taking $l + 1$, which is the lowest value that does not satisfy Equation (6.16). The results show that $l + 1$ is not small enough to produce a 100% rate of use of the memory, therefore l is in fact the optimal level to choose. We notice that in the case of 7 million points, we have very few (or none) empty slots even by taking the level $l + 1$. This is explained by the fact that a PRTL of level 19 needs 7207281 points to have all of its lists filled, as per 6.16. Given that 7 million is very close to this number, some of the runs finish with 100% rate of use.

The attribution of a point to a leaf is determined by its prefix and we know in advance that all the leaves will be allocated. Therefore, in practice we do not actually have to construct the whole tree, but only the leaves. Hence, we allocate an array indexed by prefixes beforehand and then we insert each point in the list for the corresponding prefix. The operation used to map a point to an index is faster than a hash table function. More precisely, we perform a bitwise AND operation between the x-coordinate of the point and a precomputed mask to extract the prefix. Furthermore, the lists are sorted. Since we are doing a search-and-add operation, sorting the lists does not take additional time and

proves to be more efficient than simply adding at the end of the list. Figure 6.9 illustrates the implementation of this structure.

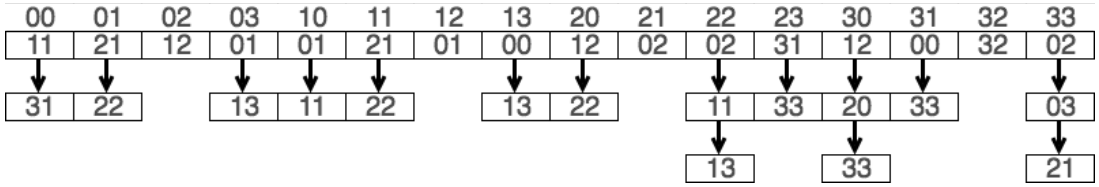


Figure 6.9: PRTL implementation. Same points stored as in Figure 6.8.

Remark 6.4.1 When implementing the attack for curves defined over sparse primes, we advise taking an l-bit suffix instead of an l-bit prefix. Prefixes of numbers in sparse prime fields are not uniformly distributed and one might end up only with prefixes starting with the 0-bit, and therefore a half empty array.

Remark 6.4.2 To experiment with this structure, we chose the example of ECDLP. In this case, we store the starting point of the Pollard walk aP and the first distinguished point we find, represented by the coefficient a and the x-coordinate correspondingly. Consequently, we store a pair $(x\text{-}coordinate, a)$. However, the analysis and choices we made for constructing the PRTL are valid for every collision search application which needs to store pairs $(key, data)$ and requires pairs to be efficiently looked up by keys. For the ECDLP, Bailey et al. [BBB+09] propose, for example, to store a 64-bit seed on the server instead of the initial point, which makes the pair $(x\text{-}coordinate, seed)$.

6.5 Implementation and benchmarks

To support our findings, we implemented the PCS using both PRTLs and hash tables for discrete logarithms on elliptic curves defined over prime fields. Our C implementation relies on the GNU Multiple Precision Arithmetic Library [MM11] for large numbers arithmetic, and on the OpenMP (Open Multi-Processing) interface [OPE] for shared memory multiprocessing programming. Our implementation is publicly available [TID19]. Our experiments were performed on a 28-core Intel Xeon E5-2640 processor using 128 GB of RAM and we experimented using between 1 and 28 threads. In this section, first we explain in detail the implementation of the PRTL structure and the hash table. Then, we show experimental results using both structures. For completeness,we enumerate the choices we made in our implementation and which are common in the literature.

6.5.1 PRTL implementation

An entry in the lists in the PRTL stores one $(key, data)$ pair. The intuitive structure implementation of such a link would be the following:

```
struct {
pointer to key-suffix;
pointer to data;
pointer to next;
} link;
```

In addition, we have the bytes storing the actual values for the *key*-suffix and the *data*.

In order to have the best *packed* structure, we look to avoid wasting space on addressing, structure memory alignment and unintended padding. Hence we propose to store all relevant data in one byte-vector. Our compact slot has the following structure:

```
struct {
byte vector[vector size];
pointer to next;
} link;
```

The *key*-suffix and *data* are bound in one single vector. In this way, we have at most 7 bits wasted due to alignment. We designed functions that allow us to extract and set values in the vector.

Remark 6.5.1 In our implementation, we always use $b = 2$, and thus, the parameter b will no longer be specified.

Example 6.5.1 Let $E : y^2 = x^3 + 25x + 3$ be a 15-bit curve over $\mathbb{F}_{16411}$, with $N = 16333$ and let $\theta = \frac{1}{2^3}$, which means that distinguished points are points whose x-coordinates have 3 trailling zero bits. To solve a discrete log on this curve, we expect to store $K = 20$ points and thus, we take $l = 2$. In this case, it might be wiser to take $l = 3$, as it needs 22 points to be filled (as per (6.16)) and is closer to our estimated K than $l = 2$ that needs only 8 points to be filled. But to avoid confusion, we keep our recommendation to take the highest l that satisfies (6.16) without exception.

$R = (2328, 12535)$ is a distinguished point on E that has been obtained on a random walk starting from aP, where $a = 10337$. Thus, the pair $(key, data)$ that will be stored is $(x\text{-}coordinate, a) = (2328, 10337)$. The binary representation of the x-coordinate is 100100011000, and when we remove the 3 trailling zero bits and the *key*-prefix with $l = 2$, we get the *key*-suffix 0100011. The binary representation of a, which we concatenate to the *key*-suffix, is 10100001100001 and thus we obtain the vector 010001110100001100001 of size 3 bytes. Including the pointer to the next slot[4], the point is stored using 11 bytes.

6.5.2 PRTL vs. hash table.

We experimented with the ElfHash function, which is used in the UNIX ELF format for object files. It is a very fast hash function, and thus comparable to the mask operation in our implementation. Small differences in efficiency are negligible since the insertion is the less significant part of the algorithm. Indeed, recall that one insertion is done after $\frac{1}{\theta}$ computations.

As is the practice with the PCS, we allocate K indexes for the hash table, since we expect to have K stored points. Recall that this guarantees an average search time of $O(1)$, but it does not avoid multi-collisions. Indeed, according to [Jou09, Section 6.3.2], in order to avoid 3-multi-collisions, one should choose a hash table with $K^{\frac{3}{2}}$ buckets. Consequently, we insert points in the linked lists corresponding to their hash keys, as we did with the PRTL. Every element in the list holds a pair $(key, data)$ and a link to the next element. The PRTL is more efficient in this case as we only need to store the suffix of key.

With this approach, we can not be sure that a 100% of the hash table indexes will have at least one element. We test this by inserting a given number of random points on a 65-bit curve and counting the number of empty lists at the end of each run, like we did to test the rate of use for the PRTL. We try out two different table sizes: the recommended hash table size and for comparison, a size which matches the number of leaves in the PRTL. All results are an average of 100 runs.

[4]Considering a pointer is 8 bytes for a 64-bit architecture

Nb. of points	Average nb. of empty lists for $size = K$	Average nb. of empty lists for $size = 2^l$
5 million	2592960 (51.85%)	98308 (37.50%)
7 million	3632679 (51.89%)	98304 (37.50%)
10 million	5138792 (51.38%)	196615 (37.50%)

Table 6.3: Test the rate of memory use of a hash table structure.

Results in Table 6.3 show that when we choose a smaller table size, we have fewer empty lists, but the hash table is still not 100% full. Due to these results, when implementing a hash table we choose to allocate a table of pointers to slots, instead of allocating a table of actual slots which will not be filled. This is the optimal choice because we only waste 8 bytes for each empty slot, instead of 24 (the size of one slot).

Since results in Table 6.2 show that the array in PRTL will be filled completely, when using this structure we allocate an array of slots directly. This makes PRTL save a constant of $8K$ bytes compared to a hash table.

Calculating the exact memory occupation. Let f be the size of the field in bits and t the number of trailing bits set to zero in a distinguished point. We keep the notation of K for the expected number of stored points and of l for the level of the PRTL structure. To calculate the expected memory occupation of the entire PRTL structure, we first calculate the size of a compact slot. Recall that a compact slot holds one byte-vector and a pointer to the next slot, because we used linked lists. Thus, one compact slot takes $\lceil (f-l-t+f)/8 \rceil + 8$ bytes ($f-l-t$ bits for the *key*-suffix and f bits for the *data*). The size of the slot is multiplied by K, as K slots will be allocated. To make sure that the access to the shared memory is asynchronous, we use locks on the shared data structure. To minimize the time threads spend on locks, there is a lock for every entry in the array, which makes a total of 2^l locks. This adds $8 \cdot 2^l$ bytes to the total memory occupation. Let us compare the total cost to that of a hash table. Recall that, the entries in our classical implementation of a hash table are linked lists as well. To store a pair $(key, data)$, every element in the linked list holds a pointer to the *key*, a pointer to the *data* and a pointer to the next element, which need 24 bytes in total. To this we add $\lceil (f-t)/8 \rceil$ bytes for storing the *key* and $\lceil f/8 \rceil$ bytes for storing the *data*. All this is multiplied by K. Finally, we allocate $8K$ bytes for the array of pointers to slots and $8K$ bytes for the locks (one lock for each entry in the array). Hence, the total memory occupation of the hash table is $(40 + \lceil (f-t)/8 \rceil + \lceil f/8 \rceil)K$.

Table 6.4 shows examples of memory requirements of large ECDLP computations calculated in this way. The first two lines in this Table concern the computation in [BEL+] on the elliptic curve **target117** over $\mathbb{F}_{2^{127}}$. For this example, we suppose that a seed of 64 bits is stored instead of the k-coefficient representing the *data* (see Remark 6.4.2). On the first line we give the memory amount needed to store the expected number of distinguished points, while on the second line we consider the actual computation in [BEL+], that finished after collecting 968531433 distinguished points. Note that, in the case of the actual computation, the l parameter was calculated with respect to the estimated value of K, since l always needs to be set beforehand. Similarly, the size of the hash table and the number of locks correspond to the estimated value of K, as the table is allocated beforehand. On the third line, we also give the memory requirements for a discrete log computation on a 160-bit curve, with an estimated number of stored distinguished points.

Field	θ	K	l	PRTL	Hash table
117.35-bit [BEL+] estimation	$1/2^{30}$	~ 379821956	24	9.6 GB	23.1 GB
117.35-bit [BEL+] computation	$1/2^{30}$	968531433	24	24 GB	49.6 GB
160-bit estimation	$1/2^{40}$	$\sim 2^{40}$	35	43155 GB	82463 GB

Table 6.4: Memory requirements of large ECDLP computations using PRTLs and hash tables.

It is clear that when one implements the PRTL, this structure takes the form of a hash table where the hash function is in fact the modulo of a specific value calculated using Equation (6.16). It might seem counter-intuitive that the optimal solution for a hash function is the modulo function. However, collision search algorithms do not require a memory structure that has hash table properties, such as each key to be assigned to a unique index.

Indeed, a well distributed hash function is useful when we look to avoid multi-collisions. With collision search algorithms, the number of stored elements is so vast that we can not possibly allocate a hash table of the appropriate size and thus we are sure to have longer than usual linked lists. Fortunately, this is not a problem since the insertion time is, in this case, not significant compared to the $\frac{1}{\theta}$ random walk computations needed before each insertion. For example, $\frac{1}{\theta}$ would be of order 2^{32} for a 129-bit curve. On the other hand, as shown in Section 6.2, the available storage space is a significant factor in the time complexity, which makes the use of this alternative structure more appropriate for collision searches.

Our implementation of a PRTL yields a better memory occupation, but most importantly, manipulating this structure does not slow down the overall runtime of the attack. Indeed, as per Equation (6.16), the average number of elements in a chained list corresponding to a prefix is $\frac{K}{b^l} \approx l \log b + 0.577$. This shows that the search time in our structure is negligible. We show experimental results that verify this in Table 6.5, where we insert a given number of random points on a 65-bit curve, using both a hash table and the PRTL. To have a measurement of the runtime that does not depend on point computation time, we take $\theta = 1$, meaning every point is a distinguished one. The *key* length is thus $c = 65$. All results are an average of 100 runs.

We show similar experiments in Table 6.6. This time, we performed actual attacks on the discrete log over elliptic curves, instead of inserting random points. Since the number of stored points is now random and can be different between two sets of runs, the runtime per stored point and memory per stored point are more relevant results. The results are an average of 100 runs and they show that by using a PRTL for the storage of distinguished points we optimize the memory complexity by a factor of 3.

K	Memory		Runtime	
	PRTL	Hash table	PRTL	Hash table
5 million	106 MB	324 MB	5.05 s	5.20 s
7 million	148 MB	454 MB	6.74 s	7.01 s
10 million	213 MB	649 MB	9.84 s	10.2 s

Table 6.5: Comparing the insertion runtime and memory occupation of a PRTL vs. a hash table.

Field	Memory		Memory per point		Runtime		Runtime per point	
	PRTL	Hash table	PRTL	Hash table	PRTL	Hash table	PRTL	Hash table
55-bit	402 KB	1172 KB	19 B	59 B	35 s	36 s	1.69 ms	1.81 ms
60-bit	618 KB	1801 KB	20 B	59 B	210 s	212 s	6.88 ms	6.91 ms
65-bit	1856 KB	5212 KB	21 B	60 B	1292 s	1291 s	14.90 ms	14.95 ms

Table 6.6: Runtime and the memory cost for attacking ECDLP using PRTLs and hash tables.

6.5.3 ECDLP implementation details and scalability.

Additive walks.

Teske [Tes01] showed experimentally that the walk proposed by Pollard (6.1) originally performs on average slightly worse than a random walk. She proposes alternative mappings that lead to the same performance as expected in the random case: additive walks and mixed walks. The additive walks are presented as follows. Let r be the number of sets S_i which give a partition of the group G, and let M_i be a linear combination of P and Q: $M_i = a_iP + b_iQ$, for $i = \{1, \ldots, r\}$. We choose the iterating function of the form:

$$R_{i+1} = f(R_i) = \begin{cases} R_i + M_1, & R_i \in S_1; \\ R_i + M_2, & R_i \in S_2; \\ \ldots & \\ R_i + M_r, & R_i \in S_r. \end{cases} \tag{6.17}$$

In the case of mixed walks, we introduce squaring steps to r-additive walks. However, Teske's experimental results show that apart from the case $r = 3$, the introduction of squaring steps does not lead to a significantly better performance. After experimenting with both of them, we confirmed her conclusion and decided to use additive walks.

In [Tes01], it is shown experimentally that if $r \geq 20$ then additive walks are close to random walks. We therefore chose $r = 20$ in our implementation.

Use of automorphisms

If the function f is chosen such that $f(R) = f(-R)$ then we may regard f as being defined on equivalence classes under $\pm$. Since there are $\frac{N}{2}$ equivalence classes, this would lead to a theoretical speedup of $\sqrt{2}$. However, it was observed that the use of the negation map leads to so-called fruitless cycles, cycles that trap the random-walks. In practice, since these cycles need to be handled, the actual speed-up is significantly less than $\sqrt{2}$ and actually depends on the platform one uses [BLS11]. In our work, we aim at evaluating

the performance of our algorithm independently of the platform one may choose for its implementation. Therefore, we do not use automorphisms in our implementation.

Long walks vs. short walks

As we explained in Section 6.1 every thread selects a starting point, which is a multiple of P, and computes the random walk until a distinguished point is found. After the distinguished point is stored in the radix tree, the thread starts a new walk from a new starting point. We refer to this as a short walk because the walk stops at the first distinguished point and has an average length of $1/\theta$. A second possibility is that the thread would continue the walk from a distinguished point rather than start from a new one. We refer to this as a long walk. This approach is used in the Pollard's rho method because it allows the walk to enter a cycle and is an indispensable factor in finding a collision using the Floyd's cycle finding algorithm [Jou09]. However, in the PCS algorithm every distinguished point is stored, and thus the cycle property is irrelevant.

Furthermore, using the short walk method, we are not required to calculate the coefficients a and b every time. We calculate only the value of R for each iteration, and when we find a distinguished point we store the coefficients of the starting point (only the coefficient a is stored because the starting point being a multiple of P, the b coefficient is zero). It is only when a collision is found that we start iterating from the beginning of the short walk, this time computing a and b. This convenience makes short walks the better choice. Furthermore, we experimented with both short walks and long walks, finding that short walks give slightly better runtime results. All our results presented here use short walks.

Parallel Performance

In the theoretical model [vOW99], the PCS is considered to have linear scalability and our time complexity in Theorem 6.2.1 confirms this. To assess the parallel performance of our implementation, we experimented with $L \in \{1, 2, 7, 14, 28\}$ threads, solving the discrete log over a 60-bit curve. Table 6.7 shows the Wall clock runtime and the parallel performance of the attack when we double the number of threads. The parallel performance is an indication of how the runtime of a program changes when the number of parallel processing elements increases. It is computed as

$$\frac{L_1 t_1}{L_2 t_2},$$

where t_i is the Wall clock runtime with L_i threads and $L_1 > L_2$. A program is considered to scale linearly if the speedup is equal to the number of threads used i.e. if the parallel performance is equal to 1 (or very close to 1, in practice). From our results, we conclude that the parallel performance is not as good as expected for a small number of threads, but gets closer to linear as the number of threads grows.

6.5.4 Multi-collision search computation.

To prove our claims from Section 6.2 that more storage space yields a faster algorithm, we ran a multi-collision search while limiting the available memory. When the memory is filled, each thread continues to search for collisions without adding new points. As a practical application of this computation, we chose the discrete logarithm in the multi-user setting. Hence, the data that we store for each distinguished point is the coefficient a, plus an integer representing the user. Note that, this implementation includes only

L_1	Runtime t_1	L_2	Runtime t_2	Parallel performance
1	2459 s	2	1699 s	0.72
7	776 s	14	411 s	0.94
14	411 s	28	210 s	0.97

Table 6.7: Runtime and Parallel performance of the attack on ECDLP. Results are based on 100 runs per $L_i \in L$.

a collision search phase, and is missing the graph generation and the computation of discrete logs. Results in Table 6.8 show that the PRTL yields a better runtime compared to a classic hash table due to the more efficient memory use. In this table, values for 1GB memory limit are an average of 100 runs and values for 2GB and 4GB memory limits are an average of 10 runs.

Collisions	Memory limit	Runtime		Stored points	
		PRTL	Hash table	PRTL	Hash table
4000000	1 GB	34,64 h	58,80 h	46820082	12912177
16000000	2 GB	88,18 h	137,46 h	93640161	25824345
50000000	4 GB	203,24 h	276,80 h	168325978	51648716

Table 6.8: Runtime for multi-collision search for a 55-bit curve using PRTLs and hash tables.

Chapter 7

Index Calculus

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 some algebraic curves. It includes the Number Field Sieve (NFS) [LJMP93], dedicated to logarithms in $\mathbb{Z}_q$ and the algorithms of Gaudry [Gau09] and Diem [Die11] for algebraic curves defined over $\mathbb{F}_{q^n}$, where $q = p^k$. A crucial step of the latter application is to solve the *point decomposition problem* (PDP). The PDP can be reduced to the problem of solving a Boolean polynomial system, which is usually solved using algebraic techniques, such as the ones described in Chapter 2. In our work, we take important steps towards fully replacing algebraic techniques for solving PDP with constraint programming ones. Specifically, this chapter is concentrated on binary elliptic curves over prime-degree extension fields. First, we model the point decomposition problem as a logical formula and we define it as an instance of the SAT problem. The formula is then solved with WDSat, as well as with other generic SAT solvers and a Gröbner basis method, for comparison. Our WDSat solver was initially conceived for solving instances derived from this problem. As such, it includes two extensions that, to this date, have yielded better running times only for instances derived from the PDP. First, we propose a symmetry breaking technique, that allows us to eliminate symmetric solutions without enlarging the model and with no additional computational cost. Secondly, we propose a novel preprocessing technique based on the Minimal Vertex Cover (MVC) Problem in graph theory. This technique is used before the solving phase. Hence, it can be considered as a preprocessing that statically determines the order of branching variables. For ANF instances, we consider that an optimal order of branching variables is the one that will lead as fast as possible to a linear polynomial system. The contributions of this work are divided between [TID20c] and [TID20b].

This chapter is organized as follows. Section 7.1 gives an overview of the index calculus algorithm. First, we give a historical background, and then we explain how the attack is adapted for the ECDLP. Section 7.2 details the logical models used in our experiments. Section 7.3 explains the symmetry breaking technique that we implemented as an extension of our WDSat solver. Section 7.4 describes our preprocessing technique. In Section 7.5 we give worst time complexity estimates for solving a PDP instance and derive the complexity of our SAT-based index calculus algorithm. Finally, Section 7.6 presents benchmarks obtained with our implementation. We compare this against results obtained using Magma's F4 implementation and several available best generic SAT-solvers.

7.1 Background

7.1.1 Classical index calculus

Originally, the index calculus denoted a method for computing discrete logarithms in the multiplicative group of a finite field. It was first formulated as such by Maurice Kraitchik in 1922 [Kra26]. However, the core ideas behind this algorithm are even older. They can be traced back to computation methods for discrete logs from the 19$^{\text{th}}$ century. In fact, the name of the algorithm is linked to these origins. *Index* is just the original term for the discrete logarithm, and *calculus* stands for a *method of calculation* (not to the branch of mathematics).

Definition 7.1.1 *Let p be a prime number. Then a primitive root mod p is a natural number $A < p$ such that for every natural number B relatively prime to p there exists some $x \in \mathbb{N}_0$ such that $A^x \equiv B \mod p$.*

Definition 7.1.2 *Let A be a primitive root mod p, and let B be a natural number relatively prime to p. Then the index of B mod p to the base A, denoted as $\text{ind}_A(B)$, is the smallest $x \in \mathbb{N}_0$ that satisfies $A^x \equiv B \mod p$.*

These definitions date from Disquisitiones Arithmeticae by Carl Friedrich Gauss published in 1801 [GWC86]. Today, it is common to talk about generators and discrete logarithms, instead of primitive roots and indices. In the 19$^{\text{th}}$ century, tables of indices were used alongside tables of the usual logarithm. Such tables were published, for instance, in Jacobi's book Canon Arithmeticus [Jac39]. This book from 1839 contains tables of primitive roots and indices for primes up to 1000, computed by iteration, by many contributors.

However, to find an index without the use of modern calculators, it is not necessary to have an entire table of indices. Having a table of indices of prime numbers is enough. Given such a table, we can compute other indices using the following observations, which are at the core of the index calculus method.

- Any natural number can be factored into prime numbers.
- As with the ordinary logarithm, there is a link between the multiplication of natural numbers and the addition of indices. Namely, we have that

$$\text{ind}_A(B \cdot C) \equiv \text{ind}_A(B) + \text{ind}_A(C) \mod p - 1. \tag{7.1}$$

Hence, if we know the indices of all prime numbers below a certain bound S and if a number can be factored into prime numbers below this bound, then we can easily compute the index of this number. The bound S is usually referred to as the smoothness bound and the set of primes below the smoothness bound is called a factor base. A number that can be factored into primes below S is called S-smooth. To compute the index of one number, without a table of indices, Kraitchik proposed a method based on collecting relations and linear algebra [Kra26], that later became known as the index calculus method. Let us introduce this method with a toy example.

Example 7.1.1 Let $B = 28$ be a natural number and let $A = 2$ be a primitive root mod $p = 47$. We are looking to compute $x = \text{ind}_A(B) \mod p$. Let us fix $S = 7$ and thus,

obtain the factor base $\mathcal{F} = \{R_1, R_2, R_3, R_4\} = \{2, 3, 5, 7\}$. Then, we look for relations of the form

$$\prod_{j=1}^{4} R_j^{r_j} \equiv A^r \bmod p, \tag{7.2}$$

and, as per Equation (7.1), we derive linear relations on indices, of the form

$$\sum_{j=1}^{4} r_j \operatorname{ind}_A(R_j) \equiv r \bmod p-1. \tag{7.3}$$

Such relations form a linear system, that when solved, derives the index of all elements of the factor base. Thus, to derive a system with a unique solution, we need the number of relations to be equal to the number of elements in the factor base, which is 4 in our example. To find these relations, we choose random powers of 2 and check if they can be split over the factor base. For instance, we start with the trivial relation $2^1 \equiv 2 \bmod p$. Then, we choose 2^7 and obtain

$$2^7 \equiv 34 = 2 \cdot 17 \pmod{47}.$$

Clearly, 2^7 can not be split over the chosen factor base, so we discard it. Continuing with this approach, we end up with the following relations of the form (7.2).

$$\begin{aligned} 2^2 &\equiv 2 && \pmod{47} \\ 2^8 &\equiv 21 = 3 \cdot 7 && \pmod{47} \\ 2^{12} &\equiv 7 && \pmod{47} \\ 2^{18} &\equiv 25 = 5^2 && \pmod{47}. \end{aligned} \tag{7.4}$$

From (7.4), we derive 4 relations of the form (7.3), and we obtain the following linear system over $\mathbb{Z}/46\mathbb{Z}$.

2	3	5	7	
1	0	0	0	1
0	1	0	1	8
0	0	0	1	12
0	0	2	0	18

Solving this system, we obtain

2	3	5	7	
1	0	0	0	1
0	1	0	0	42
0	0	1	0	9
0	0	0	1	12

and we derive $\operatorname{ind}_2(2) = 1$, $\operatorname{ind}_2(3) = 42$, $\operatorname{ind}_2(5) = 9$ and $\operatorname{ind}_2(7) = 12$. The index of B can be found only if B is S-smooth, which is the case in our example. We conclude with the following computation

$$\operatorname{ind}_2(28) = \operatorname{ind}_2(2^2 \cdot 7) = 2 \cdot \operatorname{ind}_2(2) + \operatorname{ind}_2(7) = 14.$$

Let us summarize the approach that we used in Example 7.1.1 in three steps. More generally, the method is applied for a finite cyclic group G generated by g, where the discrete log problem is to compute x, such that for $h \in G$ we have that $h = [x]g$. Let N be the order of G. We continue with the additive notation.

1. Choice of an appropriate factor base $\mathcal{F} = \{g_1, \ldots, g_s\}$, such that $\mathcal{F} \subseteq G$.
2. *Decomposition phase* : compute random integers a_i and try to decompose $[a_i]g$ into the factor base. This is also called the *relation search phase*, since every successful decomposition of the form $[a_i]g = \sum_{j=1}^{s}[p_{ij}]g_j$ is called a *relation*.
3. *Linear algebra* : when $k \geq s$ linearly independent relations are found and the matrices $A = (a_i)$ with $1 \leq i \leq k$ and $M = (p_{ij})$ with $1 \leq i \leq k$ and $1 \leq j \leq s$ are stored, use linear algebra to find a solution to the linear system $MX = A \bmod N$. Vector $X = (x_1, \ldots, x_s)$ contains the discrete logs to the base g of all elements in the factor base.

After all discrete logs of elements in the factor base are computed, the log of h can be found either by the method that we used in Example 7.1.1 or by a different relation involving h. For instance, a commonly used variant is to find a relation of the form

$$[a]g + [b]h = \sum_{j=1}^{s}[p_j]g_j$$

and compute the log of h as $x = (\sum_{j=1}^{s} p_j x_j - a)^{b^{-1}}$, provided that the greatest common divisor of b and N is 1. The variant described in this section is referred to as the precomputation-and-descent variant and it is much more efficient for the multi-user setting. Indeed, to compute more that one discrete log to the base g, we need to perform the three precomputation steps only once and then repeat the descent method for each discrete log.

The choice of the factor base is a significant step that can determine the efficiency of the overall algorithm. If we choose the S-bound to be too big, or analogously, if we choose $\mathcal{F}$ to be of high order, close to the order of G, then we have a higher probability of finding a relation, as most elements will split over the factor base. However, with such a choice, we will need to find many relations. Finding more relations takes more time, but also, results in a bigger matrix and thus, raises the cost of the linear algebra step. On the other hand, if we choose the S-bound to be too small, we obtain the opposite effect. The linear algebra step is easier, but the probability of finding a relation is lower. The trade-off will be discussed further in this chapter.

When performing the index calculus method in $\mathbb{F}_p^*$, we observe that there are elements that do not split into the factor base because they have one or more factors that are not below the S bound, called large prime factors. This observation has led to the idea of a large prime variation of the index calculus algorithm [Pom82]. The main idea behind the large prime variation is that relations that contain only one large prime should be kept because when we have two such relations that contain the same large prime, we can subtract them to obtain one relation over the factor base. According to the birthday paradox, these couples of relations occur frequently. Later, a double large prime variation was introduced [LM91]. In this algorithm, we store relations containing up to two large primes and we make use of a graph to find relation combinations that lead to the elimination of all large primes and yield a relation over the factor base.

We conclude this historical introduction with a result of Carl Pomerance [Pom87] which states that the expected running time of the index calculus method described in

this section for solving the classical discrete log is

$$\exp((\sqrt{2}+o(1))\cdot(\log(p)\cdot\log\log(p))^{1/2}).$$

7.1.2 Index calculus for ECDLP

Looking at the success of the index calculus attack in $\mathbb{F}_p^*$, a natural question is whether this method can be applied to attack the discrete log in the group of points on an elliptic curve. At first glance, we are confronted with two main problems. First, the notion of primality is not defined for points on an elliptic curve, so there is no obvious choice for a factor base. Secondly, the analogue of prime factorization is point decomposition and there is no straightforward method to decompose points on an elliptic curve. Before these two barriers were addressed, a workaround was found for computing discrete logs in the Jacobian of hyperelliptic curves defined over finite fields. For hyperelliptic curves, there is no group law on the points, but instead, the divisor class group of the curve is used. The polynomial representation of a divisor can be used to define the notions of smoothness and prime elements, which are the necessary building blocks for an index calculus attack. These notions were first defined in [ADH94] and the ideas were later used in [Gau00] to propose an attack that is asymptotically faster than Pollard's Rho method for curves of genus greater than 4. The attack for computing discrete logarithms in small genus hyperelliptic curves was improved by using the double large prime variation [GTTD07]. Another workaround to use the index calculus method for curves over extension fields $\mathbb{F}_{q^n}$ is to transfer the DLP defined on the curve to the Weil restriction of the curve, defined over $\mathbb{F}_q$. This idea is due to Frey [Fre98] and was used to construct the GHS attack [GHS02] for binary fields and Diem's attack [Die03] for the odd characteristic case.

For index calculus attacks, it is common to use a direct variant instead of the precomputation-and-descent variant that we introduced in Section 7.1.1. With the direct variant, we compute only one discrete logarithm. The following is a basic outline of the algorithm in additive notation.

1. Choice of an appropriate factor base $\mathcal{F}=\{g_1,\ldots,g_s\}$, such that $\mathcal{F}\subseteq G$. This step is not different between the two variants.

2. *Decomposition phase* : compute random integers a_i, b_i and try to decompose $[a_i]g+[b_i]h$ into the factor base. Every successful decomposition of the form $[a_i]g+[b_i]h=\sum_{j=1}^{n}[p_{ij}]g_j$ is called a relation.

3. *Linear algebra* : when $k\geq s$ linearly independent relations are found and the matrices $A=(a_i\ b_i)$ with $1\leq i\leq k$ and $M=(p_{ij})$ with $1\leq i\leq k$ and $1\leq j\leq s$ are stored, use linear algebra to find a kernel vector $v=(v_1...v_k)$ of the matrix M. The discrete log of h can be solved by computing $x=-(\sum_i a_iv_i)/(\sum_i b_iv_i) \bmod N$.

Applying the index calculus algorithm to the group of points on an elliptic curve became possible in 2004 when Semaev [Sem04] proposed a technique for solving the point decomposition problem. Solving this problem allows us to perform the second step of the index calculus method. To this end, Semaev defined summation polynomials that are related to the arithmetic operation on the curve. Since our work 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 with j-invariant different from 0, defined by an equation

$$E: y^2+xy=x^3+ax^2+b, \tag{7.5}$$

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 $\mathcal{O}$ 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 $P_1 \pm \ldots \pm P_m = \mathcal{O}$ if and only if $S_m(\mathbf{x}_{P_1}, \ldots, \mathbf{x}_{P_m}) = 0$. We have that

$$\begin{aligned} &S_2(X_1, X_2) = X_1 + X_2, \\ &S_3(X_1, X_2, X_3) = X_1^2X_2^2 + X_1^2X_3^2 + X_1X_2X_3 + X_2^2X_3^2 + b, \end{aligned} \tag{7.6}$$

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

$$\begin{aligned} &S_m(X_1, \ldots, X_m) = \\ &Res_X(S_{m-k}(X_1, \ldots, X_{m-k-1}, X), S_{k+2}(X_{m-k}, \ldots, X_m, X)). \end{aligned} \tag{7.7}$$

The polynomial S_m is symmetric and has degree 2^{m-2} in each of the variables.

Semaev's contribution addressed the problem of point decomposition, but his approach was not practical due to the lack of appropriate factor base. This obstacle was addressed in 2008 and 2009, when Gaudry [Gau09] and Diem [Die11] independently proposed a technique to perform the PDP of the index calculus attack for elliptic curves over extension fields, using Semaev's summation polynomials. Namely, when E is an elliptic curve defined over $\mathbb{F}_{q^n}$, the choice of factor base is the set of points whose x-coordinate lies in $\mathbb{F}_q$. Another idea in these algorithm that is crucial to making the index calculus attack possible is to combine the new approach with existing Weil descent techniques. Using this technique, the problem of finding the zeros of a summation polynomial is reduced to the problem of solving a multivariate polynomial system of equations over $\mathbb{F}_q$. The system is then solved using one of the algebraic techniques described in Chapter 2. 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 than generic methods for solving the discrete logarithm problem and Gröbner basis algorithms have become a well-established technique to solve these systems [JV12; JV13; FPPR12; PQ12; YJPST13]. 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 [FHJ+14].

When we consider elliptic curves defined over $\mathbb{F}_{2^n}$ with n prime, the base field is $\mathbb{F}_2$ and we can not use the choice of a factor base proposed in [Gau09] and [Die11]. A different choice was proposed in [Die13]. 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 $\mathcal{F}$ to be :

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

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 $P_1, \ldots, P_m \in \mathcal{F}$ such that $R = P_1 \pm \ldots \pm P_m$. Using Semaev's polynomials, this problem is reduced to the one of solving a multivariate polynomial system.

Definition 7.1.3 *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 Weil descent method in this case consists in rewritting the equation $f(\mathbf{x}_1, \ldots, \mathbf{x}_m) = 0$ as a system of n equations over $\mathbb{F}_2$, with ml variables. 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.

A common technique when working with Semaev's polynomials is to use a symmetrization process [Gau09] 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

$$\begin{aligned} e_1 &= \sum_{1 \leq i_1 \leq m} X_{i_1}, \\ e_2 &= \sum_{1 \leq i_1, i_2 \leq m} X_{i_1} X_{i_2}, \\ &\ldots, \\ e_m &= \prod_{1 \leq i \leq m} X_i. \end{aligned}$$

We denote by S'_{m+1} the polynomial obtained after symmetrizing S_{m+1} in the first m variables, i.e. we have $S'_{m+1} \in \mathbb{F}_{2^n}[e_1, \ldots, e_m, X_{m+1}]$.

Recent work on solving the decomposition problem has focused on using advanced methods for Gröbner basis computation such as Faugère's F4and F5 algorithms [Fau99; Fau02]. 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. However, when we consider elliptic curves defined over $\mathbb{F}_{2^n}$ with n prime, solving the PDP system via Gröbner bases 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 [YJPST13]. Finally, note that for random systems, pure Gröbner basis algorithms are both theoretically and practically slower than simpler methods, typically exhaustive search [BCC+13; LMPP18], hybrid methods [BFP09] and SAT solvers. Using a SAT solver for solving the PDP has recently been considered in the work of Galbraith and Gebregiyorgis [GG14], 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.

7.2 Model description

Since our interest is in elliptic curves defined over $\mathbb{F}_{2^n}$ with n prime, it is natural that we turn our attention towards combinatorics tools to solve the DLP. In our approach, the PDP is solved using a dedicated SAT solver, WDSat, however our experimental work includes the use of state-of-the-art SAT solvers as well as Magma's F4 available implementation. These different solving methods use a different model of the PDP, and the purpose of this section is to present in full detail the three models we used in our experiments. The starting point for deriving all three models is the approach in [YJPST13], where the authors report on experiments carried on systems obtained using a careful choice of the vector space V and application of the symmetrization process. Therefore, we start by

detailing their approach and presenting the algebraic model used when solving the PDP problem using Gröbner basis. Then we show the model we propose and finally, the CNF model used by Galbraith and Gebregiyorgis [GG14].

7.2.1 The Algebraic Model

Let t be a root of a defining polynomial of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. Following [YJPST13], we choose the vector space V to be the dimension-l subspace generated by $1, t, t^2, \ldots, t^{l-1}$. Assuming that $m(l-1) \leq n$ we can write:

$$\begin{aligned} e_1 &= d_{1,0} + \ldots + d_{1,l-1}t^{l-1} \\ e_2 &= d_{2,0} + \ldots + d_{2,2l-2}t^{2l-2} \\ &\ldots \\ e_m &= d_{m,0} + \ldots + d_{m,m(l-1)}t^{m(l-1)}, \end{aligned} \tag{7.8}$$

where the $d_{i,j}$ with $1 \leq i \leq m$, $0 \leq j \leq i(l-1)$ are binary variables. After choosing $\mathbf{x}_{m+1} \in \mathbb{F}_{2^n}$ and substituting $e_1, \ldots, e_m$ as in Equation (7.8), we get:

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

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

$$f_0 = f_1 = \ldots = f_{n-1} = 0. \tag{7.9}$$

One can see that with this symmetrization approach, the number of variables is increased by a factor m, but the degrees of the polynomials in the system are significantly reduced compared to a version that does not use symmetrization. Further simplification of this system can be obtained if the elliptic curve has a rational point of order 2 or 4 [GG14]. However, we did not implement this approach and we used the system in Equation (7.9).

The elementary symmetric polynomials e_i are written in terms of the $d_{i,j}$ binary variables, as in Equation (7.8). 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 (7.8), we derive the following equations:

$$\begin{aligned} d_{1,0} = c_{1,0} + &\ \ldots\ + c_{m,0} \\ d_{1,1} = c_{1,1} + &\ \ldots\ + c_{m,1} \\ &\ldots \\ d_{m,m(l-1)} = c_{1,l}\cdot &\ \ldots\ \cdot c_{m,l}. \end{aligned} \tag{7.10}$$

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

7.2.2 The CNF-XOR Model

To derive a CNF-XOR model from an algebraic model, we use the techniques described in Section 3.3.1. Recall that, we first need to create constraints in ANF by replacing all multiplications of variables with conjunctions of literals and by transforming an equation into 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 (7.10) is as follows:

$$\begin{aligned}
&\neg d_{1,0} \oplus c_{1,0} \oplus \ldots \oplus c_{m,0} \\
&\neg d_{1,1} \oplus c_{1,1} \oplus \ldots \oplus c_{m,1} \\
&\ldots \\
&\neg d_{m,m(l-1)} \oplus (c_{1,l} \wedge \ldots \wedge c_{m,l}).
\end{aligned} \tag{7.11}$$

The next step is to substitute all conjunctions in a XOR clause by a newly added variable. In our model, 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

$$\begin{aligned}
&(c' \vee \neg c_{i_1,j_1} \vee \neg c_{i_2,j_2} \vee ... \vee \neg c_{i_k,j_k}) \wedge \\
&(\neg c' \vee c_{i_1,j_1}) \wedge \\
&(\neg c' \vee c_{i_2,j_2}) \wedge \\
&\ldots \\
&(\neg c' \vee c_{i_k,j_k})
\end{aligned} \tag{7.12}$$

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.

After substituting conjunctions, we will refer to the set of clauses obtained from Equation (7.11) 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 CNF-XOR model.

Proposition 7.2.1 *Assigning all $c_{i,j}$ variables, for $1 \leq i \leq m$ and $0 \leq j \leq l-1$, leads to the assignment of all variables in the* CNF-XOR *model through unit propagation.*

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

1. Clauses in the X-substitutions set are obtained by transforming $c'_i \Leftrightarrow (c_{i_1,j_1} \wedge c_{i_2,j_2} \wedge ... \wedge c_{i_k,j_k})$. 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'_i 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'_i variables.

2. Clauses in the E-X-relations set are obtained by transforming the algebraic system in (7.10). 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 $d'_i \Leftrightarrow (d_{i_1,j_1} \wedge d_{i_2,j_2} \wedge ... \wedge d_{i_k,j_k})$. Similarly as with the X-substitutions set, we have only $d_{i,j}$ variables on the right of these equivalences and one single d'_i variable on the left. The assignment of all of the $d_{i,j}$ variables will thus yield the assignment of all d'_i variables.

4. At this point, all variables in the parity constraints in the set F were assigned and we simply check whether the obtained interpretation satisfies the formula.

We conclude that variables in all four types of clauses of our CNF-XOR model were assigned through unit propagation. □

This result shows that a DPLL-based algorithm only needs to make assignments of $c_{i,j}$ variables i.e. only $c_{i,j}$ variables are involved in the binary search.

7.2.3 The CNF Model

Most of the solvers used in our experiments read and process CNF formulas. To derive the CNF model from the CNF-XOR model, we used a classical technique that is recalled in Section 3.3.1. This technique consists in cutting up XOR-clauses into manageable size ones, before applying the CNF-encoding. The CNF model used in our experiments was obtained after cutting into ternary XOR-clauses, since any XORSAT problem reduces in polynomial time to a 3-XORSAT problem [BHvMW09]. MAGMA's implementation on the other hand, adopts a size 5 for XOR clauses. As we mentioned, the optimal size at which to cut the XOR-clauses depends on the nature of the model and can be determined by running experiments using different values. Running these experiments was out of the scope of our work, as the WDSAT solver does not use exclusive CNF instances.

We implemented all three models described in this section and we present Table 7.1 to serve as a comparison on the number of variables, equations and clauses. Note that, the WDSAT solver is also adapted to read ANF formulas, such as the one in Equation (7.11). However, since the ANF model is just a different representation of the algebraic model, we do not consider it separately. To identify the number of literals and XOR-clauses in the ANF model from Table 7.1, we can refer to the number of variables and equations in the Gröbner model. Values for the algebraic and CNF-XOR model are exact, whereas those for the CNF model are averages obtained from experiments presented in Section 7.6. This concerns the experiments where the value of m is 3 (the fourth summation polynomial).

Table 7.1: The number of variables and equations/clauses for the three models.

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

In [GG14], the authors 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 infer from Table 7.1 that the model they used has a significantly larger number of clauses and variables when compared to the CNF-XOR model.

7.3 Symmetry breaking technique

Since Semaev's summation polynomials are symmetric, if $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ is a solution, then all permutations of this indexed tuple are solutions as well. These solutions are equivalent and finding more than one is of no use for the PDP. When a DPLL-based SAT solver is used, we observe redundancy in the binary search tree. Indeed, for $m = 3$ when an indexed tuple $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ 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 $\mathbf{x}_1 \leq \mathbf{x}_2 \leq \ldots \leq \mathbf{x}_m$, where $\leq$ is the lexicographic order on $\{\text{FALSE}, \text{TRUE}\}^l$ with FALSE $<$ TRUE.

It would be tedious to add this constraint to the model itself, since this would imply adding new clauses and complexifying the SAT model. Instead, we decided to add this constraint in the DPLL algorithm using a tree-pruning-like technique. In a classical DPLL implementation (see Algorithm 4.1) we try out both FALSE and TRUE for the truth value of a chosen variable. In our symmetry breaking variation of DPLL, in some cases, the truth value of FALSE will not be tried out as all potential solutions after this assignment would not satisfy the constraint $\mathbf{x}_1 \leq \mathbf{x}_2 \leq \ldots \leq \mathbf{x}_m$. Our variation of DPLL is detailed in Algorithm 7.1 and the line numbers that distinguish it from the classical DPLL method in Algorithm 4.1 are in bold. Note that one crucial difference between the two algorithms is the choice of a variable on line 4. While this choice is arbitrary in Algorithm 4.1, in Algorithm 7.1 variables need to be chosen in the order from the leading bit of $\mathbf{x}_1$ to the trailing bit of $\mathbf{x}_m$. If this is not respected, our algorithm does not yield a correct answer. Indeed, the assignment of variables must be done with respect to the lexicographic order on $\{\text{FALSE}, \text{TRUE}\}^l$ with FALSE $<$ TRUE.

Using the notation in Section 7.2, $c_{i,j}$ corresponds to the j^{th} bit of the i^{th} $\mathbf{x}$-vector, where $1 \leq i \leq m$ and $0 \leq j \leq l-1$. We recall from Proposition 7.2.1 that assigning all $c_{i,j}$ variables in the CNF-XOR model leads to the assignment of all variables through unit propagation. In Algorithm 7.1, we decide whether to try out the truth value of FALSE for $c_{i,j}$ or not by comparing two $\mathbf{x}$-vectors bit for bit, in the same way that we would compare binary numbers. When we are deciding on the truth value of $c_{i,j}$ we have the following reasoning:

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

- If $c_{i-1,j}$ is TRUE, we only try out the truth value of TRUE for $c_{i,j}$ and we continue to do bit comparison since the potential $\mathbf{x}_i$ solutions are still greater than or equal to $\mathbf{x}_{i-1}$ at this point.

Lastly, we give further information which explains in full detail Algorithm 7.1. We use a flag denoted *compare* to instruct whether to do bit comparison at the current search tree level or not. On line 6 we reset the *compare* flag to TRUE since $c_{i,j}$, when $j = 0$, corresponds to a leading bit of the next $\mathbf{x}$-vector. Finally, if-conditions on line 8 have to be checked in the specified order.

Algorithm 7.1 Function DPLL_BR_SYM(F, *compare*) : DPLL algorithm coupled with our symmetry breaking technique.

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

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

The ASSIGN procedure, which was detailed in Algorithm 4.2, assigns the specified literal to TRUE in a formula F, simplifies F and infers truth values for other literals. Recall that, the BACKTRACK procedure is used to undo all changes made to F after the last truth-value assignment. For more details on how these procedures are handled in the WDSAT implementation, see Chapter 4.

As an example, in Figure 7.1 we show a pruned binary search tree that is built from Algorithm 7.1. This tree illustrates the case where $m = 3$ and $l = 2$. To show the complete binary tree, we consider that in this example of execution of Algorithm 7.1, on each branch, there is no conflict until the leaf is reached. The branches that are not visited due to the symmetry breaking technique are presented with dashed lines.

7.4 Branching order

For DPLL and CDCL-based algorithms, it is well known that the number of conflicts needed to prove the inconsistency is correlated to the order in which the branching variables are assigned. Among the state-of-the-art branching rules one can find two categories according to the type of heuristics. The first are based on Maximum number of Occurrences in the

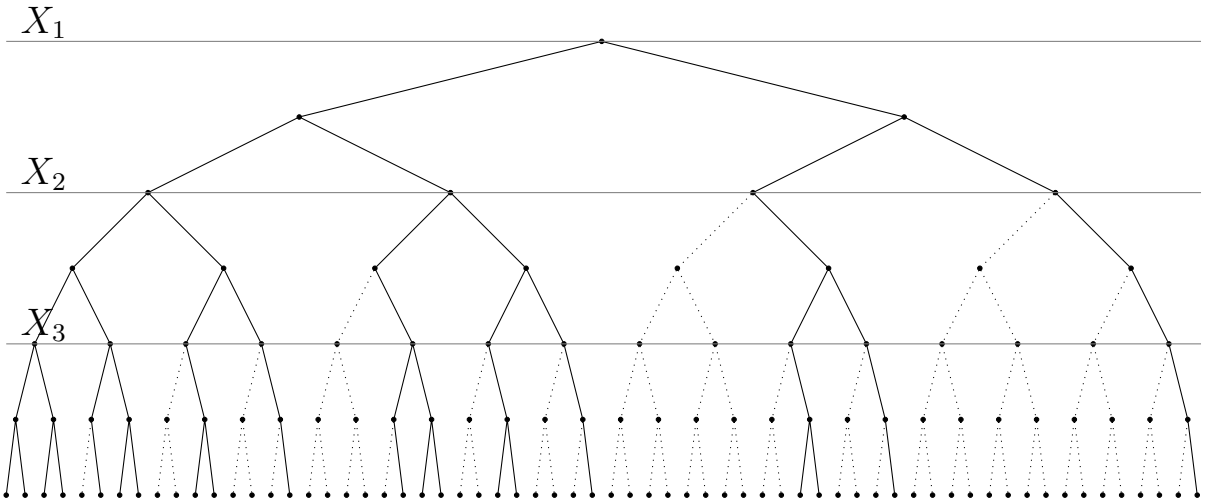


Figure 7.1: A complete binary search tree constructed by Algorithm 7.1 when $m = 3$ and $l = 2$

Minimum clauses Size (MOMs), whereas the second adopt the Variable State Independent Decaying Sum (VSIDS) branching heuristic.

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

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

Example 7.4.1 Consider the following Boolean polynomial system:

$$\begin{aligned} &\mathbf{x}_1 + \mathbf{x}_2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_4\mathbf{x}_5 = 0 \\ &\mathbf{x}_1 + \mathbf{x}_2\mathbf{x}_3 = 0 \\ &\mathbf{x}_1 + \mathbf{x}_3\mathbf{x}_5 + \mathbf{x}_6 = 0 \\ &\mathbf{x}_1 + \mathbf{x}_2\mathbf{x}_5\mathbf{x}_6 + \mathbf{x}_6 = 0 \end{aligned} \tag{7.13}$$

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

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

Definition 7.4.1 *A vertex cover is a subset of vertices such that for every edge (v_i, v_j) of the graph, either v_i or v_j is in the vertex cover. Given an undirected graph, the Mini-*

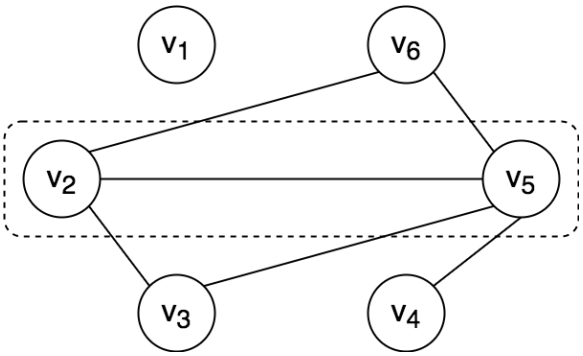


Figure 7.2: Graph derived from Example 7.4.1

mum Vertex Cover Problem is a classic optimization problem of finding a vertex cover of minimal size.

Definition 7.4.2 *A monomials connectivity graph is an undirected graph derived from a Boolean polynomial system as follows.*

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

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

Figure 7.2 shows the monomials connectivity graph derived from Example 7.4.1. The MVC of this graph is $\{v_2, v_5\}$. As a result, when all variables in the subset $\{\mathbf{x}_2, \mathbf{x}_5\}$ are assigned, the remaining polynomial system is linear. We give here the system derived after the assignment $\mathbf{x}_2 = 1$ and $\mathbf{x}_5 = 1$.

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

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

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

When variables are assigned in the order defined by this preprocessing technique, the worst-case time complexity of a DPLL-based algorithm drops from $O(2^k)$ to $O(2^{k'})$, where k' is the number of vertices in the MVC set. Note that the MVC of a complete graph is equal to the number of its vertices. Consequently, when the monomials connectivity graph

of a Boolean polynomial system is complete, solving the system using this preprocessing technique is as hard as solving the system without it.

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

7.4.1 MVC and summation polynomials

Let us look at the first nontrivial summation polynomial, which is S_3. Figure 7.3 shows the graph of a polynomial system derived from S_3 when l is 5. The system derived from a summation polynomial before symmetrization contains no intermediate variables. Hence, the number of variables in the system is ml, which is 10 in this example. The monomials connectivity graph is represented by an adjacency matrix that has the graph vertices in both the rows and the columns. The row (resp. column) of vertex v_i is denoted by (i mod 10). This simply allows us to have a uniform width of rows and columns, which makes for a better visual representation. The value in each cell shows if there exists an edge between the pair of vertices of the corresponding row and column. If an edge exists, we write ∘, otherwise we write •. The graph in Figure 7.3 has two minimal vertex covers,

Figure 7.3: Monomials connectivity graph derived from the model of S_3 when $l = 5$

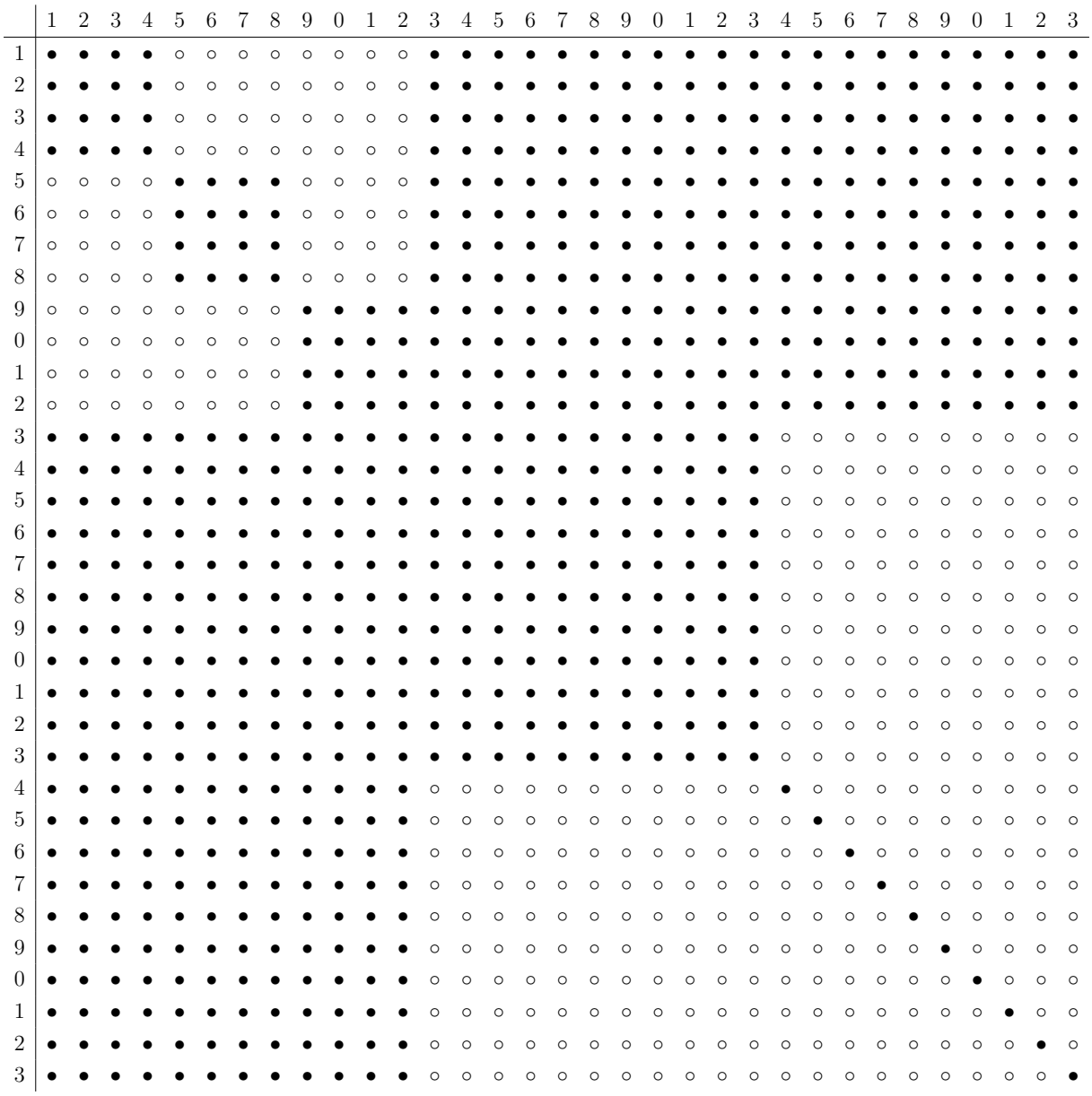
	1	2	3	4	5	6	7	8	9	0
1	•	•	•	•	•	∘	∘	∘	∘	∘
2	•	•	•	•	•	∘	∘	∘	∘	∘
3	•	•	•	•	•	∘	∘	∘	∘	∘
4	•	•	•	•	•	∘	∘	∘	∘	∘
5	•	•	•	•	•	∘	∘	∘	∘	∘
6	∘	∘	∘	∘	∘	•	•	•	•	•
7	∘	∘	∘	∘	∘	•	•	•	•	•
8	∘	∘	∘	∘	∘	•	•	•	•	•
9	∘	∘	∘	∘	∘	•	•	•	•	•
0	∘	∘	∘	∘	∘	•	•	•	•	•

which are $\{v_1, \ldots, v_5\}$ and $\{v_6, \ldots, v_{10}\}$. In general, the MVC of every graph derived from S_3 is either $\{v_1, \ldots, v_l\}$ or $\{v_{l+1}, \ldots, v_{2l}\}$. This gives us an MVC of size l when the number of variables in the corresponding system is $2l$. In other words, the preprocessing technique reveals that when either vector X_1 or vector X_2 is assigned in all variables, the remaining polynomial system is linear. This is not surprising when we look at S_3 is Equation (7.6). We observe that in S_3, there is no term containing X_i^d with $1 \leq i \leq m$, where d is not a power of 2. As a result, in the polynomial system obtained after a Weil descent, there is no monomial comprised of more than one variable from the same vector X_i. When this is the case, all variables from this vector can be left outside the MVC and thus, the size of the MVC is lesser than the number of variables in the polynomial system. Finding such a case in a summation polynomial is significant, as we have shown that the complexity

of the PDP is exponential in the size of the MVC of the corresponding polynomial system. Note that, finding a summation polynomial where two distinct vectors X_i and X_j do not appear simultaneously in any term, would yield a similar result. However, to the best of our knowledge, there is no such a case for $m > 2$. We have computed S_4 and S_5 to confirm and we have no reason to believe that we will find such an occurrence for $m > 4$.

For the fourth summation polynomial, we experimented with both the symmetric and the non-symmetric version for PDP systems and found, as in [YJPST13], that the symmetric version yields better results. Since S_4' is used in our experiments, we computed the corresponding graph and we show the example of $l = 4$ in Figure 7.4. We use the same representation as in Figure 7.3. Using our preprocessing technique, we found that

Figure 7.4: Monomials connectivity graph derived from the model of S_4' when $l = 4$



the MVC of a graph derived from S_4' is $\{v_1, \ldots, v_l, v_{l+1}, \ldots, v_{2l}, v_{6l}, \ldots, v_{9l-3}\}$. This means that when vectors X_1, X_2 and e_3 are assigned in all variables, the remaining polynomial system is linear. In this case, the size of the MVC is $(5l-2)$. Hence, if we choose the MVC as a criterion for the branching order, we obtain a linear system after $(5l-2)$ assignments. However, as per Proposition 7.2.1, we know that in the general case, when vectors X_i with $1 \leq i \leq m$ are assigned in all variables, the values of all other variables in the system

are propagated. Thus, if we use our findings in Proposition 7.2.1 to determine the order of branching variables for S_4', the system is solved after $3l$ variables are assigned. It is clear that the second choice yields better results. In a general manner, for $m > 2$, the branching order should be chosen as per our findings in Proposition 7.2.1 and our complexity analysis is based on this choice.

7.5 Complexity analysis

As we explained in Section 3.2, 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 7.2.1, we only reason on the $c_{i,j}$ variables from the CNF-XOR model. Therefore, $h = ml$ and the worst-case complexity for the PDP is 2^{ml}. Furthermore, using the symmetry breaking technique explained in Section 7.3, 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

$$\frac{2^{ml}}{m!}. \tag{7.14}$$

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-substitutions 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 (7.12)). In total, the number of clauses in the X-substitutions set is

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

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

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

Remark 7.5.1 Using this analysis, we approximate the number of clauses, denoted by C, for $m = 3$, as all experiments presented in this chapter are performed using the fourth summation polynomial.

$$\begin{aligned} C &\approx \binom{3}{2} \cdot 3l^2 + \binom{3}{3} \cdot 4l^3 + \left(\binom{3}{2}\right) \cdot 3(3l-2)^2 + (6l-3) + n \approx \\ &\approx 4l^3 + 171l^2 - 210l + n + 69. \end{aligned} \tag{7.15}$$

In practice, many monomials have no occurrence in the system after the Weil descent. In fact, the value in Equation (7.15) is a huge overestimate and exact values for $l \in \{6, \ldots, 11\}$ are shown in Table 7.1.

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

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

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

Theorem 7.5.1 *The estimated complexity of the relation search phase 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 linearly independent relations. Following [Die13], 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 the WDSAT solver coupled with the breaking symmetry technique 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* [FPPR12]. 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 an 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 (see Equation (7.14)).

7.5.1 The third summation polynomial

When the third summation polynomial is used, the choice of branching variables is determined by the MVC preprocessing technique. As shown in Section 7.4, the size of the MVC is l. Since in the case of S_3 the system is solved after the first solution vector is assigned in all variables, the breaking symmetry technique does not make any difference in this case. This concludes that the worst-case number of conflicts reached for one PDP computation when S_3 is used is 2^l.

The number of OR-clauses in SAT instances derived from S_3 can be computed using our analysis in Section 7.4.1. As per this analysis, in the Boolean polynomial system derived from S_3, there are exactly l^2 different monomials of degree 2, which will in turn produce $3l^2$ OR-clauses in the CNF part of the CNF-XOR model. As we are working with the non-symmetric version of the third polynomial, the number of XOR-clauses is simply n. We conclude that

$$C \quad = \quad 3l^2 + n.$$

Theorem 7.5.2 *The estimated complexity of the relation search phase of the index calculus algorithm for solving* ECDLP *on a curve defined over $\mathbb{F}_{2^n}$, using the third summation polynomial and a factor base given by a vector space of dimension l, is $\tilde{O}(2^{n+1})$, where the $\tilde{O}$ hides a polynomial factor in l.*

Proof. As previously, the number of linearly independent relations that need to be found is 2^l. When $m = 2$, the probability that a random point can be written as a sum of m factor basis elements is heuristically approximated by $\frac{2^{2l}}{2^{n+1}}$. The time complexity for the full decomposition phase, using the WDSAT solver coupled with the breaking symmetry technique is $T(3l^2 + n)2^{n+1}$. □

7.6 Experimental results

To support our claims, we conducted experiments using S_3 and S_4' on binary Koblitz elliptic curves [Kob92a] defined over $\mathbb{F}_{2^n}$. We compare the WDSat solver presented in this thesis to the following approaches: the best currently available implementation of Gröbner basis (F4 [Fau99] in Magma [BCP97]), the solvers MiniSat [ES04], Glucose [AS09], MapleLCMDistChronoBT [NR18], CaDiCaL [Bie] and CryptoMiniSat [SNC09]. Note that MapleLCMDistChronoBT and CaDiCaL are the winners in the main track of the sat competition [vMF] in 2018. All tests were performed on a 2.40GHz Intel Xeon E5-2640 processor. Our Weil descent implementation used to generate benchmarks is open source [TID20a].

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. Solvers MiniSat, Glucose, MapleLCMDistChronoBT and CaDiCaL process a cnf model input, whereas CryptoMiniSat (CMS) and WDSat use the cnf-xor model. WDSat can also process an algebraic model in anf form directly. Using the cnf-xor model is a huge advantage, as it has far fewer clauses and variables than the cnf model. Gaussian elimination can be beneficial for sat instances derived from cryptographic problems. However, it has been reported to yield slower running times for some instances, as performing the operation is very costly. For this reason, CryptoMiniSat and WDSat do not include Gaussian elimination by default, but the feature can be turned on explicitly. We experimented with both variants for both xor-able solvers.

With WDSat we set a custom order of branching variables, which allowed us to make use of unit propagation as explained in Proposition 7.2.1 and branch only on the $c_{i,j}$ variables. For instances derived from S_3, the branching order is defined by the mvc preprocessing technique. CryptoMiniSat does not allow for custom order of branching variables in the current version as the authors report that it leads to slower running times in most cases. We added this feature to the source code of CryptoMiniSat and we ran tests both with a custom order and with the order chosen by the solver.

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

In all Tables, 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, [YJPST13] indicates that the computational complexity of Gröbner bases is lower when a solution does not exist. Unless it is otherwise specified, in all tables in this section, running times are in seconds and memory use is in MB.

Benchmarks derived from S_3

Results on solving the third summation polynomial ($m = 2$) are shown in Table 7.3. The parameters used to obtain these benchmarks are $n = 41$ and $l = 20$. As a result, we obtained a Boolean polynomial system of 41 equations in 40 variables. In this Table, results are an average of 100 runs. As different variants of our solver can yield better results for different benchmarks, we compared all variants to decide on the optimal one. We also tested the solver with and without our preprocessing technique (denoted by mvc

Table 7.2: Comparing different versions of WDSat for solving the third summation polynomial.

l	n	WDSat version	SAT		UNSAT	
			Runtime	#Conflicts	Runtime	#Conflicts
20	41	WDSat_{XG}	6028.4	200957178	11743.2	354094821
		$\text{WDSat}_{\text{XG+MVC}}$	639.6	21865963	2973.0	94489361
		$\text{WDSat}_{\text{XG-EXT}}$	375.9	4911099	870.1	10789518
		$\text{WDSat}_{\text{XG-EXT+MVC}}$	**4.2**	**27684**	**13.5**	**86152**

Table 7.3: Comparing different approaches for solving the third summation polynomial.

l	n	Solving approach	SAT		UNSAT	
			Runtime	#Conflicts	Runtime	#Conflicts
20	41	Gröbner	16.8	N/A	18.7	N/A
		MiniSat	> 600		> 600	
		Glucose	> 600		> 600	
		MapleLCMDistChronoBT	> 600		> 600	
		CaDiCaL	> 600		> 600	
		CMS	29.0	226668	84.3	627539
		CMS+mvc	237.4	1263601	> 600	
		$\text{WDSat}_{\text{XG-EXT+MVC}}$	**4.2**	**27684**	**13.5**	**86152**

in the tables). The results in Table 7.2 show that WDSat yields optimal results for these benchmarks when the xg-ext method is used coupled with the preprocessing technique. This is due to the fact that the size of the mvc obtained by the preprocessing technique is 20, whereas the number of variables in the system is 40. We confirm that when the xg-ext method of WDSat is used coupled with the preprocessing technique, the worst-case time complexity of the models derived from S_3 drops from 2^k to $2^{\frac{k}{2}}$, where k is the number of variables in the polynomial system.

By analyzing the average running time and the average number of conflicts in Table 7.3, we see that the chosen variant of the WDSat solver outperforms all other approaches for solving instances derived from the third summation polynomial. For these experiments, we set a timeout of 10 minutes. Line CMS+mvc in Table 7.3 corresponds to the case where CryptoMiniSat is coupled with the mvc preprocessing technique and we report that in this case, only 9 out of 100 unsatisfiable and 54 out of 100 satisfiable instances were solved. This confirms that the mvc preprocessing technique is strongly linked to our xg-ext method. Indeed, when the xg-ext method is not used, one can not guarantee that when all variables from the mvc are assigned the system becomes linear. This is confirmed also by looking at the number of conflicts for the CMS+mvc approach, which is greater than $2^{\frac{k}{2}}$ even for benchmarks that were solved before the timeout. Recall that $\frac{k}{2}$ is the size of the mvc. On the other hand CryptoMiniSat without the preprocessing technique succeeds in solving these instances after less than $2^{\frac{k}{2}}$ conflicts. We conclude that the searching technique in CryptoMiniSat used to decide on the next branching variable is optimal for this solver.

The solvers which are not xor-enabled did not solve any of the 200 satisfiable and unsatisfiable instances before the 10-minute timeout. This is not surprising as instances

derived from the third summation polynomial are solved a lot faster when a GE technique is used. For reference, these solvers are able to solve within a day instances with parameters $l = 15$ and $n = 31$.

Benchmarks derived from S_4'

Results on solving the symmetric version of the fourth summation polynomial ($m = 3$) are shown in Table 7.5, where we compare different approaches, showing results for optimal variants of each solving tool. Running times of all variants of CryptoMiniSat and WDSat are given in Table 7.4. We experimented with different values of n for each l and we performed tests on 20 instances for each parameter size.

We set a timeout of 10 hours and a memory limit of 200GB for each run. Using MiniSat and Glucose, 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. Indeed, [YJPST13] and [ST13] show experiments using the fourth summation polynomial with $l = 6$, whereas the highest parameter size achieved in [GG14] is $l = 8$.

Table 7.5 shows the average runtime in seconds, the average number of conflicts and the average memory use in MB. The WDSat solver allocates memory statically, according to predefined constant memory requirements. This explains why memory averages do not vary much between the different size parameters, or between satisfiable and unsatisfiable instances.

Table 7.4: Comparing different variations of CryptoMiniSat and WDSat for solving the PDP.

Solving approach	l	n	SAT			UNSAT		
			Runtime	#Conflicts	Mem.	Runtime	#Conflicts	Mem.
CMS	6	17	133.983	775948	48.4	363.513	1709971	59.5
		19	560.080	3396192	64.1	1172.740	5726372	70.1
	7	19	1210.612	5713259	85.3	10258.351	26079224	117
		23	3637.032	12159752	80.4	19857.454	47086152	130
	8	23	9846.554	18509058	123	>10 hours		
		26	6905.477	13269631	115	>10 hours		
CMS_{GE}	6	17	119.866	677336	54.5	436.811	1877699	64.2
		19	224.484	1219840	58.7	615.952	2763754	76.5
	7	19	893.425	3722805	86.5	3587.929	8642108	107
		23	580.007	1753040	82.4	3253.786	8183887	132
	8	23	11265.010	19604250	155	>10 hours		
		26	3933.637	7920920	157	>10 hours		
CMS with Prop.7.2.1	6	17	15.673	61812	34.5	62.396	260843	39.3
		19	14.128	53767	33.2	64.563	259688	42.1
	7	19	176.463	484098	41.5	843.367	2077747	72.3
		23	300.021	638152	48.9	1012.412	2070190	73.6
	8	23	1700.949	2420937	76.7	11959.938	16756106	82.4
		26	3000.831	4179236	79.4	14412.193	16783213	81.8

Table 7.4: Comparing different variations of CryptoMiniSat and WDSat for solving the PDP.

Solving approach	l	n	SAT			UNSAT		
			Runtime	#Conflicts	Mem.	Runtime	#Conflicts	Mem.
CMS_{GE} with Prop.7.2.1	6	17	17.698	62161	39.1	86.049	294428	63.2
		19	16.301	52730	39.8	88.738	293859	62.7
	7	19	220.037	**479197**	51.2	2551.277	2418051	72.5
		23	367.105	653673	59.4	1329.494	2380614	93.1
	8	23	2493.328	**2419268**	112	19058.671	19359334	164
		26	4956.952	**4171674**	126	19907.670	19534832	167
WDSat with Prop.7.2.1	6	17	**.601**	49117	**1.4**	**3.851**	254686	**1.4**
		19	**.470**	38137	**1.4**	**3.913**	255491	**1.4**
	7	19	**9.643**	534867	**16.7**	**44.107**	2073089	**16.7**
		23	**9.303**	477632	**16.7**	**47.347**	2067168	**16.7**
	8	23	**68.929**	2646071	**16.8**	**525.057**	16666331	**16.8**
		26	**185.480**	6261107	**16.9**	**533.607**	16684378	**16.9**
$WDSat_{GE}$ with Prop.7.2.1	6	17	9.193	**48178**	**1.4**	56.718	**253123**	**1.4**
		19	7.041	**36835**	**1.4**	58.876	**252799**	**1.4**
	7	19	169.629	528383	**16.7**	736.863	**2062232**	**16.7**
		23	159.101	**473223**	**16.7**	779.432	**2060501**	**16.7**
	8	23	1290.702	2630567	**16.8**	9124.361	**16639322**	**16.8**
		26	3404.765	6231289	**16.9**	9623.677	**16636122**	**16.9**

Our experimental results show that performing Gaussian elimination on the system comes with a significant computational cost and yields a small decrease in the number of conflicts (see Table 7.4). As this was the case for all instances derived from the Weil descent on S_4', we concluded that GE is not beneficial for this model. We attribute this fallout to the particularly small improvement in the number of conflicts, compared to the significant computational cost of performing the GE technique. Indeed, the monomials connectivity graph of the fourth summation polynomial is complete and thus the size of the MVC is equivalent to the number of variables in the formula. This leads us to believe there is no optimal choice for the order of branching variables and the system generally does not become linear until the second-to-last branching.

Choosing the WDSat variant without GE as optimal, we continued experiments for bigger size parameters using this variant coupled with the symmetry breaking technique. Table 7.6 shows results for $l \in \{6, 7, 8, 9, 10, 11\}$ 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 values for l in this Table to that of the basic WDSat solver without the breaking symmetry extension in Table 7.5, we observe a speedup factor that rapidly approaches 6.[2] This confirms our claims in Section 7.5 that the symmetry breaking technique that we proposed yields a speedup by a factor of $m!$.

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

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

Table 7.5: Comparing different approaches for solving the PDP.

Solving approach	l	n	SAT			UNSAT		
			Runtime	#Conflicts	Mem.	Runtime	#Conflicts	Mem.
GRÖBNER	6	17	207.220	NA	3601	142.119	NA	3291
		19	215.187	NA	3940	155.765	NA	4091
	7	19	3854.708	NA	38763	2650.696	NA	38408
		23	3128.844	NA	35203	2286.136	NA	35162
	8	23			>200GB			>200GB
		26[1]			>200GB			>200GB
MINISAT	6	17	62.702	408189	12.7	270.261	1463309	24.2
		19	229.055	1778377	23.6	388.719	2439933	29.8
	7	19	406.918	1919565	33.6	6777.431	25180492	105
		23	12945.613	61610582	152	13260.586	59289671	163
	8	23	8027.974	63384411	256	>10 hours		
		26	>10 hours			>10 hours		
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		
CMS with Prop.7.2.1	6	17	15.673	61812	34.5	62.396	260843	39.3
		19	14.128	53767	33.2	64.563	259688	42.1
	7	19	176.463	**484098**	41.5	843.367	2077747	72.3
		23	300.021	638152	48.9	1012.412	2070190	73.6
	8	23	1700.949	**2420937**	76.7	11959.938	16756106	82.4
		26	3000.831	**4179236**	79.4	14412.193	16783213	81.8
WDSAT with Prop.7.2.1	6	17	**.601**	**49117**	**1.4**	**3.851**	**254686**	**1.4**
		19	**.470**	**38137**	**1.4**	**3.913**	**255491**	**1.4**
	7	19	**9.643**	534867	**16.7**	**44.107**	**2073089**	**16.7**
		23	**9.303**	**477632**	**16.7**	**47.347**	**2067168**	**16.7**
	8	23	**68.929**	2646071	**16.8**	**525.057**	**16666331**	**16.8**
		26	**185.480**	6261107	**16.9**	**533.607**	**16684378**	**16.9**

Comparing results for $l = 6$ and $l = 7$ in Table 7.6 with the equivalent results for the Gröbner basis method in Table 7.5, we observe that WDSAT is up to 300 times faster than Gröbner bases for the cases where there is no solution and up to 1700 times faster for instances allowing a solution. This is a rough comparison, as the factor grows with parameters l and n.

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

Table 7.6: Experimental results using the complete WDSAT solver. Running times are in seconds and memory use is in MB.

l	n	SAT			UNSAT		
		Runtime	#Conflicts	Mem.	Runtime	#Conflicts	Mem.
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

likely to yield faster running times for higher scale instances of that problem.

Open question 5 Our experimental work shows comparison only with complete SAT solvers. Another perspective would be to use incomplete SAT solvers, such as WalkSat, coupled with an XOR reasoning module.

7.6.1 Whole Point Decomposition Phase Computation

Previously shown experiments for solving PDP are done with arbitrary choices of parameters n and l. However, when performing a whole ECDLP attack, choosing the factor base is a crucial step in the index calculus method. The number of relations that needs to be found is exponential in l, as is the running time for one point decomposition (see (7.14)). However, taking a smaller l decreases the probability of successfully decomposing a randomly chosen point, and thus increases the number of times we solve the PDP.

To understand better the optimal ratio n/l, we computed the whole point decomposition phase for $n = 24$ using different l sizes. The experiment consists essentially in computing the PDP on instances for randomly chosen X_{m+1} until we find 2^l valid decompositions. Instances that turn out to not have a solution are tossed.

Results from these experiments are in Table 7.7. We present average running times in hours (Runtime), the number of generated satisfiable (#Generated SAT) and unsatisfiable (#Generated UNSAT) instances and the probability that a random point can be decomposed (P). This probability is, in fact, the ratio between the number of generated

Table 7.7: Whole Point decomposition phase computation.

n	l	Runtime	#Generated SAT	#Generated UNSAT	P	P_{approx}	#Runs
24	4	2.59 h	16	149475	.00010	.00004	10
	5	**1.83 h**	32	49484	.00064	.00032	10
	6	2.49 h	64	11478	.00557	.00260	10
	7	6.45 h	128	2994	.04275	.02083	10
	8	17.99 h	256	688	.37209	.16666	8

instances that have a solution and the total number of generated instances. We compare this to the heuristically approximated probability $\frac{2^{ml}}{m!2^n}$, denoted by (P_{approx}). #Runs denotes the number of times we ran the experiment for each n/l ratio.

Further research is needed to fully understand the best choice of l, but current experimental results suggest that it is not $l \approx n/m$. With this ratio, even though there is a good chance of finding a composition for a randomly chosen point, the runtime of PDP and the number of relations needed are too high. The cost of the final phase linear algebra increases with l as well.

At the time of writing of this thesis, we do not have results for bigger values of n and l. For this reason, we refrain from drawing a conclusion from Table 7.7, as results can be ambiguous for such small parameters. As an example, the time to compute the Weil descent and derive the XOR model, is not negligible compared to the time to solve the instance in the cases of $l = 4$ and $l = 5$, as in these cases, the time to solve the instance is less than 60 milliseconds long. This might be the only reason we observe a decrease of the runtime between $l = 4$ and $l = 5$, whereas from the remaining results in Table 7.7 we observe that the time increases with l.

Open question 6 More research is needed to find the optimal l. For instance, it would be interesting to find the lower bound of l where 2^l linearly independent relations can be found regardless of the time it takes to find them.

Lastly, we experimented with the collision search [vOW99] generic method, using the source code that we created for our work in Chapter 6. Recall that, this implementation solves the discrete log problem in the case of prime field curves. We did not adapt the code for extension fields and the computation time for scalar 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 index calculus method when the WDSAT solver is used. In a uni-thread environment, a whole collision search computation for parameter $n = 59$ has an average runtime of 0.8 hours on our platform. Computing 2^l successful decompositions for parameters $n = 59$ and $l = 9$ would take more than 86 hours according to results in Table 7.6. The estimated running time becomes considerably higher 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.

Chapter 8

Other applications of logical cryptanalysis

The WDSat solver was developed for solving systems arising from a Weil descent, with the goal to be used for index calculus attacks. However, the solver can handle any logical formula in ANF or other standard forms. In this chapter, we investigate the use of WDSat in attacks on other cryptographic primitives. First, we use our solver to perform a logical attack on the Trivium stream cipher. Then, WDSat was tested on instances arising from the Multivariate Quadratic polynomial problem (MQ), with various levels of difficulty. We compare the performance of WDSat against CryptoMiniSat, as they are both XOR-enabled SAT solvers conceived for solving Boolean polynomial systems arising from cryptographic attacks. The two different models studied in this chapter allow us to get a better understanding of which properties of Boolean polynomial systems are better suited for WDSat. As we have seen, systems derived from the index calculus attack have few variables but are highly dense, or in other words, have long XOR-clauses. This is also the case for MQ instances. On the other hand, Trivium systems are sparse and have a higher number of variables.

This chapter is organized as follows. Section 8.1 is dedicated to the Trivium stream cipher. We show experimental results on attacks using both WDSat and CryptoMiniSat, followed by a complexity discussion of logical cryptanalysis of Trivium. In Section 8.2 we give initial results of a general attack on multivariate cryptography schemes using WDSat .

8.1 The Trivium stream cipher

The purpose of stream ciphers is to emulate the functionality of the one-time-pad (OTP), which is a theoretically unbreakable cipher. In the OTP encryption scheme, the ciphertext is obtained by combining the plaintext with a keystream using, for instance, the binary XOR operator. The keystream is a random sequence that is at least as long as the plaintext. The perfect secrecy of this scheme was proven by Shannon in 1949. The greatest challenge in constructing an OTP encryption scheme is the requirement for the keystream to be random and of size at least equal to the size of the plaintext. Since the requirement is not practical, a stream cipher uses a smaller secret key to generate a pseudorandom keystream of the required size.

Trivium is a stream cipher designed in 2005 by C. De Cannière and B. Preneel [DC06] as a submission for the eSTREAM project [eST]. Its simple structure is specifically intended for fast hardware implementation. Trivium is comprised of three Nonlinear-Feedback Shift Registers (NLFSR). NLFSR is a shift register whose input bit is a non-

linear function of its previous state. In these registers are stored 288 bits representing Trivium's internal state, denoted $(s_1, \ldots, s_{288})$. This internal state is initialized using an 80-bit secret key vector K and an 80-bit vector IV holding the initial value. The initialization is defined as follows:

$$(s_1, \ldots, s_{93}) \leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0)$$
$$(s_{94}, \ldots, s_{177}) \leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots, 0)$$
$$(s_{178}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1).$$

At the end of each round, all registers are shifted by one bit and the first bit in each register is updated using the defined non-linear feedback function. The initialization process of Trivium consists in loading the state bits into the registers and performing 1152 rounds without producing an output. After this process, every next round produces one output bit obtained as a linear combination of six state bits. The iterative process of Trivium is shown in Algorithm 8.1, where Z denotes the number of generated keystream bits. The pseudo-code for the initialization process is similar, with the exception that line 5 is missing and i goes from 1 to 1155.

Algorithm 8.1 Trivium's iterative function for keystream generation.

Input: The number of bits to be generated, denoted Z.
Output: Keystream vector z.

1: **for** $i = 1$ to Z **do**
2: $t_1 \leftarrow s_{66} + s_{93}$
3: $t_2 \leftarrow s_{162} + s_{177}$
4: $t_3 \leftarrow s_{243} + s_{288}$
5: $z_i \leftarrow t_1 + t_2 + t_3$
6: $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
7: $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
8: $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
9: $(s_1, s_2 \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
10: $(s_{94}, s_{95} \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
11: $(s_{178}, s_{179} \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
12: **end for**

8.1.1 Model generation and experimental results

As most symmetric cryptographic primitives, Trivium can be described by Boolean functions with secret variables in the key bit-vector and public variables in the IV bit-vector. To generate the SAT model for Trivium, we produce a XOR constraint for every output bit, as an output bit is a non-linear combination of internal state bits. Our model contains only 288 variables corresponding to the bits of the internal state after the initialization process is finished. Hence, this constitutes an attack on the internal state, in contrast to attacks on the key. The modelization approach that we described is different from the one used in the Grain of Salt tool [Soo10b]. In models constructed using Grain of Salt, each input bit obtained from the non-linear feedback function is an additional variable, whereas, in the model we use, these input bits are recursively replaced by a non-linear combination of state bits from previous rounds, until we obtain a clause containing only bits from the initial state. Both approaches are common in the literature [SPF08; RS06; Soo10a; LJN14]. The model obtained using the Grain of Salt tool is useful for attacks that

Table 8.1: Comparing different approaches for solving Trivium. Running times are in seconds.

	SAT		UNSAT	
Solving approach	Runtime	#Conflicts	Runtime	#Conflicts
CMS_{GE}	**0.03**	**1336**	**0.03**	**1602**
WDSAT_{XG}	17.73	781202	32.83	1374475
$\text{WDSAT}_{\text{XG-EXT}}$	23.95	696383	44.15	1225498
$\text{WDSAT}_{\text{XG-EXT+MVC}}$	0.68	25982	1.50	58064

suppose the property of sparse polynomial systems [RS06]. In this model, each Trivium iteration adds three new variables and four equations. All equations are of degree at most two. Recall that, the number of Trivium iterations corresponds to the number of generated keystream bits. On the other hand, the model that we use in these experiments has 288 variables, and the number of equations is equal to the number of generated keystream bits. In this modelization, intermediate variables are substituted recursively, the degree of the equations increases with the number of Trivium cycles. More specifically, the first 66 equations are linear, then we have 82 degree-two equations, followed by 66 equations of degree three, and the degree continues to grow from there. We chose this model because, in general, our MVC preprocessing technique makes more sense with a model that does not contain intermediate variables.

Table 8.1 shows average running times (100 runs) and average number of conflicts for solving SAT instances derived from Trivium. The number of output bits we used is 148 and thus the derived boolean polynomials system has 148 equations and 288 variables.

The size of the MVC for this model is 122 and the exact variables whose corresponding vertices are in the MVC are shown in Table 8.2. Note that the size of the MVC grows with the number of generated output bits, since when more output bits are generated, more initial state bits are involved in non-linear feedback function outputs. We do not have a precise estimation of the number of output bits that need to be generated so that the derived Boolean polynomial system has a unique solution. In any case, the size of the MVC in this example is still greater than the size of the secret key, which is 80 bits. Thus, the MVC is an indicator that a SAT-based attack on this model for Trivium is unlikely to give better results than a brute force attack for recovering the entire key.

For benchmarking, we provided both solvers with 160 help bits out of the 288 bits from the initial state. This is a standard approach in the literature [SNC09; Soo10a]. Results in Table 8.1 show that the XG-EXT version coupled with the MVC preprocessing technique improves the performance of WDSAT for instances derived from the Trivium cipher. However, WDSAT is outperformed by CRYPTOMINISAT for the resolution of these instances, both in terms of running time and number of conflicts. We believe that this difference is due to the fact that the CDCL approach that CRYPTOMINISAT relies on is better suited for these instances than the basic DPLL behind WDSAT. In particular, the backtracking algorithm with clause learning involves *backjumping* techniques, which results in fewer conflicts.

8.1.2 Complexity discussion

The technique described in this section is a straightforward modelization of Trivium and does not result in an attack whose complexity is better than the brute force attack.

Table 8.2: Minimal Vertex Cover of Trivium when 148 keystream bits are generated.

x_{14}, x_{16}, x_{18}, x_{20}, x_{22}, x_{24}, x_{26}, x_{28}, x_{30}, x_{32}, x_{34}, x_{36}, x_{38},
x_{40}, x_{42}, x_{44}, x_{46}, x_{48}, x_{50}, x_{52}, x_{54}, x_{56}, x_{58}, x_{60}, x_{62}, x_{64},
x_{66}, x_{68}, x_{70}, x_{72}, x_{74}, x_{76}, x_{78}, x_{80}, x_{82}, x_{84}, x_{86}, x_{88}, x_{90},
x_{95}, x_{97}, x_{99}, x_{101}, x_{103}, x_{105}, x_{107}, x_{109}, x_{111}, x_{113}, x_{115},
x_{117}, x_{119}, x_{121}, x_{123}, x_{125}, x_{127}, x_{129}, x_{131}, x_{133}, x_{135}, x_{137},
x_{139}, x_{141}, x_{143}, x_{145}, x_{147}, x_{149}, x_{151}, x_{153}, x_{155}, x_{157}, x_{159},
x_{161}, x_{163}, x_{165}, x_{167}, x_{169}, x_{171}, x_{173}, x_{175}, x_{206}, x_{208}, x_{210},
x_{212}, x_{214}, x_{216}, x_{218}, x_{220}, x_{222}, x_{224}, x_{226}, x_{228}, x_{230}, x_{232},
x_{234}, x_{236}, x_{238}, x_{240}, x_{242}, x_{244}, x_{246}, x_{248}, x_{250}, x_{252}, x_{254},
x_{256}, x_{258}, x_{260}, x_{262}, x_{264}, x_{266}, x_{268}, x_{270}, x_{272}, x_{274}, x_{276},
x_{278}, x_{280}, x_{282}, x_{284}, x_{286}, x_{91}

However, this result is not unusual for Trivium attacks on the internal state, as in this case, all 288 variables of the state are unknown. In contrast, an algebraic or logical attack on the key can be considered only for round reduced Trivium [QW15]. The model generation is not practical for a full round Trivium encryption and consequently, the Trivium model is often viewed as a black-box polynomial.

To construct a more practical model for attacking Trivium, one should consider advanced cryptanalytic techniques for stream ciphers, such as Cube attacks [DS09] and Cube testers [ADMS09]. We explored the possibility of incorporating SAT solving techniques into Cube attacks, however, we did not find an impactful use of SAT solvers for these algorithms. Indeed, the bottleneck of Cube attacks is the preprocessing phase which consists in computing the sum of all cubes and deriving a polynomial system in the unknown variables from the key vector. The resulting systems are usually linear or quadratic and are highly sparse with few variables. Most of the time they are solvable by hand.

Trivium and other stream ciphers are still of great interest to the SAT community, as they are useful for generating SAT instances with nonrandom properties that can be used to assess different solving techniques.

8.2 The MQ problem

Public-key cryptographic systems that are currently used in practice are based on the hardness of integer factorization and discrete logarithm. For instance, the most eminent examples, RSA, ECDHE and ECDSA are implemented in the SSL/TLS protocol. The underlying problems of these encryption schemes have been researched for decades and thus, today, we have a solid understanding of their complexity. However, it is well known that these schemes are not resistant to quantum attacks. It was shown by Shor in 1999 [Sho97] that the integer factorization, as well as the discrete logarithm problem are solvable by a sufficiently large quantum computer in polynomial time. Consequently, in 2016, the NIST launched a standardization process for post-quantum cryptography. The new schemes that are required to be resistant to attacks on a quantum computer are divided into five families, namely hash-based, code-based, lattice-based, multivariate and supersingular elliptic-curve isogeny cryptography. In this work, we concentrate on the problem of solving a multivariate polynomial (MP) system which is at the core of

multivariate public-key cryptography. This problem is considered to be NP-hard.

There are several signature schemes proposed for the NIST standardization process that are based on the hardness of the MP problem. Rainbow [DS05], based on the UOV [KPG99] signature scheme, is part of the recently announced Round 3 finalists. The Lifted UOV scheme (LUOV) was a serious candidate in the second round, however, it was recently shown that it has some weaknesses against a subfield differential attack [DDS+20]. It is still important to consider schemes that are not in the third round as potential alternatives, in case all of the proposed schemes are proven to be insecure in the future. These primitives have been a target of cryptographic attacks since recently and their security is not explored as much as the security of elliptic-curve based systems, for instance. In this work, we do not concentrate on a particular scheme, but rather, we attack the underlying MP problem, for different parameter values. The crucial parameters in evaluating the hardness of a multivariate polynomial system are the number of variables, denoted by n, the number of equations, denoted by m and their ratio. The case of $m = n$ is considered to be the hardest, whereas overdetermined systems are easier to solve. Our complexity analysis and experimental results are in line with this well-known result.

8.2.1 Generation of the MQ model

An MQ system is a system of multivariate polynomial equations that consists only of quadratic polynomials. To use logical cryptanalysis, we will concentrate on systems over $\mathbb{F}_2$. Let $a_{i,j}^{(t)}$, $b_i^{(t)}$ and $c^{(t)}$ with $1 \leq i \leq j \leq n$ and $1 \leq t \leq m$, be elements in $\mathbb{F}_2$. An MQ polynomial system is described as follows:

$$
\begin{aligned}
f_1(x_1, \ldots, x_n) &= \sum_{1 \leq i \leq j \leq n} a_{i,j}^{(1)} x_i x_j + \sum_{1 \leq i \leq n} b_i^{(1)} x_i + c^{(1)} = 0 \\
f_2(x_1, \ldots, x_n) &= \sum_{1 \leq i \leq j \leq n} a_{i,j}^{(2)} x_i x_j + \sum_{1 \leq i \leq n} b_i^{(2)} x_i + c^{(2)} = 0 \\
&\ldots \\
f_m(x_1, \ldots, x_n) &= \sum_{1 \leq i \leq j \leq n} a_{i,j}^{(m)} x_i x_j + \sum_{1 \leq i \leq n} b_i^{(m)} x_i + c^{(m)} = 0.
\end{aligned}
$$

To construct such systems, we took as reference the MQ challenge initiated in [YDH+15]. Contrary to the case of the index calculus attack, where instances can be both satisfiable and unsatisfiable, the MQ instances that need to be solved in a practical attack always have a solution. Hence, we generate instances with (pseudo)random solutions, where all coefficients are random and have an equal probability to take the value of 0 or 1. The process of generating one random instance follows these steps:

- Fix parameters m and n.
- Choose randomly an n-bit solution vector.
- For each equation $1 \leq t \leq m$, choose randomly all coefficients except the $c^{(t)}$ constant, and then compute the $c^{(t)}$ constant according to the solution vector chosen in the previous step.

Considering these instances, the MQ problem is defined as follows

Definition 8.2.1 *Given m multivariate quadratic polynomials $f_1, \ldots, f_m$ of n variables over a finite field $\mathbb{F}$, find a tuple $\boldsymbol{w} = (w_1, \ldots, w_n)$ in $\mathbb{F}^n$, such that $f_1(\boldsymbol{w}) = \cdots = f_m(\boldsymbol{w}) = 0$.*

8.2.2 Complexity analysis

We experimented with the WDSat solver on these instances and the results are in Section 8.2.3. Before we show experimental results, we present a complexity analysis. The following analysis concerns the WDSat xg-ext variant, as this variant was found to be the most efficient for mq systems. Recall that, the complexity of solving a polynomial system using a sat-based approach is strongly linked to the number of conflicts. The earlier we discover the conflict, the faster we arrive at a solution. Hence, as an initial assessment of the complexity analysis, it is interesting to look at the dpll-tree level on which conflicts are found. Since the very first runs that we performed on the mq model, we noticed that conflicts are always found on the same two levels for a given set of parameters m and n. For instance, for $n = 30$ and $m = 60$ all conflicts are either on level 19 or on level 20. In other words, the model that performs ge on the system, always finds a solution or encounters a conflict after 19 or 20 variables are assigned. This has prompted us to think in terms of the Macaulay matrix [Mac16] associated to the polynomial system. It is well known that highly overdetermined systems are easy to solve. In a dpll-based setting, as we descend further in the binary tree, the number of (non-constant) variables decreases, yet the number of equations remains unchanged. As a result, the system becomes more and more overdetermined. We argue that the exact level at which the system is solved using ge, the result of which can be either a conflict or a found solution, is when the associated Macaulay matrix becomes a square matrix (or very close to one).

The *EC* structure in the xorgauss module of the WDSat solver can be viewed as an analog of the Macaulay matrix, with the exception that it does not contain the square terms, since in $\mathbb{F}_2$ we have that $x^2 = x$. The number of columns in the *EC* structure, for an mq system, supposing that all monomials have at least one occurrence, is $n(n+1)/2$. The assumption that all monomials have at least one occurrence holds for random systems generated as described in Section 8.2.1. Let n' be the number of remaining variables in the system. Then, at level h of the binary search tree, we have that $n' = n - h$. As per our analysis, the system is solved or a conflict is met when $m \approx n'(n'+1)/2$. In our experiments, a conflict is found either for the highest value of n' that satisfies $m \geq n'(n'+1)/2$ or the lowest value of n' where $m \leq n'(n'+1)/2$, with no exceptions.

Even though this analysis is strongly linked to the ge, it does not necessarily hold for other sat solvers that perform ge, such as CryptoMiniSat. If a solver does not apply the xg-ext technique, it can not be guaranteed that the number of remaining variables in the sat instance will be $n'(n'+1)/2$ on level h.

Using this analysis, we computed complexity estimations for various parameters and our findings are shown in Table 8.3. The complexity is exponential in the level at which conflicts are found, and thus we consider the complexity to be in the interval between the two levels that are closest to $m \approx n'(n'+1)/2$. As in most cases, one level is closer the approximation than the other, we compute a more precise estimate as well.

8.2.3 Experimental results and perspectives

As in previous chapters, all experiments were performed on a 2.40GHz Intel Xeon E5-2640 processor using 128 GB of RAM, and results are an average of 100 runs. Table 8.4 shows a comparison between different approaches for solving mq systems. First, we conclude that the best version of WDSat is the one with the xg-ext technique. Then, we can see that CryptoMiniSat gives better results when the ge is turned on, however, WDSat outperforms CryptoMiniSat for these instances. We did not run a Gröbner basis attack, instead we show results from [YDH+15] for reference. In [YDH+15], Magma's

Table 8.3: Complexity estimates of the MQ problem with different parameters.

n	$m = 2n$		$m = n$	
	Interval	Precise	Interval	Precise
30	$[2^{19}, 2^{20}]$	$2^{19.55}$	$[2^{22}, 2^{23}]$	$2^{22.75}$
35	$[2^{23}, 2^{24}]$	$2^{23.67}$	$[2^{27}, 2^{28}]$	$2^{27.13}$
40	$[2^{27}, 2^{28}]$	$2^{27.85}$	$[2^{31}, 2^{32}]$	$2^{31.56}$
128	$[2^{105}, 2^{106}]$	$2^{105.87}$	$[2^{112}, 2^{113}]$	$2^{112.5}$
145	$[2^{121}, 2^{122}]$	$2^{121.42}$	$[2^{128}, 2^{129}]$	$2^{128.48}$

implementation of F4 is used and instances are generated using the same approach as in our model generation. Thus, these results can serve as a rough comparison, even though they are executed on different platforms. Both experiment sets are performed in a uni-thread environment. This comparison shows that the Gröbner basis approach is far less efficient for these parameters than SAT solvers, which can be expected in the case of $\mathbb{F}_2$. The current record holder for this type of instances of the MQ challenge is the Joux-Vitse hybrid algorithm [JV17]. We did not implement this algorithm to have comparative experimental results, but according to running times reported in [Tak], our solver would be outperformed by this algorithm.

Table 8.4: Comparing different approaches for solving the MQ problem.

Solving approach	n	m	Runtime	#Conflicts
Gröbner [YDH+15]	25	50	658.157	NA
CMS	25	50	2598.660	9806242
CMS_{GE}	25	50	383.064	2007847
	30	60	28954.142	116013784
WDSat	25	50	57.851	14177200
	30	60	2774.440	483437900
WDSat_{GE}	25	50	23.772	1046328
	30	60	1223.163	34718415
$\text{WDSat}_{\text{XG-EXT}}$	25	50	**0.822**	**21140**
	30	60	**17.714**	**379346**

Choosing the WDSat XG-EXT variant as optimal, we continued with experiments for bigger parameters and results are shown in Table 8.5. It is clear that logical formulas derived from polynomial systems where $m = 2n$ are a lot easier to solve than those derived from systems where $m = n$. The $m = 2n$ parameter ratio is often used for encryption

schemes, such as ABC [TDTD13; DPW14], ZHFE [PBD14] and the QUAD [BGP06] stream cipher.

Table 8.5: WDSat xg-ext version solving the MQ problem.

n	m	Runtime	#Conflicts
30	60	17.714	379346
	30	78.034	2325179
35	70	605.837	7215819
	35	3375.770	47649168
40	80	14867.168	124127894
	40	105362.775	1059322736

To verify our complexity analysis in Section 8.2.2, we ran experiments on the same instances, but this time, we asked the solver to find all solutions instead of just the one. We took this approach because when the solver stops as soon as it finds a solution, running times and number of conflicts can vary between different instances. When we look for all solutions, we transverse the whole binary search tree, which is analogous to the UNSAT case. Hence, this approach allows us to have an impartial comparison to the estimated complexity, as the running time does not depend on the solution vector, which is an arbitrary value. The results of these experiments are in Table 8.6 and are an average of 100 runs. We observe that the number of conflicts is fairly close to the estimated complexity.

Table 8.6: WDSat xg-ext version solving the MQ problem.

n	m	Estimated complexity	Runtime	#Conflicts
30	60	$2^{19.55}$	38.748	809310
	30	$2^{22.75}$	245.308	7124700
35	70	$2^{23.67}$	1349.579	15413208
	35	$2^{27.13}$	9893.066	132561750
40	80	$2^{27.85}$	32734.160	265041578
	40	$2^{31.56}$	315769.284	3067821370

In these randomly generated polynomial systems, each monomial has a 1/2 probability to appear in each equation and thus has a very strong probability to have at least one occurrence in the system. As a result, when we use the MVC preprocessing technique, introduced in Chapter 7, the monomials connectivity graph of these polynomial systems is complete and using the MVC technique does not yield better running times. Recall that, when the graph is complete, the size of the MVC is equal to the number of variables. However, we are hopeful that by building on the MVC preprocessing technique we can develop improved attacks on MQ .

Open question 7 There is a potential for the MVC technique to be used in more elaborate ways for MQ attacks. Some ideas include computing the MVC dynamically instead of as preprocessing, and looking for the MVC of a subset of equations.

Open question 8 WDSAT can be used for attacks on specific MQ-based cryptosystems and coupled with more advanced MQ attacks.

Chapter 9

Conclusion

This thesis explores the use of combinatorial techniques, such as graph-based algorithms and constraint programming, in cryptanalysis. We present several instances where we discover a reduction from a problem in cryptography to a problem in combinatorics and then, we use techniques for solving one problem to solve the other. Most notably, since so many cryptographic problems deal with Boolean constraints, there is a growing number of cryptographic attacks that use SAT solvers. Usually, generic SAT solvers are used in these attacks. Using a state-of-the-art generic SAT solver has the advantage of having a highly optimized implementation that encompasses best solving techniques, and, as a result, these solvers can tackle huge instances in practice. However, we find that when one has a solid understanding of the problem and the resulting model, there are specific techniques that can be used to improve the resolution process. Motivated by the initial results of a solver-aided attack in [GG14], we decided to take this approach for the index calculus problem on curves over prime-degree binary extension fields. After developing a SAT solver dedicated to this specific problem, named WDSAT, we used the solver as a tool for attacks on other cryptosystems that yield algebraic and logical models with similar properties to the index calculus one. As our main focus is on the ECDLP, we also tackled this problem in the generic case. More specifically, we investigated the use of tree and graph structures in the PCS attack.

Our original contributions include the following.

- In our work on the PCS algorithm, we proposed to replace the classical hash table by a simple structure with lower memory requirements, that is inspired by radix trees.
- We provided a more refined analysis of the running time of a parallel collision search for finding multiple collisions, eliminating the heuristic in the analysis of van Oorschot and Wiener [van91].
- We proposed an original XOR-reasoning SAT solver, dedicated to the problem of solving the PDP in the index calculus attack.
- We developed a symmetry breaking technique for finding roots of Semaev's summation polynomials that, in contrast to other symmetry breaking techniques, is not applied to the modelization or the choice of a factor base, but to the solving process. As a result of the simple algorithmic adjustments that we proposed, our technique comes with no additional computational cost.
- Extending our solver with the XG-EXT module yields, to the best of our knowledge, a first SAT solver that fully exploits the power of GE. Specifically, we detected a

limitation of the GE technique in XOR-enabled SAT solvers that is not present in algebraic solving techniques. As we proposed a simple solution to fix this oversight, we are confident that future works will aim at making our solution more efficient at the implementation level, as well as proposing a similar repair for other XOR-enabled SAT solvers.

- We proposed a novel preprocessing technique, that uses the MVC problem to statically determine the order of branching variables. It is well-known that variables whose assignment brings the polynomial system closer to a linear system should be assigned first. In [SPF08], for instance, these are called "suitable" variables. However, we formalized this idea by equating the problem of finding the lowest number of variables that need to be assigned to obtain a linear system to the problem of finding a Minimal Vertex Cover in a graph.

The complexity analysis and experimental work of this thesis show several results of our contributions. Notably, our PRTL storage structure results in lower memory occupation for the PCS algorithm, both experimentally and asymptotically. In our work on solving the PDP for prime-degree extension fields in characteristic 2, we presented experiments using S_4' with 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 $l > 8$ have never been shown before for this case. Moreover, memory is no longer a constraint for the PDP when the Gröbner basis computation is replaced with SAT solving. Lastly, we show that the complexity of the PDP using the third summation polynomial is inherently different than that of higher summation polynomials. However, using this polynomial is considered impractical for the linear algebra step. Hence, it is unclear to us whether S_3 should be reconsidered for whole index calculus computations.

In terms of limitations, we ought to discuss our work on the cryptanalysis of the Trivium cipher. Our experimental results show that the solver which was already used for attacks on this cryptosystem, CryptoMiniSat, performs better than our solver for these instances. Furthermore, our attempts to incorporate SAT solving in Cube attacks were unsuccessful. On the other hand, WDSat performs better than CryptoMiniSat for instances arising from the MQ problem. The conclusion that we can draw from these experiences is that WDSat is more suitable for dense polynomial systems with fewer variables and longer XOR constraints, whereas CryptoMiniSat should be used for sparse polynomial systems. As CryptoMiniSat does not fully profit from GE, through a XG-EXT-like technique for instance, a merge of best practices between the two solvers might be of interest.

9.1 Open questions and extendibility

As we find that one of the most important contributions of a PhD thesis is to provide ideas for future work, we dedicate a separate section to this discussion. In the different chapters of this thesis, we introduced 8 clearly marked open questions and perspectives. Some of them are recalled here in more detail.

Complexity of CNF-XOR instance solving. Solving SAT instances derived from cryptographic problems involves two polynomial SAT solving algorithms: the HORNSAT algorithm for the CNF part, and the GE technique for the XOR part. The difficulty comes from the requirement to find a solution in the intersection of the sets of solutions for these two problems, which raises the fundamental question of how two polynomial SAT problems can be combined and what would be the resulting complexity of such an algorithm.

WDSat and CDCL. WDSat does not implement clause learning at this time. We have shown through several core examples that classical CDCL techniques do not improve the number of conflicts for solving instances derived from multivariate polynomial systems. However, we have not explored the possibility of using clauses learned from inconsistencies in XOR constraints. It may be of interest to use CDCL techniques coupled with the WDSat solver and perform GE when the system becomes linear.

XOR reasoning with incomplete SAT solving techniques. This thesis discusses only complete SAT solving approaches. Another perspective would be to use incomplete SAT solvers, such as WalkSat, coupled with an XOR reasoning module.

Understand the ml/n ratio for a whole index calculus computation. For the case of elliptic curves over $\mathbb{F}_{2^n}$, with n prime, it is clear that the PDP computation is the bottleneck of the index calculus attack. The complexity of all existing approaches for solving the PDP is strongly linked to the choice of the l parameter. Besides, the number of relations that need to be found is exponential in l. This has prompted us to reason that, apart from coming up with better algorithms for solving the PDP, the overall complexity of the index calculus attack can only be improved by determining an optimal choice for the value of l, with respect to parameters n and m.

Develop more elaborate techniques for using the MVC for solving MQ systems. Random MQ instances usually have complete monomials connectivity graphs. As a result, the MVC technique can not compute an optimal order of branching variables when it is used as preprocessing. A natural question that arises is whether computing the MVC dynamically, i.e. on each, or some levels of the binary search tree, would yield better results. Recall that, even though the MVC is an NP-complete problem, MVC instances arising from the problem of finding an optimal branching order are practical. In addition, one can consider looking for the MVC of a subset of equations or more specifically, of any subset of polynomials in the vector space generated by polynomials in the initial system.

Use WDSat for attacks on other cryptosystems. After the success of WDSat with instances derived from a Weil descent, and instances of the MQ problem, the solver can be tested against other problems for which Gröbner basis approaches are used in practice. Such cryptosystems include, for instance, HFE [Pat96a], the code-based McEliece [McE78], the UOV-based [KPG99] signatures Rainbow [DS05] and LUOV [BP17] and the MQ-based encryption schemes ABC [TDTD13; DPW14], ZHFE [PBD14] and QUAD [BGP06].

In addition to introducing these open questions, we hope that the material presented in this thesis will encourage the cryptographic community to use SAT solvers extensively for cryptanalysis over $\mathbb{F}_2$. In larger fields, other constraint programming techniques can be considered.

Bibliography

[ACC+18] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 322–343. Springer, 2018.

[ADH94] Leonard M. Adleman, Jonathan DeMarrais, and Ming-Deh Huang. A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields. In Leonard M. Adleman and Ming-Deh Huang, editors, *Algorithmic Number Theory*, pages 28–40, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round md6 and trivium. In Orr Dunkelman, editor, *Fast Software Encryption*, pages 1–22, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[AFI+04] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison between XL and gröbner basis algorithms. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 2004.

[AFK87] Martin Abadi, Joan Feigenbaum, and Joe Kilian. On hiding information from an oracle. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 195–203, New York, NY, USA, 1987. Association for Computing Machinery.

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 399–404, 2009.

[Bar04] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Pierre and Marie Curie University, Paris, France, 2004.

[BBB+09] Daniel V. Bailey, Lejla Batina, Daniel J. Bernstein, Peter Birkner, Joppe W. Bos, Hsieh-Chung Chen, Chen-Mou Cheng, Gauthier van Damme, Giacomo

de Meulenaer, Luis Julian Dominguez Perez, Junfeng Fan, Tim Güneysu, Frank Gurkaynak, Thorsten Kleinjung, Tanja Lange, Nele Mentens, Ruben Niederhagen, Christof Paar, Francesco Regazzoni, Peter Schwabe, Leif Uhsadel, Anthony Van Herrewege, and Bo-Yin Yang. Breaking ecc2k-130. Cryptology ePrint Archive, Report 2009/541, 2009. https://eprint.iacr.org/2009/541.

[BBB+20] Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich. An algebraic attack on rank metric code-based cryptosystems. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 64–93, Cham, 2020. Springer International Publishing.

[BCC+10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 203–218, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BCC+13] Charles Bouillaguet, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang. Fast exhaustive search for quadratic systems in $\mathbb{F}_2$ on FPGAs. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2013.

[BCLA83] Bruno Buchberger, George E. Collins, Rudiger Loos, and Rudolf Albrecht, editors. *Computer Algebra: Symbolic and Algebraic Computation (2nd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1983.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997.

[BEL+] Daniel J. Bernstein, Susanne Engels, Tanja Lange, Ruben Niederhagen, Christof Paar, Peter Schwabe, and Ralf Zimmermann. Faster elliptic-curve discrete logarithms on FPGAs. https://eprint.iacr.org/2016/382.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, page 213–229, Berlin, Heidelberg, 2001. Springer-Verlag.

[BFP09] 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.

[BFP12] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Solving polynomial systems over finite fields: Improved analysis of the hybrid approach. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, ISSAC '12, page 67–74, New York, NY, USA, 2012. Association for Computing Machinery.

[BFS15] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the F5 gröbner basis algorithm. *J. Symb. Comput.*, 70:49–70, 2015.

[BFSS13] Magali Bardet, Jean-Charles FaugèRe, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic boolean systems. *J. Complex.*, 29(1):53–75, February 2013.

[BFSY05] Magali. Bardet, Jean-Charles Faugère, Bruno Salvy, and Bo-Yin Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *IN MEGA'05, 2005. EIGHTH INTERNATIONAL SYMPOSIUM ON EFFECTIVE METHODS IN ALGEBRAIC GEOMETRY*, 2005.

[BGP06] Côme Berbain, Henri Gilbert, and Jacques Patarin. Quad: A practical stream cipher with provable security. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 109–128, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BHvMW09] 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.

[Bib07] Wolfgang Bibel. Early history and perspectives of automated deduction. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *KI 2007: Advances in Artificial Intelligence*, pages 2–18, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Bie] Armin Biere. CaDiCaL Simplified Satisfiability Solver. http://fmv.jku.at/cadical/. Accessed: 2020-05-27.

[BKK+12] Joppe W. Bos, Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra, and Peter L. Montgomery. Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. *Int. J. Appl. Cryptogr.*, 2(3):212–228, 2012.

[BLS11] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. On the Correct Use of the Negation Map in the Pollard rho Method. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Conference on Theory and Practice of Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 128–146, Taormina, Italy, March 6–9, 2011. Springer, Heidelberg, Germany.

[Blu18] BlueKrypt. Cryptographic key length recommendation. https://www.keylength.com, 2018. Accessed: 2020-07-08.

[BM00] Peter Baumgartner and Fabio Massacci. The taming of the (x)or. In John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Computational Logic — CL 2000*, pages 508–522, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Bol01] Béla Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 2001.

[Bou16] Charles Bouillaguet. LibFES-lite. https://github.com/cbouilla/libfes-lite, 2016.

[BP17] Ward Beullens and Bart Preneel. Field lifting for smaller uov public keys. Cryptology ePrint Archive, Report 2017/776, 2017. https://eprint.iacr.org/2017/776.

[Bre80] Richard P. Brent. An improved Monte Carlo factorization algorithm. *BIT*, 20:176–184, 1980.

[BS96] Max Böhm and Ewald Speckenmeyer. A fast parallel sat-solver - efficient workload balancing. *Ann. Math. Artif. Intell.*, 17(3-4):381–400, 1996.

[Buc79] Bruno Buchberger. A criterion for detecting unnecessary reductions in the construction of gröbner-bases. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, pages 3–21, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

[Buc85] Bruno Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. *Multidimensional Systems Theory*, pages 184–232, 01 1985.

[Buc06] Bruno Buchberger. Bruno buchberger's phd thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *J. Symb. Comput.*, 41(3–4):475–511, March 2006.

[Che09] Jingchao Chen. Building a hybrid sat solver via conflict-driven, look-ahead and xor reasoning techniques. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, pages 298–311, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 392–407, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[CL05] Carlos Cid and Gaëtan Leurent. An analysis of the xsl algorithm. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, pages 333–352, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[CLO07] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2007.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '02, page 267–287, Berlin, Heidelberg, 2002. Springer-Verlag.

[CSCM19] Davin Choo, Mate Soos, Kian Ming Adam Chai, and Kuldeep S. Meel. Bosphorus: Bridging ANF and CNF solvers. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, pages 468–473, 2019.

[DC06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security*, pages 171–186, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[DD04] Gilles Dequen and Olivier Dubois. kcnfs: An efficient solver for random k-sat formulae. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, pages 486–501, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[DDS+20] Jintai Ding, Joshua Deaton, Kurt Schmidt, Vishakha, and Zheng Zhang. Cryptanalysis of the lifted unbalanced oil vinegar signature scheme. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 279–298, Cham, 2020. Springer International Publishing.

[DG84] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976. URL: `http://cr.yp.to/bib/entries.html#1976/diffie`.

[Die03] Claus Diem. The ghs attack in odd characteristic, 2003.

[Die04] Claus Diem. The xl-algorithm and a conjecture from commutative algebra. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, pages 323–337, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Die11] Claus Diem. On the discrete logarithm problem in elliptic curves. *Compositio Mathematica*, 147(1):75–104, 2011.

[Die13] Claus Diem. On the discrete logarithm problem in elliptic curves II. *Algebra & Number Theory*, 7(6):1281–1323, 2013.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[DPW14] Jintai Ding, Albrecht Petzoldt, and Lih-chung Wang. The cubic simple matrix encryption scheme. In Michele Mosca, editor, *Post-Quantum Cryptography*, pages 76–87, Cham, 2014. Springer International Publishing.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 164–175, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 278–299, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[ES04] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[eST] ECRYPT Stream Cipher Project. https://www.ecrypt.eu.org/stream/. Accessed: 2020-11-06.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner basis (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.

[Fau02] 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.

[Fau10] Jean-Charles Faugère. FGb: A Library for Computing Gröbner Bases. In Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 84–87, Berlin, Heidelberg, September 2010. Springer Berlin / Heidelberg.

[FGLM93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *J. Symb. Comput.*, 16(4):329–344, 1993.

[FHJ+14] Jean-Charles Faugère, Louise Huot, Antoine Joux, Guénaël Renault, and Vanessa Vitse. Symmetrized summation polynomials: Using small order torsion points to speed up elliptic curve index calculus. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 40–57. Springer, 2014.

[FJ03] Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of hidden field equation (hfe) cryptosystems using gröbner bases. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[FJM14a] Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to discrete logarithm, even-mansour and prince. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 420–438, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[FJM14b] Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to discrete logarithm, Even-Mansour and PRINCE. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 420–438, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[FPPR12] Jean-Charles Faugère, Ludovic Perret, Christophe Petit, and Guénaël Renault. Improving the Complexity of Index Calculus Algorithms in Elliptic Curves over Binary Fields. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - Eurocrypt 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 27–44, 2012.

[FR94] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comput.*, 62(206):865–874, April 1994.

[Fre60] Edward Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, September 1960.

[Fre95] Jon William Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, USA, 1995.

[Fre98] Gerhard Frey. How to disguise an elliptic curve (weil descent). Talk at the 2nd Elliptic Curve Cryptography Workshop (ECC), 1998.

[Gau00] Pierrick Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 19–34, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Gau09] 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.

[GG14] 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.

[GHS02] Pierrick Gaudry, Florian Hess, and Nigel P. Smart. Constructive and destructive facets of weil descent on elliptic curves. *J. Cryptology*, 15:19–46, 2002.

[GLMS18] 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.

[GLMS20] David Gerault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Computing AES related-key differential characteristics with constraint programming. *Artif. Intell.*, 278, 2020.

[GLS84] Martin Grötschel, László Lovász, and Alexander Schrijver. Polynomial algorithms for perfect graphs. *North-Holland Mathematics Studies*, 88(C):325–356, January 1984.

[GMS17] David Gerault, Marine Minier, and Christine Solnon. Using constraint programming to solve a cryptanalytic problem. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4844–4848. ijcai.org, 2017.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[GTTD07] Pierrick Gaudry, Emmanuel Thomé, Nicolas Thériault, and Claus Diem. A double large prime variation for small genus hyperelliptic index calculus. *Math. Comput.*, 76(257):475–492, 2007.

[GWC86] Carl Friedrich Gauss, William C. Waterhouse, and Arthur A. Clarke. *Disquisitiones Arithmeticae*. Springer-Verlag, 1986.

[Har77] Robin Hartshorne. *Algebraic Geometry*, volume 52 of *Graduate Texts in Mathematics*. Springer, 1977.

[HDvZvM04] Marijn Heule, Mark Dufour, Joris van Zwieten, and Hans van Maaren. March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2004.

[HGSW03] Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. A meet-in-the-middle attack on an NTRU private key. Tehnical report, NTRU Cryptosystems, 2003.

[HJ12a] 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.

[HJ12b] 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.

[Hor51] Alfred Horn. On sentences which are true of direct unions of algebras. *J. Symbolic Logic*, 16(1):14–21, 03 1951.

[Iso11] Takanori Isobe. A single-key attack on the full GOST block cipher. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305, Lyngby, Denmark, February 13–16, 2011. Springer, Heidelberg, Germany.

[Jac39] Carl G. J. Jacobi. *Canon arithmeticus*. Typis academicis, 1839.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie–hellman. In Wieb Bosma, editor, *Algorithmic Number Theory*, pages 385–393, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis*, chapter 7, pages 225–226. Chapman & Hall/CRC, 2009.

[JV12] Antoine Joux and Vanessa Vitse. Cover and Decomposition Index Calculus on Elliptic Curves made practical. Application to a seemingly secure curve over $\mathbb{F}_{p^6}$. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - Eurocrypt 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 7237, pages 9–26. Springer, 2012.

[JV13] Antoine Joux and Vanessa Vitse. Elliptic curve discrete logarithm problem over small degree extension fields - application to the static diffie-hellman problem on $e(\mathbb{F}_{q^5})$. *J. Cryptol.*, 26(1):119–143, 2013.

[JV17] Antoine Joux and Vanessa Vitse. A crossbred algorithm for solving boolean polynomial systems. In Jerzy Kaczorowski, Josef Pieprzyk, and Jacek Pomykala, editors, *Number-Theoretic Methods in Cryptology - First International Conference, NuTMiC 2017, Warsaw, Poland, September 11-13, 2017, Revised Selected Papers*, volume 10737 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2017.

[JWB09] Peter L. Montgomery Joppe W. Bos, Marcelo E. Kaihara. Pollard rho on the playstation 3. Workshop record of SHARCS'09 http://www.hyperelliptic.org/tanja/SHARCS/record2.pdf, 2009.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[KNW09] Dmitry Khovratovich, Ivica Nikolic, and Ralf-Philipp Weinmann. Meet-in-the-middle attacks on SHA-3 candidates. In Orr Dunkelman, editor, *Fast Software Encryption – FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 228–245, Leuven, Belgium, February 22–25, 2009. Springer, Heidelberg, Germany.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[Kob92a] 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.

[Kob92b] 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.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 206–222, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Kra26] Maurice Kraitchik. *Théorie des nombres. Tome II. Analyse indéterminée du second degré et factorisation.* Paris: Gauthier-Villars, 1926.

[KS99] Aviad Kipnis and Adi Shamir. Cryptanalysis of the hfe public key cryptosystem by relinearization. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 19–30, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[KS01] Fabian Kuhn and René Struik. Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, pages 212–229, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Laz83] 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.

[LCM+17] Fanghui Liu, Waldemar Cruz, Chujiao Ma, Greg Johnson, and Laurent Michel. A tolerant algebraic side-channel attack on aes using cp. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming*, pages 189–205, Cham, 2017. Springer International Publishing.

[LCM18] Fanghui Liu, Waldemar Cruz, and Laurent Michel. A complete tolerant algebraic side-channel attack for aes with cp. In John Hooker, editor, *Principles and Practice of Constraint Programming*, pages 259–275, Cham, 2018. Springer International Publishing.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3), 1973.

[Li99] Chu Min Li. A constraint-based approach to narrow search trees for satisfiability. *Information Processing Letters*, 71(2):75 – 80, 1999.

[Li00] Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA*, pages 291–296. AAAI Press / The MIT Press, 2000.

[LJMP93] Arjen K. Lenstra, Hendrik W. Lenstra Jr., Mark S. Manasse, and John M. Pollard. *The Number Field Sieve*, pages 11–42. Springer Berlin Heidelberg, 1993.

[LJN11] Tero Laitinen, Tommi A. Junttila, and Ilkka Niemelä. Equivalence Class Based Parity Reasoning with DPLL(XOR). In *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pages 649–658, Nov 2011.

[LJN14] Tero Laitinen, Tommi A. Junttila, and Ilkka Niemelä. Conflict-Driven XOR-Clause Learning (extended version). *CoRR*, abs/1407.6571, 2014.

[LM91] Arjen K. Lenstra and Mark S. Manasse. Factoring with two large primes (extended abstract). In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, EUROCRYPT '90, page 72–82, Berlin, Heidelberg, 1991. Springer-Verlag.

[LMPP18] 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.

[Mac02] Francis S. MacAulay. Some Formulæ in Elimination. *Proceedings of the London Mathematical Society*, s1-35(1):3–27, 05 1902.

[Mac16] Francis S. Macaulay. *The Algebraic Theory of Modular Systems*. Cambridge tracts in mathematics and mathematical physics. University Press, 1916.

[McE78] Robert J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[MCP07] Cameron McDonald, Chris Charnes, and Josef Pieprzyk. An algebraic analysis of trivium ciphers based on the boolean satisfiability problem. *IACR Cryptol. ePrint Arch.*, 2007:129, 2007.

[MH06] Ralph Merkle and Martin Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inf. Theor.*, 24(5):525–530, September 2006.

[Mil85] Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology*, CRYPTO '85, page 417–426, Berlin, Heidelberg, 1985. Springer-Verlag.

[MM00] Fabio Massacci and Laura Marraro. Logical Cryptanalysis as a SAT Problem. *J. Autom. Reasoning*, 24(1/2):165–203, 2000.

[MM11] Martin Maechler and Maintainer Martin Maechler. GNU Multiple Precision Arithmetic Library. https://gmplib.org/, 2011.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 530–535, 2001.

[MOV93] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Inf. Theory*, 39(5):1639–1646, 1993.

[MRST09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *Fast Software Encryption – FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276, Leuven, Belgium, February 22–25, 2009. Springer, Heidelberg, Germany.

[MSS96] João P. Marques-Silva and Karem A. Sakallah. Conflict Analysis in Search Algorithms for Satisfiability. In *ICTAI*, pages 467–469. IEEE Computer Society, 1996.

[MSS97] João P. Marques-Silva and Karem A. Sakallah. Grasp—a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '96, page 220–227, USA, 1997. IEEE Computer Society.

[Nat] National Institute of Standards and Technology. NIST Post-Quantum Cryptography Standardization. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization. Accessed: 2020-07-23.

[NIS92] CORPORATE NIST. The digital signature standard. *Commun. ACM*, 35(7):36–40, July 1992.

[NNY18] Ruben Niederhagen, Kai-Chun Ning, and Bo-Yin Yang. Implementing joux-vitse's crossbred algorithm for solving MQ systems over GF(2) on gpus. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 121–141, Cham, 2018. Springer International Publishing.

[NR18] Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 111–121. Springer, 2018.

[OPE] Open Multi-Processing Specification for Parallel Programming. https://gmplib.org/.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[Pat96a] 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.

[Pat96b] Patarin Jacques. HFE first challenge. http://www.minrank.org/challenge1.txt, 1996. Accessed: 2020-11-15.

[PBD14] Jaiberth Porras, John Baena, and Jintai Ding. Zhfe, a new multivariate public key encryption scheme. In Michele Mosca, editor, *Post-Quantum Cryptography*, pages 229–245, Cham, 2014. Springer International Publishing.

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[Pol78] John M. Pollard. Monte carlo methods for index computation ((mod p)). *Mathematics of Computation*, 32(143):918–924, 1978.

[Pom82] Carl Pomerance. Analysis and comparison of some integer factoring algorithms. In *Computational methods in number theory, Part I*, volume 154 of *Math. Centre Tracts*, page 89–139. Math. Centrum, Amsterdam, 1982.

[Pom87] Carl Pomerance. Fast, rigorous factorization and discrete logarithm algorithms. In David S. Johnson, Takao Nishizeki, Akihiro Nozaki, and Herbert S. Wilf, editors, *Discrete Algorithms and Complexity*, pages 119 – 143. Academic Press, 1987.

[PQ12] 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.

[QW15] Frank-M. Quedenfeld and Christopher Wolf. Advanced algebraic attack on trivium. In Ilias S. Kotsireas, Siegfried M. Rump, and Chee K. Yap, editors, *Mathematical Aspects of Computer and Information Sciences - 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers*, volume 9582 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 2015.

[Rad06] Håvard Raddum. Cryptanalytic results on TRIVIUM, 2006.

[RS06] Håvard Raddum and Igor A. Semaev. New technique for solving sparse equation systems. *IACR Cryptol. ePrint Arch.*, 2006:475, 2006.

[RS07] Håvard Raddum and Igor A. Semaev. Solving MRHS linear equations. *IACR Cryptol. ePrint Arch.*, 2007:285, 2007.

[S+] William A. Stein et al. *Sage Mathematics Software*. The Sage Development Team. `http://www.sagemath.org`.

[Sch90] Claus P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[SE05] Niklas Sörensson and Niklas Eén. A SAT Solver with Conflict-Clause Minimization. *Proc. Theory and Applications of Satisfiability Testing*, 2005.

[Sem04] Igor A. Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. *IACR Cryptology ePrint Archive*, 2004:31, 2004.

[Sem08] Igor A. Semaev. On solving sparse algebraic equations over finite fields. *Des. Codes Cryptogr.*, 49(1-3):47–60, 2008.

[Sem16] Igor Semaev. Maxminmax problem and sparse equations over finite fields. *Des. Codes Cryptography*, 79(2):383–404, May 2016.

[Sha83] Adi Shamir. A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 279–288, Boston, MA, 1983. Springer US.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[Sil86] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 1986.

[SKC93] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–531. DIMACS/AMS, 1993.

[SLM92] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, page 440–446. AAAI Press, 1992.

[SNC09] 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.

[Soo10a] Mate Soos. Enhanced Gaussian elimination in DPLL-based SAT solvers. In *In Pragmatics of SAT*, 2010.

[Soo10b] Mate Soos. Grain of Salt — an Automated Way to Test Stream Ciphers through SAT Solvers. In *Tools'10: the Workshop on Tools for Cryptanalysis 2010*, pages 131–144, London, United Kingdom, June 2010.

[Spa12] Pierre-Jean Spaenlehauer. *Solving multi-homogeneous and determinantal systems: algorithms, complexity, applications. (Résolution de systèmes multi-homogènes et déterminantiels : algorithmes, complexité, applications)*. PhD thesis, Pierre and Marie Curie University, Paris, France, 2012.

[SPF08] Ilaria Simonetti, Ludovic Perret, and Jean Charles FaugÃšre. Algebraic attack against Trivium. In *First International Conference on Symbolic Computation and Cryptography*, pages 95–102, 2008.

[ST13] 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.

[Sta10] Juraj Stacho. 3-colouring at-free graphs in polynomial time. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation*, pages 144–155, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Ste04] Steel Allan. Allan Steel's Gröbner Basis Timings Page. http://magma.maths.usyd.edu.au/users/allan/gb/, 2004. Accessed: 2020-11-15.

[Sva14] Jules Svartz. *Solving zero-dimensional structured polynomial systems. (Résolution de systèmes polynomiaux structurés de dimension zéro)*. PhD thesis, Pierre and Marie Curie University, Paris, France, 2014.

[SW16] Peter Schwabe and Bas Westerbaan. Solving binary *MQ* with grover's algorithm. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 303–322. Springer, 2016.

[Tak] Takanori Yasuda and Xavier Dahan and Yun-Ju Huang and Tsuyoshi Takagi and Kouichi Sakurai. Fukuoka MQ Challenge. https://www.mqchallenge.org/. Accessed: 2020-11-15.

[TDTD13] Chengdong Tao, Adama Diene, Shaohua Tang, and Jintai Ding. Simple matrix scheme for encryption. In Philippe Gaborit, editor, *Post-Quantum Cryptography*, pages 231–242, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Tes01] Edlyn Teske. On random walks for Pollard's rho method. *Math. Comp.*, 70(234):809–825, 2001.

[TID17] Monika Trimoska, Sorina Ionica, and Gilles Dequen. Time-memory tradeoffs for parallel collision search algorithms. Cryptology ePrint Archive, Report 2017/581, 2017. https://eprint.iacr.org/2017/581.

[TID19] Monika Trimoska, Sorina Ionica, and Gilles Dequen. Parallel Collision Search Implementation. https://github.com/mtrimoska/PCS, 2019.

[TID20a] Monika Trimoska, Sorina Ionica, and Gilles Dequen. EC Index Calculus Benchmarks. https://github.com/mtrimoska/EC-Index-Calculus-Benchmarks, 2020.

[TID20b] Monika Trimoska, Sorina Ionica, and Gilles Dequen. Parity (xor) reasoning for the index calculus attack. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming*, pages 774–790, Cham, 2020. Springer International Publishing.

[TID20c] Monika Trimoska, Sorina Ionica, and Gilles Dequen. A sat-based approach for index calculus on binary elliptic curves. In Abderrahmane Nitaj and Amr Youssef, editors, *Progress in Cryptology - AFRICACRYPT 2020*, pages 214–235, Cham, 2020. Springer International Publishing.

[TID20d] Monika Trimoska, Sorina Ionica, and Gilles Dequen. WDSat Solver. https://https://github.com/mtrimoska/WDSat, 2020.

[van91] Paul C. van Oorschot. A comparison of practical public key cryptosystems based on integer factorization and discrete logarithms. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 576–581, Santa Barbara, CA, USA, August 11–15, 1991. Springer, Heidelberg, Germany.

[Vit11] Vanessa Vitse. *Attaques algébriques du problème du logarithme discret sur courbes elliptiques. (Algebraic attacks on the elliptic curve discrete logarithm problem).* PhD thesis, Versailles Saint-Quentin-en-Yvelines University, France, 2011.

[vMF] Hans van Maaren and John Franco. The International SAT Competition Web Page. http://www.satcompetition.org/. Accessed: 2020-05-27.

[vOW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

[vV16] Christine van Vredendaal. Reduced memory meet-in-the-middle attack against the NTRU private key. *LMS Journal of Computation and Mathematics*, 19(Issue A (Algorithmic Number Theory Symposium XII)):43–57, 2016.

[vW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[YC04] Bo-Yin Yang and Jiun-Ming Chen. Theoretical analysis of xl over small fields. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *Information Security and Privacy*, pages 277–288, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[YC05] Bo-Yin Yang and Jiun-Ming Chen. All in the xl family: Theory and practice. In Choon-sik Park and Seongtaek Chee, editors, *Information Security and Cryptology – ICISC 2004*, pages 67–86, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[YCC04] Bo-Yin Yang, Jiun-Ming Chen, and Nicolas T. Courtois. On asymptotic security estimates in xl and gröbner bases-related algebraic cryptanalysis. In Javier Lopez, Sihan Qing, and Eiji Okamoto, editors, *Information and Communications Security*, pages 401–413, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[YCY13] Jenny Yuan-Chun Yeh, Chen-Mou Cheng, and Bo-Yin Yang. *Operating degrees for XL vs. F_4/F_5 for generic mathcalMQ with number of equations linear in that of variables.* Berlin: Springer, 0 edition, 2013.

[YDH+15] Takanori Yasuda, Xavier Dahan, Yun-Ju Huang, Tsuyoshi Takagi, and Kouichi Sakurai. MQ challenge: Hardness evaluation of solving multivariate quadratic problems. *IACR Cryptol. ePrint Arch.*, 2015:275, 2015.

[YJPST13] 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.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '01, page 279–285. IEEE Press, 2001.

Résumé

Les attaques cryptographiques que nous décrivons dans cette thèse reposent sur des approches combinatoires, relevant notamment de la théorie des graphes et de la satisfaction sous contraintes. Notre objectif principal concerne l'étude du problème du logarithme discret sur courbes elliptiques. Dans un premier temps, nous nous concentrons sur l'attaque de calcul d'index pour le cas des courbes elliptiques définies sur des extensions de corps finis de degré premier. Ainsi, la première phase du calcul d'index, phase de recherche de relations, consiste à résoudre des systèmes d'équations obtenus à partir de polynômes de Semaev, dont les zéros représentent des coordonnées de points. La résolution de ces systèmes répond au problème de décomposition de points. Dans le cadre de cette attaque, premièrement, nous modélisons le problème de décomposition de points sous la forme d'une formule logique et nous le définissons comme une instance du problème SAT. En ajout de cela, nous développons un solveur SAT dédié à ce problème spécifique, nommé WDSat. Le solveur est muni d'une extension qui vise à éliminer les solutions symétriques des polynômes de Semaev sans agrandir le modèle SAT et sans introduire de coût de calcul supplémentaire. Les temps d'exécution expérimentaux montrent que notre approche de résolution utilisant WDSat est significativement plus rapide que les méthodes algébriques actuelles basées sur le calcul de bases de Gröbner. De plus, notre solveur a des meilleures performances que d'autres solveurs SAT couramment utilisés, pour ce problème spécifique. Au final, nous abordons le problème du logarithme discret sur courbes elliptiques dans le cas générique. Notamment, pour la mise en oeuvre de l'attaque de recherche de collisions en contexte parallèle de van Oorschot et Wiener, nous proposons une nouvelle structure de données, ayant des conséquences importantes sur la complexité en mémoire et en temps.

Abstract

In this thesis, we explore the use of combinatorial techniques, such as graph-based algorithms and constraint satisfaction, in cryptanalysis. Our main focus is on the elliptic curve discrete logarithm problem. First, we tackle this problem in the case of elliptic curves defined over prime-degree binary extension fields, using the index calculus attack. A crucial step of this attack is solving the point decomposition problem, which consists in finding zeros of Semaev's summation polynomials and can be reduced to the problem of solving a multivariate Boolean polynomial system. To this end, we encode the point decomposition problem as a logical formula and define it as an instance of the SAT problem. Then, we propose an original XOR-reasoning SAT solver, named WDSat, dedicated to this specific problem. As Semaev's polynomials are symmetric, we extend the WDSat solver by adding a novel symmetry breaking technique that, in contrast to other symmetry breaking techniques, is not applied to the modelization or the choice of a factor base, but to the solving process. Experimental running times show that our SAT-based solving approach is significantly faster than current algebraic methods based on Gröbner basis computation. In addition, our solver outperforms other state-of-the-art SAT solvers, for this specific problem. Finally, we study the elliptic curve discrete logarithm problem in the general case. More specifically, we propose a new data structure for the Parallel Collision Search attack proposed by van Oorschot and Wiener, which has significant consequences on the memory and time complexity of this algorithm.