LPN in Cryptography: an Algorithmic Study

THÈSE N^{O} 7800 (2017)

PRÉSENTÉE LE 23 JUIN 2017

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE SÉCURITÉ ET DE CRYPTOGRAPHIE
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Sonia Mihaela BOGOS

acceptée sur proposition du jury:

Prof. O. N. A. Svensson, président du jury
Prof. S. Vaudenay, directeur de thèse
Prof. W. Meier, rapporteur
Prof. P.-A. Fouque, rapporteur
Prof. A. Lenstra, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2017

To my family and to my friends

Abstract

The security of public-key cryptography relies on well-studied *hard problems*, problems for which we do not have efficient algorithms. *Factorization* and *discrete logarithm* are the two most known and used hard problems. Unfortunately, they can be easily solved on a quantum computer by Shor's algorithm. Also, the research area of cryptography demands for crypto-diversity which says that we should offer a range of hard problems for public-key cryptography. If one hard problem proves to be easy, we should be able to provide alternative solutions. Some of the candidates for post-quantum hard problems, i.e. problems which are believed to be hard even on a quantum computer, are the *Learning Parity with Noise* (LPN), the Learning with Errors (LWE) and the Shortest Vector Problem (SVP). A thorough study of these problems is needed in order to assess their hardness. In this thesis we focus on the algorithmic study of LPN.

LPN is a hard problem that is attractive, as it is believed to be post-quantum resistant and suitable for lightweight devices. In practice, it has been employed in several encryption schemes and authentication protocols.

At the beginning of this thesis, we take a look at the existing LPN solving algorithms. We provide the theoretical analysis that assesses their complexity. We compare the theoretical results with practice by implementing these algorithms. We study the efficiency of all LPN solving algorithms which allow us to provide secure parameters that can be used in practice.

We push further the state of the art by improving the existing algorithms with the help of two new frameworks. In the first framework, we split an LPN solving algorithm into atomic steps. We study their complexity, how they impact the other steps and we construct an algorithm that optimises their use. Given an LPN instance that is characterized by the noise level and the secret size, our algorithm provides the steps to follow in order to solve the instance with optimal complexity. In this way, we can assess if an LPN instance provides the security we require and we show what are the secure instances for the applications that rely on LPN.

The second framework handles problems that can be decomposed into steps of equal complexity. Here, we assume that we have an adversary that has access to a finite or infinite number of instances of the same problem. The goal of the adversary is to succeed in just one instance as soon as possible. Our framework provides the strategy that achieves this. We characterize an LPN solving algorithm in this framework and show that we can improve its complexity in the scenario where the adversary is restricted. We show that other problems, like password guessing, can be modelled in the same framework.

Keywords: Learning Parity with Noise, LPN, algorithmic study, cryptography, Walsh Hadamard transform, BKW, LF1, LF2, covering codes, Gaussian elimination, post-quantum cryptography, public-key cryptography, password quessing

Résumé

La sécurité de la cryptographie à clé publique est basée sur des problèmes mathématiques supposés difficiles à résoudre. La factorisation et le logarithme discret sont deux problèmes de ce type. Ce sont les plus connus et les plus utilisés. Malheureusement, ils deviennent faciles à résoudre avec l'algorithme de Shor sur un ordinateur quantique. La recherche en cryptographie exige également de la crypto-diversité. Cette dernière demande que l'on offre une variété de problèmes difficiles à résoudre en cryptographie à clé publique. Si un problème se révèle facile à résoudre, nous devons être capable de proposer des solutions alternatives. Le problème *Learning Parity with Noise* (LPN), le problème *Learning with Errors* (LWE) et le problème Shortest Vector Problem (SVP) sont quelques candidats de problèmes mathématiques difficiles à résoudre sur un ordinateur quantique. Une étude approfondie de ces problèmes est nécessaire afin d'évaluer leur sécurité. Dans cette thèse, nous effectuons une étude algorithmique approfondie de LPN.

LPN est un problème qui est intéressant, car on le suppose difficile à résoudre sur un ordinateur quantique. Il est aussi adapté pour l'utilisation de la cryptogrpahie sur des appareils bas-coût. En pratique, ce problème est utilisé dans plusieurs systèmes de chiffrement et protocoles d'authentification.

Au début de cette thèse, nous étudions les algorithmes existants qui résolvent LPN. Nous présentons une analyse théorique qui évalue leur complexité. Nous comparons les résultats théoriques avec la pratique. Nous étudions l'efficacité de tous ces algorithmes et cela nous permet de fournir des paramètres qui sont sûrs et qui peuvent être utilisés en pratique.

Nous poussons l'état de l'art en améliorant les algorithmes existants à l'aide de deux frameworks. Dans le premier, nous séparons les algorithmes résolvant LPN en étapes atomiques. Nous étudions la complexité de chaque étape ainsi que la façon dont ces étapes interagissent. Nous construisons un algorithme qui optimise leur utilisation. Étant donné une instance LPN, qui est caractérisée par un niveau de bruit et une taille de secret, notre algorithme fournit les étapes à suivre pour résoudre l'instance avec une complexité optimale. De cette façon, nous pouvons évaluer si une instance LPN fournit la sécurité nécessaire et nous montrons quelles sont les instances sûres pour les applications qui utilisent LPN.

Le second framework traite des problèmes qui peuvent être décomposés en étapes de même complexité. Ici, nous supposons que nous avons un adversaire qui a accès à un nombre fini ou infini d'instances du même problème. Le but de l'adversaire est de résoudre une seule instance le plus vite possible. Notre framework fournit la stratégie qui permet d'atteindre cet objectif. Nous adaptons un algorithme qui résout LPN dans ce framework et nous montrons que nous pouvons améliorer sa complexité dans le scénario où l'adversaire est restreint en nombre de requètes. Nous montrons aussi que d'autres problèmes, comme celui demandant de deviner un mot de passe, peuvent être modelés à l'aide de ce même framework.

Mots clefs: Learning Parity with Noise, LPN, étude algorithmique, cryptographie, transformée d'Hadamard, BKW, LF1, LF2, code couvrant, élimination de Gauss-Jordan, cryptographie post-quantique, cryptographie à clé publique, deviner un mot de passe

Acknowledgments

If I was asked to describe shortly my PhD, I would say that my years as a PhD student meant coping and working with the unknown, learning by myself and from the others through collaboration, spending time with my family and friends and receiving their support. I wouldn't have been able to do this by myself.

First, I would like to express my gratitude towards my supervisor Prof. Serge Vaudenay. Serge, thank you for accepting me in your laboratory to do my PhD studies. Thank you for teaching me how to do research properly and guiding me through my PhD. You were always helpful and had a lot of patience in answering all my questions.

But, most importantly, I want to thank you for trusting me. Your confidence in me was contagious and because of you, I completely changed my attitude. While at the beginning I was very unsure on myself when it came to research and teaching, at the end of my PhD, I can say that I trust myself, I got to discover that I enjoy teaching and I am aware of what my strengths and my weaknesses are. I believe you had an optimised algorithm on how to give me tasks so that I become more efficient. Thank you Serge for all your help. I truly feel honoured that I had the chance to work in LASEC and to learn from you.

Secondly, I would like to thank the members of my PhD thesis committee. I thank Prof. Ola Svensson, my committee president. I also express my gratitude towards Prof. Piere-Alain Fouque and Prof. Willi Meier who accepted to be my external jury examiners. Thank you for your time and effort to be part of my committee. Finally, I wish to thank Prof. Arjen Lenstra for accepting to be my internal jury examiner. Thank you Arjen for your direct and frank comments.

Next, I would like to thank my two office mates, with whom I shared the office INF 239. My first three years and a half of my PhD were closely watched by the honorary EPFL-NSA member Alexandre Duc. Alexandre welcomed me in his office and made me feel like at home in LASEC from the first days. He has always offered his help and support regardless of whether it was research related or not. Thank you Alexandre for your kindness, for your encouragements, for being a funny office mate. I learnt a lot from you when it came to working with students and supervising them. You are my model for how to be a dedicated assistant and help the students. Thank you for all our funny discussions. Thank you for all our non-funny discussions about Switzerland (one does

not question the rules from Switzerland). Thank you for all the hikes you organized and for making me discover this beautiful country! And, the most important, thank you for the chocolate you always kindly offered to me and the whole lab, daily. We now continue this tradition in the lab. Last but not least, thank you for all your valuable comments regarding my thesis. You are a member of the very small exclusive club of the people that actually read my thesis.

Almost one year after Alexandre finished his PhD, Gwangbae Choi joined LASEC and I shared the office with him during the last months of my PhD. Gwangbae has a very calm personality and I believe this helped me being less stressed while I was preparing my thesis. Thank you Gwangbae for all the jokes about doing ”nothing” during your weekends and for our discussions about traditions in South Korea. Good luck for your PhD!

While in the laboratory we are all concerned with research, there is a person that actually takes care of us when it comes to going to conferences, taking holiday and all the administrative things that have to be done at EPFL and in Switzerland. I would like to kindly thank Martine Corval, our secretary. Martine has always been very helpful, always welcoming us with a smile in her office and trying to solve our problems the best she could. Thank you Martine for all the help.

My next thanks go to the present and former LASEC members: Prof. Serge Vaudenay, Dr Philippe Oechslin, Martine Corval, Dr Aslı Bay, Dr Petr Sušil, Dr Rafik Chaabouni, Dr Alexandre Duc, Damian Vizár, Handan Kılınç, Fatih Balli, Gwangbae Choi, Dr Atefeh Mashatan, Prof. Katerina Mitrokotsa, Dr Ioana Boureanu-Carlson, Dr Miyako Ohkubo, Dr Jialin Huang, Prof. Sebastian Faust, Dr Reza Reyhanitabar, Dr Adeline Langlois–Roux, Prof. Divesh Aggarwal and to students that did an internship during summer at LASEC: Iosif Spulber, Florian Tramèr, Arka Rai Choudhuri, Fatma Betül Durak, Bogdan Ursu, Mathilde Igier, Ingy El Sayed Aly, Amir Ali Moinfar, Dario Gjorgjevski. I feel grateful to have worked in an environment that made me feel welcomed and happy even during my less happy days.

Thank you Aslı for initiating me in Turkish food and for making our days happier with your stories about your daughter Zeynep Nur. Thank you Petr for all your jokes as now I know a byte more on how to play with words. Damian, thank you for all our laughs. It is refreshing to have, in the lab, a person that accepts life as it is, with a smile on the face. It’s been really fun to plot tricks on other people and to hear all your funny stories. Thank you also for manifesting your artistic talent on my white board. Handan, thank you for all the delicious Turkish food you prepared for us. You were always kind to me and I enjoyed a lot our girly discussions. Handan and Damian, it has been a pleasure to TA with you for the ”Cryptography and Security” course. Fatih, thank you for all our philosophical discussions and for taking the responsibility to continue the tradition of our ”secret” association. Ioana, thank you for all your kindness and help with the moving. Thank you for bringing dynamism in the lab. Miyako, it has been a pleasure to have you as a colleague. Reza, thank you for all the stories about Iran and for the

delicious Iranian food your wife prepared for us at Christmas. Sebastian, thank you for keeping up our working spirits. Adeline, it was fun to have you in the lab. Thank you for organizing and gathering people for games nights. Divesh, thank you for our discussions about the Indian wedding. Arka, Betül, Bogdan, Mathilde, Ingy, Amir and Dario, thank you for bringing energy and life in the lab during the summers. It was nice to have you in LASEC.

Next I would like to thank my co-authors: Prof. Serge Vaudenay, Dr Ioana Boureanu–Carlson, Florian Tramèr and John Gaspoz. Ioana, thank you for your guidance and help with our primeless research. Florian, thank you for all your work. You are very talented and you always did your work very fast. I am sure you will have very good results in research. John, it has been a pleasure to supervise and work with you and I enjoyed the trip to ArcticCrypt.

I would also like to express my gratitude towards the person that introduced me in the world of research, Prof. Ferucio Laurenţiu Ţiplea. Prof. Ţiplea was the first to explain to me what is research, encouraged me to do research and to do my studies abroad. Thank you for your wise advices that I am happy to have followed.

The EPFL environment is excellent for research and for working with the best experts, but also for meeting new people and making friends. I want to thank Tudor David, Régis Blanc, Alevtina Dubovitskaya, Pamela Delgado, Onur Yürüten and Clemens Nyffeler. Thank you for the nice moments that you shared with me. Onur, thank you for our discussions about politics and the human nature. Régis thank you for all the fun coffee breaks, for the games designed by you that I could test and for the delicious carrot cake you prepared. Tudor, thank you for all the hikes we did together. Thank you for being my Romanian fellow during the master and the PhD. Alevtina, thank you for our discussions regarding Romanian and Russian traditions. Pamela, thank you for all the nice stories about your daughter Aileen and for the nice lunches we had.

While building new friendships at EPFL, I maintained my old friendships that were always a breeze of fresh air during my stressful periods. I would like to express my gratitude towards Ana Maria Butnaru, Roman Beatrice, Irina Ţenu, Mihaela Florea, Andreea Vasilachi, Alexandru Sîrghi, Oana Niţu, Maria Obreja, Andrei Zapodeanu and Rodica Bozianu for their friendship and support. Thank you Ana for the delicious cakes you brought to me when you visit me. Thank you Ir, Ela, Bea for all the relaxing trips we organized in the last years. Thank you Andreea for all our discussions about life and for the Belgian chocolate. Thank you Rodi for all the weekends spent together in Paris. Thank you Alex for all the support, care and help during this PhD. Thank for listening to me when I needed. You know very well how is the life of a PhD student without needing to be one. Thanking to all of you for a particular thing cannot express a friendship. So, I wish to thank you for your support and for coping with me for so many years.

Last but not least, I want to thank my family for their love and unconditional support. My father and my mother, Valeriu and Georgeta, are my models in life. I still have so many things to learn from them and from my sister, Corina. I have been blessed to have a family that always encouraged me and had trust that I can achieve my goals. They are my biggest supporters. Thank you mom and dad for your love! Thank you Corina for being the best sister and a dear friend!

As words cannot encompass everything, I hope that during my PhD I showed you, through my actions, my gratitude and expressed my luck to have you all.

Mulţumiri

În primul rând, vreau să îmi exprim recunoștiința față de îndrumătorul meu, Prof. Serge Vaudenay. Serge, îți mulțumesc că m-ai acceptat în laboratorul tău. Îti mulțumesc că m-ai învățat cum să fac cercetare și că m-ai îndrumat în timpul doctoratului. Mi-ai oferit întotdeauna ajutorul tău și ai avut multă răbdare cu mine. Dar, cel mai important, îți mulțumesc că ai avut încredere în mine. Încrederea ta a fost contagioasă și datorită ție mi-am schimbat atitudinea față de mine. La începutul doctoratului eram nesigură pe mine când venea vorba de cercetare și predat. Acum am mult mai multă încredere în mine și sunt conștientă de punctele mele slabe și de calitățile mele. Ai avut o metodă optimă de a îmi da responsabilități care m-a făcut să lucrez eficient. Mulțumesc Serge pentru tot ajutorul acordat. Sunt onorată că am avut oportunitatea să lucrez în LASEC și să învăț de la tine.

În al doilea rând, adresez sincere mulţumiri comitetului meu. Mulțumesc Prof. Ola Svensson, președintele comitetului. Sunt recunoscătoare față de Prof. Pierre-Alain Fouque și Prof. Willi Meier care au acceptat să facă parte din comitet. Îi mulțumesc Prof. Arjen Lenstra pentru comentariile dumnealui.

Doresc să le mulțumesc celor doi colegi de birou, cu care am împărțit biroul INF 239. Primii trei ani și jumătate au fost supervizați de membrul onorific EPFL-NSA Alexandre Duc. Alexandre m-a primit în biroul lui și m-a făcut să mă simt binevenită în LASEC încă din primele zile. Mi-a oferit ajutorul lui întotdeauna, fie că era vorba de cercetare sau nu. Mulțumesc Alexandre pentru prietenia ta, pentru încurajările tale și pentru că ai fost un coleg de birou simpatic. Am învățat foarte multe de la tine în ceea ce privește cum să lucrez, să ajut și să îndrum studenții. Îți mulțumesc pentru toate discuțiile amuzante pe care le-am avut. Îți mulțumesc pentru discuțiile despre regulile ce trebuie respectate în Elveția. Îți mulțumesc pentru toate drumețiile pe care le-ai organizat. Și cel mai important, îți mulțumesc pentru ciocolata pe care ne-ai oferit-o zi de zi. Continuăm acum această tradiție în laborator. În final, îți mulțumesc pentru tot ajutorul oferit. Ești membrul unui club mic și select compus din toate persoanele care au citit într-adevăr teza mea.

La aproape un an după ce Alexandre și-a obținut doctoratul, Gwangbae Choi a venit în LASEC și am împărțit cu el biroul pentru câteva luni. Gwangbae are o personalitate calmă și asta m-a ajutat să fiu mai puțin stresată când îmi pregăteam teza de doctorat.

Mulțumesc Gwangbae pentru toate glumele și pentru toate discuțiile noastre despre tradițiile din Coreea de Sud. Multa baftă pentru doctoratul tău!

În timp ce în laborator suntem cu toții preocupați de cercetare, cineva are grijă de noi când vine vorba de călătorit la conferințe și de rezolvat treburile admnistrative. Doresc să îi mulțumesc lui Martine Corval, secretara laboratorului nostru. Martine m-a ajutat întotdeauna, încercând să rezolve problemele mele administrative cât mai bine. Mulțumesc Martine pentru tot ajutorul oferit.

Vreau să le mulțumesc membrilor LASEC: Prof. Serge Vaudenay, Dr Philippe Oechslin, Martine Corval, Dr Aslı Bay, Dr Petr Sušil, Dr Rafik Chaabouni, Dr Alexandre Duc, Damian Vizár, Handan Kılınç, Fatih Balli, Gwangbae Choi, Dr Atefeh Mashatan, Prof. Katerina Mitrokotsa, Dr Ioana Boureanu-Carlson, Dr Miyako Ohkubo, Dr Jialin Huang, Prof. Sebastian Faust, Dr Reza Reyhanitabar, Dr Adeline Langlois–Roux, Prof. Divesh Aggarwal cât și studenților care au făcut un internship în timpul verii la LASEC: Iosif Spulber, Florian Tramèr, Arka Rai Choudhuri, Fatma Betül Durak, Bogdan Ursu, Mathilde Igier, Ingy El Sayed Aly, Amir Ali Moinfar, Dario Gjorgjevski. Sunt recunoscătoare că am avut șansa să lucrez într-un mediu binevoitor, care mi-a adus zâmbetul pe buze chiar și în zilele mele mai puțin fericite.

Îți mulțumesc Aslı că m-ai inițiat în bucătăria turcească și că ne-ai încântat cu poveștile despre fiica ta, Zeynep Nur. Mulțumesc Petr pentru toate glumele tale. Damian, îți mulțumesc pentru toate momentele amuzante. Este plăcut să ai pe cineva în laborator care acceptă orice situație cu zâmbetul pe buze. De asemenea îți mulțumesc pentru desenele făcute pe tabla mea. Handan, îți mulțumesc pentru toate felurile de mâncare turcească pe care le-ai pregătit pentru noi. Ai fost întotdeauna binevoitoare cu mine și m-am bucurat de toate conversațiile noastre. Handan și Damian, vreau să vă spun că a fost o plăcere să lucrez alături de voi pentru cursul "Cryptography and Security". Fatih, mi-a făcut plăcere să am discuții filosofice cu tine. Mulțumesc că ai acceptat să duci mai departe tradiția asociației noastre "secrete". Ioana, îți mulțumesc pentru tot ajutorul oferit. Îți mulțumesc pentru toată implicarea ta în activitatea laboratorului. Miyako, a fost o deosebită plăcere să fiu colega ta. Reza, mi-a făcut plăcere să aud poveștile tale despre Iran și îți mulțumesc pentru mâncarea iraniană pe care ai pregătit-o de Crăciun pentru noi. Sebastian, îți mulțumesc că ne-ai arătat care e stilul german de a face cercetare. Adeline, a fost o plăcere să lucrez cu tine.

Doresc să adresez sincere mulțumiri coautorilor mei: Prof. Serge Vaudenay, Dr Ioana Boureanu–Carlson, Florian Tramèr și John Gaspoz. Ioana, mulțumesc că m-ai îndrumat și m-ai ajutat în proiectul "Primeless Cryptography". Florian, îți mulțumesc pentru toată munca depusă. Am certitudinea că vei avea rezultate foarte frumoase în cercetare. John, a fost o plăcere să lucrez alături de tine.

De asemenea, doresc să îmi exprim recunoștința față de domnul Prof. Ferucio Laurenţiu Ţiplea. Prof. Ţiplea a fost persoana care m-a încurajat să fac cercetare și să studiez în străinătate. Vă mulțumesc pentru toate sfaturile și sunt foarte fericită că le-am urmat.

Mediul de la EPFL mi-a oferit ocazia să cunosc noi oameni și să leg noi prietenii. Vreau să mulțumesc lui Tudor David, Régis Blanc, Alevtina Dubovitskaya, Pamela Delgado, Onur Yürüten și Clemens Nyffeler pentru toate momentele frumoase pe care le-ați împărtășit cu mine.

În timpul doctoratului, momentele cu prietenii vechi au fost ca o briză de aer revigorant. Vreau să îmi exprim recunoștința față de Ana Maria Butnaru, Roman Beatrice, Irina Ţenu, Mihaela Florea, Andreea Vasilachi, Alexandru Sîrghi, Oana Niţu, Maria Obreja, Andrei Zapodeanu și Rodica Bozianu. Mulțumesc Ana pentru vizitele tale și pentru dulciurile pe care mi le aduceai din România. Ir, Ela, Bea, mulțumesc pentru călătoriile relaxante pe care le-am făcut împreună în ultimii ani. Andreea, mulțumesc pentru ciocolata belgiană și pentru toate discuțiile noastre. Rodi, îți mulțumesc pentru toate weekendurile petrecute împreuna la Paris. Alex, îți mulțumesc pentru tot suportul, grija, răbdarea și ajutorul pe care mi l-ai acordat în timpul doctoratului. Îți mulțumesc că m-ai ascultat atunci când aveam nevoie. Doar mulțumindu-vă pentru un singur lucru nu ilustrez ce reprezintă prietenia noastră. Vă mulțumesc din suflet că ați fost alături de mine în acești ani.

În cele din urmă, doresc să mulțumesc familiei mele pentru iubirea lor și suportul necondiționat. Mama mea și tatăl meu, Georgeta și Valeriu, reprezintă modelele mele de urmat în viață. Am încă multe lucruri de învățat de la ei și de la sora mea, Corina. Am fost binecuvântată să cresc într-o familie care m-a încurajat întotdeauna și a crezut în mine. Ei sunt cei mai mari suporteri ai mei. Mama și tata, vă mulțumesc pentru iubirea pe care mi-o purtați! Mulțumesc Corina că ești cea mai bună surioară din lume!

Întrucât cuvintele mele exprimă prea puțin, sper că în timpul acestor ani v-am arătat, prin faptele mele, că sunt recunoscătoare și norocoasă să vă am aproape de mine.

Table of Contents

Chapter 1

Introduction

The area of *cryptography* has developed tremendously in the last decades, both in its definition and in its use. Nowadays, cryptography is the science of providing primitives/solutions for scenarios in which we deal with adversaries and we want to ensure properties like *confidentiality.* Cryptography is ubiquitous, it is a tool that every person is using. Anyone with access to the Internet benefits from the security that cryptography offers. It can be about passwords, accessing the bank account or securely storing data in the cloud. In all these scenarios we use cryptography. Whether it is about ensuring *confidentiality, integrity, authentication, non repudiation, unpredictability*, cryptography offers a large palette of primitives.

The most well-known primitive of cryptography is the *encryption scheme.* An encryption scheme ensures confidentiality, i.e. when a message m is encrypted, only the legitimate recipient can decrypt and have access to it. In cryptography, we have two types of encryption: *symmetric* and *public-key*. On the symmetric encryption side we have the *block ciphers* (e.g. *DES* [Des77], *AES* [FIP01]) and the *stream ciphers* (e.g. *RC4*). Two main representatives of the public-key encryption are *RSA* [RSA78] and *ElGamal* [Gam85]. These two encryption schemes base their security on two hard problems: *factorization* and *discrete logarithm*, respectively. We believe that these two problems are hard based on the unsuccessful efforts of the cryptography and mathematics community to find efficient algorithms to solve them. If a new algorithm is able to solve easily one of these problems, then the corresponding encryption scheme becomes insecure. RSA and ElGamal are extensively studied and used in practice.

Besides the two aforementioned hard problems, we have several other candidates: *Learning Parity with Noise* (LPN), *Learning with Errors* (LWE) [Reg05], the *quadratic residuosity* problem, the *shortest vector* problem, etc..

One might ask: if we have the factorization and the discrete logarithm problems, which we believe to be hard, why should we try to analyse new problems?

The first and the most important reason is that we need to have *crypto diversity.* Assume half of our encryption schemes base their security on factorization and half on discrete logarithm. Having only two standard hard problems can simplify the choice of someone who needs to use an encryption scheme. But, if one discovers how to easily solve one of these problems, then half of our encryption schemes become insecure. Cryptography needs to offer a range of hard problems that can be the starting point for an encryption scheme. In order to create a pool of such problems, we need to carefully study them in order to assess their hardness. Different problems can better fit to different scenarios or platforms. So, having more choices can offer more flexibility and a better match.

Secondly, both problems are easily solved by a *quantum computer* with the *Shor's algorithm* [Sho97]. Thus, we cannot count on them in a quantum computer era and we need to provide alternatives that are post-quantum resistant. While there is yet no quantum computer capable of solving such problems, progresses are done [VSB+01, LBYP07, LWL+07, LBC+12, MLL+12, XZL+12, DB14] in this direction and we cannot know when this will become reality. Hence, we need to propose a standard for post-quantum cryptography.

This thesis focuses on the analysis of the Learning Parity with Noise (LPN) problem, on the implementation of the LPN solving algorithms existing in the literature and on improvements to these algorithms.

We study LPN, as it is a good candidate for post-quantum cryptography. It is also a good candidate for lightweight devices.

Informally, the LPN problem asks to solve a noisy binary system of linear equations (all the operations are done modulo 2). I.e., given an uniformly distributed secret vector $s \in \{0,1\}^k$ and an uniformly distributed vector $v \in \{0,1\}^k$, for which we compute their inner product $\langle s, v\rangle = \sum_{i=1}^{k} s_i v_i$, we make public the value of v and $c = \langle s, v\rangle \oplus d$, where $d \in \{0,1\}$ is the noise. The value of d is 1 with probability $\tau \in [0, \frac{1}{2}]$. Given many such equations, one is asked to recover the value of s. Note that the noise is the variable that gives hardness to the problem. If we always have $d = 0$, then s can be easily recovered with the Gaussian elimination algorithm. Depending on the application where it is instantiated, the noise level in LPN can be adjusted and offer a degree of flexibility. An LPN instance is characterized by two parameters: the size of the secret, k, and the noise level, τ.

The LWE problem is a generalization of LPN, where, instead of working modulo 2 we work in a ring $\mathbb{Z}_q$. While there are quantum and classical reductions [Reg05, Pei09, BLP+13] from a worst-case problem on lattices to LWE that attest the hardness of LWE, this reduction does not exist for LPN. Thus, the best way to assess its hardness is by trying to design and improve algorithms that solve it.

Here, we analyse the existing LPN solving algorithms. This allows us to define the memory and time complexity needed to solve this problem. While this is a theoretical work, we validate our analysis by comparing the theoretical results with the practical ones offered by an implementation. We further improve the complexity of an LPN solving algorithm. This allows us to asses the hardness of LPN and to propose secure parameters for the cryptographic primitives that base their security on it.

For instance, we show that RSA with 2048-bit moduli or *ECDH* with a secp224r1 curve is equivalent to LPN with 512-bit secret and $\tau = 0.4$.

Thus, this thesis deepens the study of the LPN problem by formally analysing it, by improving the existing algorithms and by validating our theoretical results with practical ones.

Outline of the Thesis

We provide the formal definition of LPN and give an overview of the existing LPN heuristic solving algorithms in Chapters 2 and 3. All these algorithms are presented in a unified framework where an LPN solving algorithm has two phases: a reduction phase and a solving phase. The reduction phase reduces the size of the secret and the solving phase recovers the truncated secret. This framework allows us to better compare the algorithms and to give an intuition on which methods are more efficient. We describe the algorithms as they were presented in the literature and we bring our improvements by providing a more detailed analysis and by using better bounds. While most of the algorithms are generic, we present three algorithms for the case when the secret is sparse, i.e. it has a small Hamming weight. For all the algorithms we describe, we state their complexity in terms of number of LPN equations, memory and time.

Our theoretical analysis is compared with the practice in Chapter 4. We implement the algorithms described in Chapter 3 and study the gap between the theoretical and the practical results. We discover that the results provided by the implementation are very close to what the theory dictates. After this comparison, we compare all the LPN solving algorithms for the case when the secret is sparse. Having validated our heuristic, we provide the security offered by each algorithm when we vary the size of the secret and the level of noise. Having these results, the LPN based cryptographic primitives can instantiate LPN with secure parameters. This also allows us to see what is the best LPN solving algorithm depending on the noise level.

Chapters 5 and 6 focus on methods that allow us to improve the performance of the LPN solving algorithms. In Chapter 5, we represent an LPN algorithm as a sequence of reduction steps followed at the end by a solving step. We maintain the same framework as in Chapter 3 but we look more into the details on how we can improve the reduction phase. We construct a graph of all possible reduction steps and search for the chain of steps that minimizes the time complexity while trying to keep the noise level as low as possible. For this, we construct an algorithm that allows us to search in an efficient way the optimal chains. This method automatizes the LPN solving algorithms: given any LPN instance our algorithm gives us what is the best complexity we can achieve together with the steps that need to be followed in order to obtain that complexity.

Chapter 6 introduces a framework where we have an adversary that can mount several attacks (e.g. key searches). These attacks can be broken down into steps of equal complexity. An attacker can stop and resume an attack. The question we answer in this scenario is what strategy an adversary should adopt such that he succeeds in one of these attacks as fast of possible. This framework is a general one and can be adopted to several problems. In this chapter, we focus on LPN and password guessing. With this framework we improve the complexity of an LPN solving algorithm in which we are limited in the number of equations.

We conclude and present the future work in Chapter 7.

Personal Bibliography

Below is the list of publications that were published during this thesis. Entries in bold are included in this dissertation.

[1] **Sonia Bogos and Serge Vaudenay. Optimization of LPN Solving Algorithms. In Jung Hee Cheon and Tsuyoshi Takagi, editors. *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I, volume 10031 of Lecture Notes in Computer Science, pages 703-728. Springer 2016.***

[2] Sonia Bogos, John Gaspoz and Serge Vaudenay. Cryptanalysis of a Homomorphic Encryption Scheme. *In ARCTICCRYPT. 2016.*

[3] **Sonia Bogos and Serge Vaudenay. How to sequentialize independent parallel attacks? - biased distributions have a phase transition. In Tetsu Iwata and Jung Hee Cheon, editors. *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II, volume 9453 of Lecture Notes in Computer Science, pages 704–731. Springer, 2015.***

[4] **Sonia Bogos, Florian Tramèr, and Serge Vaudenay. On solving LPN using BKW and variants - Implementation and analysis. *In Cryptography and Communications, volume 8, number 3, pages 331–369, 2016.***

[5] Sonia Bogos, Ioana Boureanu and Serge Vaudenay. Primeless Factoring-Based Cryptography -Solving the Complexity Bottleneck of Public-Key Generation-. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel and Reihaneh

Safavi-Naini, editors. *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings, volume 7954 of Lecture Notes in Computer Science, pages 552-569. Springer 2012.*

[6] Sonia Bogos, Ioana Boureanu and Serge Vaudenay. Primeless Factoring-Based Cryptography -Solving the Complexity Bottleneck of Public-Key Generation-. (Extended Abstract). *In Yet Another Conference on Cryptography 2012, Porquerolles Island, France, September 24-28, 2012.*

[7] Sonia Bogos, Serge Vaudenay. Cryptanalysis of the Double-Moduli Cryptosystem. *In Int'l J. of Communications, Network and System Sciences (ISSN: 1913-3723), volume 5, number 12, pages 834-838, 2012.*

Chapter 2

Preliminaries

In this chapter, we introduce notations, definitions and results that are used in the next chapters. In Section 2.1, we present the notations used for describing the LPN problem. In Section 2.2, we provide the formal definition of the Learning Parity with Noise (LPN) problem and describe the parameters that characterize an LPN solving algorithm. We recall some of the Hoeffding's bounds and the Central Limit Theorem in Sections 2.3 and 2.4. These results are used in the analysis of LPN solving algorithms. Section 2.5 gives the definition of the Walsh Hadamard Transform that we use for our algorithms. Definitions for linear codes, perfect and quasi-perfect codes are presented in Section 2.6.

2.1 Notations and Preliminaries

Let $\langle \cdot, \cdot \rangle$ denote the inner product and let $\oplus$ denote the bitwise XOR. By $\mathbb{Z}_p$ we denote the quotient group $\mathbb{Z}/p\mathbb{Z}$ and we have $\mathbb{Z}_2 = \{0, 1\}$. For a domain $\mathcal{D}$, we denote by $x \overset{U}{\leftarrow} \mathcal{D}$ the fact that x is drawn uniformly at random from $\mathcal{D}$. We use small letters for vectors and capital letters for matrices. We represent a binary vector v of size k as $v = (v_1, \ldots, v_k)$, where v_i is the i^{th} bit of v. We denote the Hamming weight of a vector v, i.e. the number of non-zero elements of v, by $\mathsf{HW}(v)$.

By Ber_τ we define the Bernoulli distribution with parameter τ, i.e. for a random variable X following Ber_τ,

$$\Pr[X = 1] = \tau = 1 - \Pr[X = 0].$$

The bias of a boolean random variable X is defined as $\delta = E((-1)^X)$. Thus, for a Bernoulli variable, we have $\delta = 1 - 2\tau$. By Ber_τ^k we denote the binomial distribution with parameters k and τ. I.e. Ber_τ^k is the sum of k independent random variables that follow the Ber_τ distribution. For a random variable Y following Ber_τ^k, we have

$$\Pr[Y = i] = \begin{cases} \binom{k}{i}\tau^i(1-\tau)^{k-i} & \text{for } i \leq k \\ 0 & \text{for } i > k. \end{cases}$$

Given $\mathcal{X}$ and $\mathcal{Y}$, two discrete distributions over a support $\mathcal{Z}$, we define by $d(\mathcal{X}, \mathcal{Y})$ the statistical distance between the two distributions as follows:

$$d(\mathcal{X}, \mathcal{Y}) = \frac{1}{2} \sum_{z \in \mathcal{Z}} |\mathcal{X}(z) - \mathcal{Y}(z)|,$$

where by $|x|$ we denote the absolute value of the real value x.

2.2 Learning Parity with Noise Problem

LPN is a well-known problem studied in cryptography, coding theory and machine learning. LPN is believed to be resistant to quantum computers. This makes it a good candidate to the number-theoretic problems (e.g. factorization and discrete logarithm) which can be solved easily with quantum algorithms. Also, due to its simplicity, it is a nice candidate for lightweight devices. Among the cryptographic applications where LPN or LPN variants are deployed, we first have the HB family of authentication protocols: HB [HB01], HB$^+$ [JW05], HB^{++} [BCD06], HB$^{\#}$ [GRS08] and AUTH [KPC$^+$11]. An LPN-based authentication scheme secure against Man-in-the-Middle attacks was presented in Crypto 2013 [LM13]. There are also several encryption schemes based on LPN: Alekhnovich [Ale03] presents two public-key schemes that encrypt one bit at a time. Later, Gilbert, Robshaw and Seurin [GRS08] introduce LPN-C, a symmetric cryptographic system. Two schemes that improve upon Alekhnovich's scheme are introduced by Döttling at al [DMN12] and Damgård and Sunoo [DP12]. In PKC 2014, Kiltz et al. [KMP14] propose an alternative scheme to [DMN12]. Duc and Vaudenay [DV13] introduce HELEN, an LPN-based public-key scheme for which they propose concrete parameters for different security levels. There are several PRNG [BFKL93, ACPS09] based on LPN.

The hardness of this problem was discussed first by Kearns in [Kea93]. It is believed to be hard and is closely related to the long-standing open problem of efficiently decoding random linear codes. Also, LPN is a special case of the Learning With Errors (LWE) problem [Reg05], where, instead of the ring $\mathbb{Z}_p$, we work in $\mathbb{Z}_2$. There are quantum and classical reductions [Reg05, Pei09, BLP$^+$13] from a worst-case problem on lattices to LWE that attest the hardness of LWE. But these reductions do not work for the LPN problem and thus, prove nothing about its hardness.

The LPN problem can be seen as a noisy system of linear equations in the binary domain. More specifically, we have a secret s and an adversary that has access to an LPN oracle which provides him tuples of uniformly distributed binary vectors v_i and the inner product between s and v_i to which some noise was added. The noise is represented by a Bernoulli variable with a probability τ of the noise bit to be 1. In the following, we assume w.l.o.g. that $\tau \in [0, \frac{1}{2}]$, otherwise we can change τ to $1 - \tau$. The goal of the adversary is to recover the secret s. So, the problem asks to recover a secret vector s given access to noisy inner products of itself with random vectors. Below, we present the formal definition.

Definition 2.1 (LPN oracle). *Let $s \xleftarrow{U} \mathbb{Z}_2^k$, let $\tau \in]0, \frac{1}{2}[$ be a constant noise parameter and let Ber_τ be the Bernoulli distribution with parameter τ. Denote by $D_{s,\tau}$ the distribution defined as*

$$\{(v, c) \mid v \xleftarrow{U} \mathbb{Z}_2^k, c = \langle v, s\rangle \oplus d, d \leftarrow \mathsf{Ber}_\tau\} \in \mathbb{Z}_2^{k+1}.$$

An LPN oracle $\mathcal{O}_{s,\tau}^{\mathsf{LPN}}$ is an oracle which outputs independent random samples according to $D_{s,\tau}$.

2.2.1 Search LPN

Definition 2.2 (Search LPN). *Given access to an LPN oracle $\mathcal{O}_{s,\tau}^{\mathsf{LPN}}$, find the vector s. We denote by $\mathsf{LPN}_{k,\tau}$ the LPN instance where the secret has size k and the noise parameter is τ. Let $k' \leq k$. We say that an algorithm $\mathcal{M}$ (n, t, m, θ, k')-*solves *the search $\mathsf{LPN}_{k,\tau}$ problem if*

$$\Pr[\mathcal{M}^{\mathcal{O}_{s,\tau}^{\mathsf{LPN}}}(1^k) = (s_1 \ldots s_{k'}) \mid s \xleftarrow{U} \mathbb{Z}_2^k] \geq \theta,$$

and $\mathcal{M}$ runs in time t, uses memory m and asks at most n queries from the LPN oracle.

Note that we consider here the problem of recovering only a part of the secret. Throughout the literature, this is how the LPN problem is formulated. The reason for doing so is that the recovery of the first k' bits dominates the overall complexity. Once we recover part of the secret, the new problem of recovering a shorter secret of $k - k'$ bits is easier. We will discuss more about this in Section 3.4.

Easy instances of LPN occur when $\tau < \frac{1}{k}$, as in this case we can assume that we have k equations with no noise and we can recover the secret with Gaussian elimination. The other extreme case is when $\tau = \frac{1}{2}$ and where we cannot recover the secret s.

An equivalent way to formulate the search $\mathsf{LPN}_{k,\tau}$ problem is as follows: given access to a random matrix $A \in \mathbb{Z}_2^{n \times k}$ and a column vector c over $\mathbb{Z}_2^n$, such that $As^T \oplus d = c$, find the vector s. Here the matrix A corresponds to the matrix that has the vectors v on its rows, s is the secret vector of size k and c corresponds to the column vector that contains the noisy inner products. The column vector d is of size n and contains the corresponding noise bits.

2.2.2 Decision LPN

The LPN problem has also a decisional form. The decisional $\mathsf{LPN}_{k,\tau}$ asks to distinguish between the uniform distribution over $\mathbb{Z}_2^{k+1}$ and the distribution $\mathcal{D}_{s,\tau}$. A similar definition for an algorithm that solves decisional LPN can be adopted as above.

Definition 2.3 (Decision LPN). *Let $\mathcal{U}_{k+1}$ denote an oracle that outputs binary random vectors of size $k+1$. We say that an algorithm $\mathcal{M}$ (n,t,m,θ)*-solves *the decisional* $\mathsf{LPN}_{k,\tau}$ *problem if*

$$| \Pr[\mathcal{M}^{\mathcal{O}^{\mathsf{LPN}}_{s,\tau}}(1^k) = 1] - \Pr[\mathcal{M}^{\mathcal{U}_{k+1}}(1^k) = 1] \,| \geq \theta$$

and $\mathcal{M}$ runs in time t, uses memory m and needs at most n queries.

The search and the decisional LPN problems are polynomially equivalent [KSS10, BFKL93]. The following lemma expresses this result.

Lemma 2.4 ([KSS10, BFKL93]). *If there is an algorithm $\mathcal{M}$ that (n,t,m,θ)-solves the decisional $\mathsf{LPN}_{k,\tau}$, then one can build an algorithm $\mathcal{M}'$ that (n',t',m',θ',k)-solves the search $\mathsf{LPN}_{k,\tau}$ problem, where $n' = \mathcal{O}(n \cdot \theta^{-2} \log k)$, $t' = \mathcal{O}(t \cdot k \cdot \theta^{-2} \log k)$, $m' = \mathcal{O}(m \cdot \theta^{-2} \log k))$ and $\theta' = \frac{\theta}{4}$.*

We do not present the details of this result as it is outside the scope of this thesis. We only analyse the solving algorithms for search LPN. From now on, we will refer to it simply as LPN.

2.3 Hoeffding's Bounds

Our results from Chapter 3 use the following Hoeffding bounds in order to formalize the solving step of an LPN solving algorithm.

Theorem 2.5. *[Hoe63] Let $X_1, X_2, \ldots, X_n$ be n independent random variables. We are given that $\Pr[X_i \in [\alpha_i, \beta_i]] = 1$ for $1 \leq i \leq n$. We define $X = X_1 + \ldots + X_n$ and $E(X)$ is the expected value of X. We have that*

$$\Pr[X - E(X) \geq \lambda] \leq e^{-\frac{2\lambda^2}{\sum_{i=1}^{n}(\beta_i - \alpha_i)^2}}$$

and

$$\Pr[X - E(X) \leq -\lambda] \leq e^{-\frac{2\lambda^2}{\sum_{i=1}^{n}(\beta_i - \alpha_i)^2}},$$

for any $\lambda > 0$.

2.4 Central Limit Theorem

The Central Limit Theorem is used in Chapter 3 as it improves upon the result that uses the Hoeffding bounds.

Theorem 2.6. *Let $X_1, X_2, \dots, X_n$ be n independent identically distributed (i.i.d) random variables. We define $X = X_1 + \dots + X_n$, $E(X_1)$ as the expected value of X_1, $\mathsf{Var}(X_1)$ as the variance of X_1 and*

$$Z_n = \frac{X - nE(X_1)}{\sqrt{n\mathsf{Var}(X_1)}}.$$

Then, as n approaches infinity, Z_n converges in distribution to a normal distribution with mean 0 and variance 1, i.e. $Z_n \xrightarrow[n\to\infty]{d} \mathcal{N}(0,1)$. This means that for all t,

$$\Pr[Z_n \le t] \xrightarrow[n\to\infty]{} \varphi(t),$$

where $\varphi(t) = \frac{1}{2} + \mathsf{erf}(\frac{t}{\sqrt{2}})$ and $\mathsf{erf}(t) = \frac{2}{\sqrt{\pi}} \int_0^t e^{-x^2} dx$ is the Gauss error function.

2.5 Walsh Hadamard Transform (WHT)

Our results in Chapters 3-5 make use of the Walsh Hadamard Transform. This transform helps improving the solving phase of LPN solving algorithms.

Definition 2.7 (Walsh Hadamard Transform)**.** *We write the n-dimensional binary vector $x \in \{0,1\}^n$ by its components $x = (x_1, \dots, x_n)$ with $x_i \in \{0,1\}$. The Walsh Hadamard Transform of a function $f : \{0,1\}^n \to \mathbb{C}$ is a function $\hat{f} : \{0,1\}^n \to \mathbb{C}$ defined as*

$$\hat{f}(\nu) = \sum_x f(x)(-1)^{\sum x_i \nu_i}.$$

The complexity to compute the Walsh Hadamard Transform is $\mathcal{O}(n2^n)$ by running the fast Hadamard transform algorithm [CT65].

2.6 Linear Codes

While presenting the state of the art in the area of LPN solving algorithms in Chapter 3, we introduce a technique called covering code reduction. This method uses linear codes. We present the main results related to linear codes needed to describe the covering code step. In Chapter 5, we improve this reduction by using perfect and quasi-perfect linear codes. Their definition is given below.

Definition 2.8 (Binary Linear Code)**.** *An $[n,k]$ binary linear code C is a linear subspace of the vector space $\mathbb{Z}_2^n$ of dimension k. The elements of C are called codewords. The value n represents the length and k is the dimension of the code.*

In the $[n,k]$ binary linear code C, the codewords can be defined as all possible linear combinations of the rows of a $k \times n$ *generator matrix* G [MS78]. In the systematic form, the matrix G is of the form $G = (I_k|A)$ where I_k is the $k \times k$ identity matrix and A is a $k \times (n-k)$ matrix.

The *Hamming distance* between two codewords x and y, is the number of places where they differ. We denote this by $d(x, y)$. The *minimum distance of a linear code* C is the minimum distance between its codewords [MS78]. We denote this by D. I.e. $D = \min_C d(x, y)$, where $x \neq y$ and $x, y \in C$. An equivalent result is that D is the minimum Hamming weight of any non-zero codeword. A code with minimum distance D can correct $\lfloor \frac{D-1}{2} \rfloor$ errors. A linear code C of length n, dimension k and minimum distance D will be denoted by $[n, k, D]$.

Definition 2.9 (Covering Radius)**.** *For a code C, the covering radius is $\rho = \max_v d(v, C)$, where $d(v, C)$ denotes the Hamming distance of v from the code C.*

Definition 2.10 (Packing Radius)**.** *The packing radius is the largest radius R such that the balls of this radius centered on all codewords are non-overlapping.*

So, the packing radius is $R = \lfloor \frac{D-1}{2} \rfloor$. We further have $\rho \geq \lfloor \frac{D-1}{2} \rfloor$.

Definition 2.11 (Perfect Code)**.** *A* perfect code *is a $[n, k, D]$ code C characterized by $\rho = \lfloor \frac{D-1}{2} \rfloor = R$.*

Examples of perfect codes are: binary Golay code $[23, 12, 7]$, Hamming codes $[2^k, 2^k - k - 1, 3]$ and the repetition codes $[k, 1, k]$.

Definition 2.12 (Quasi-perfect Code)**.** *A* quasi-perfect code *is a $[n, k, D]$ code C characterized by $\rho = \lfloor \frac{D-1}{2} \rfloor + 1 = R + 1$.*

These two classes of codes will be used in the code reduction presented in Section 5.4.

Chapter 3

Solving Algorithms for LPN

In this chapter, we present the existing LPN solving algorithms in a unified manner. We emphasize the main differences between these algorithms and discuss which improvements they bring. We improve on the previous results. The personal contribution in this chapter is a joint work with Florian Tramèr and Serge Vaudenay that was published in the Journal of Cryptography and Communications [BTV16]. Further improvements on this are a joint work with Serge Vaudenay that was published in ASIACRYPT'16 [BV16b]. The presentation of LF(4) is a joint work with Serge Vaudenay published as a note on ePrint [BV16a].

Structure of the Chapter. We give an overview of the existing LPN algorithms in Section 3.1. Following that, in Sections 3.3-3.7, we present step by step the LPN algorithms that have a sub-exponential complexity and require a sub-exponential number of queries. For each of these algorithms, we provide a new analysis that brings improvements on the complexity or corrects the previous results. In Section 3.8, we present algorithms that are efficient in the case of a sparse secret. We describe in Section 3.9 an LPN solving algorithm that works with a polynomial number of queries.

3.1 Previous Work

In the current literature, there are several algorithms to solve the LPN problem. Recall that in LPN, one is asked to recover a secret given noisy inner products of it with randomly chosen vectors. A query is composed of the random vectors and the result of the noisy inner product. Depending on how many queries are given from the LPN oracle, we can split the solving algorithms into three categories.

With a *sub-exponential number of queries*, the existing algorithms solve an LPN instance in sub-exponential time. For this category, we have the BKW [BKW00], LF1, LF2 [LF06], FMICM [FMI+06] and the covering code algorithms [GJL14, ZJW16]. All these algorithms solve $\mathsf{LPN}_{k,\tau}$ with a time complexity of $2^{\mathcal{O}\left(\frac{k}{\log k}\right)}$ and require $2^{\mathcal{O}\left(\frac{k}{\log k}\right)}$

queries.

The first that appeared, and the best known, is the BKW [BKW00] algorithm. This algorithm can be described as a Gaussian elimination on blocks of bits (instead on single bits) where the secret is recovered bit by bit. Improvements of it were presented in [FMI$^+$06, LF06]. One idea that improves the algorithm is the use of the fast Walsh-Hadamard transform as we can recover several bits of the secret at once. In their work [LF06], Levieil and Fouque provide an analysis with the level of security achieved by different LPN instances and propose secure parameters. A more efficient algorithm to solve LPN was presented at ASIACRYPT'14 [GJL14] and it introduces the use of covering codes to improve the performance. The same reduction is used also in the paper from EUROCRYPT'16 [ZJW16] and ASIACRYPT'16 [BV16b] (that will be presented in Chapter 5).

Using BKW as a black-box, Lyubashevsky [Lyu05] introduces a "pre-processing" phase and solves an $\mathsf{LPN}_{k,\tau}$ instance with a *polynomial number of queries* $n = k^{1+\eta}$ and with a time complexity of $2^{\mathcal{O}\left(\frac{k}{\log\log k}\right)}$. The queries given to BKW have a worse bias of $\tau' = \frac{1}{2} - \frac{1}{2}\left(\frac{1-2\tau}{4}\right)^{\frac{2k}{\eta \log k}}$. Thus, this variant requires a polynomial number of queries but has a worse time complexity. When $\tau = \frac{1}{\sqrt{k}}$, we can improve the result of [Lyu05] and have a complexity of $e^{\frac{1}{2}\sqrt{k}(\ln k)^2 + \mathcal{O}(\sqrt{k}\ln k)}$ [BV15]. More details about this are given in Section 6.5.

With a *linear number of queries*, the best algorithms are exponential, i.e. with $n = \Theta(k)$ the secret is recovered in exponential time $2^{\Theta(k)}$ [MMT11, Ste88].

An easy to solve instance of LPN was introduced by Arora and Ge [AG11]. They show that in the k-wise version where the k-tuples of the noise bits can be expressed as the solution of a polynomial (e.g. there are no 5 consecutive errors in the sequence of queries), the problem can be solved in polynomial time. What makes the problem easy is the fact that an adversary is able to structure the noise.

Another easy instance is when the noise is 0, i.e. $\tau = 0$. In this case, the LPN problem is solved in polynomial time through Gaussian elimination given $n = \Theta(k)$ queries. The problem becomes hard once noise is added to the inner product. The value of τ can be either independent or dependent of the value k. Usually, the value of τ is constant and independent from the value of k. A case where τ is taken as a function of k occurs in the construction of the encryption schemes [Ale03, DP12]. Intuitively, a larger value of τ means more noise and makes the problem of search LPN harder. When $\tau = \frac{1}{2}$, the queries from the LPN oracle are uniformly distributed and we cannot recover the secret. The value of the noise parameter is a trade-off between the hardness of the $\mathsf{LPN}_{k,\tau}$ and the practical impact on the applications that rely on this problem.

3.2 Our Contribution

In this chapter, we *present the current existing* LPN *solving algorithms in a unified framework*, where each LPN algorithm is split in two phases: the reduction and the solving phase. This characterization allows us to better compare and see what improvements each algorithm bring.

For these algorithms, we *give a better theoretical analysis* that brings an improvement over the work of Levieil and Fouque [LF06] and Guo et. al. [GJL14]. Furthermore, *we analyse three new algorithms for the case where the secret is sparse.* Our results show that for a sparse secret, the BKW family of algorithms is outperformed by an algorithm that uses Gaussian elimination. Our discussion on LF(4) *gives insights on why this reduction method might not bring improvements over the existing reduction methods.*

While improving the existing results, we hope to provide a theoretical analysis that matches the experimental results. Although this does not prove that LPN is hard, it gives tighter bounds for the parameters used by the LPN-based cryptographic schemes. It can also be used to have a tighter complexity analysis of algorithms related to solving LPN. Our results were actually used in [GJL14] and also for LWE solving [DTV15].

LPN solving algorithms in a nutshell. The common structure of all the aforementioned LPN algorithms is the following: given n queries from the $\mathcal{O}^{\mathsf{LPN}}_{s,\tau}$ oracle, the algorithm tries to reduce the problem of finding a secret s of k bits to one where the secret s' has only k' bits, with $k' < k$. This is done by applying several *reduction* techniques. We call this phase the *reduction phase.* Afterwards, during the *solving phase*, we can apply a *solving* algorithm that recovers the secret s'. We then update the queries with the recovered bits and restart to fully recover s. For the ease of understanding, we describe all the aforementioned LPN solving algorithms in this setting, where we separate the algorithms in two phases.

First, we assume that $k = a \cdot b$. Thus, we can visualise the k-bit length vectors v as a blocks of b bits. Recall that we define $\delta = 1 - 2\tau$. We call δ the *bias of the error bit d.* We have $\delta = E((-1)^d)$, with $E(\cdot)$ the expected value. We denote the bias of the secret bits by δ_s. As s is a uniformly distributed random vector, at the beginning we have $\delta_s = 0$.

3.3 BKW

We start by describing the BKW* algorithm. The notation BKW* is used as we illustrate the algorithm as described in [LF06] which is different from the original BKW [BKW00]. We explain what is the difference and why we use this variant later in this Section. The BKW* works in two phases:

Reduction phase. Given n queries from the LPN oracle, we group them in equivalence classes. Two queries are in the same equivalence class if they have the same value on a set q_1 of b bit positions. These b positions are chosen arbitrarily in $\{1, \ldots, k\}$. There are at most 2^b such equivalence classes. Once this separation is done, we perform the following steps for each equivalence class: pick one query at random, the representative vector, and xor it to the rest of the queries from the same equivalence class. Discard the representative vector. This will give vectors with all bits set to 0 on those b positions. These steps are also illustrated in Algorithm 3.1 (steps 5 - 11). We are left with at least $n - 2^b$ queries where the secret is reduced to $k - b$ effective bits (others being multiplied by 0 in all queries).

We can repeat the reduction technique $a - 1$ times on other disjoint position sets $q_2, \dots, q_{a-1}$ from $\{1, \dots, k\} \backslash q_1$ and end up with at least $n - (a-1)2^b$ queries where the secret is reduced to $k - (a-1)b = b$ bits. The bias of the new queries is $\delta^{2^{a-1}}$, as shown by the following Lemma where we use $w = 2^{a-1}$.

Lemma 3.1 ([LF06, BKW00]). *If $(v_1, c_1), \dots, (v_w, c_w)$ are the results of w queries from $\mathcal{O}^{\mathsf{LPN}}_{s,\tau}$, then the probability that:*

$$\langle v_1 \oplus v_2 \oplus \dots \oplus v_w, s \rangle = c_1 \oplus \dots \oplus c_w$$

is equal to $\frac{1+\delta^w}{2}$.

It is easy so see that the complexity of performing this reduction step is $\mathcal{O}(kan)$.

Algorithm 3.1 BKW* Algorithm by [LF06]

```
1: Input: a set $V$ of $n$ queries $(v_i, c_i) \in \{0,1\}^{k+1}$ from the LPN oracle $\mathcal{O}^{\mathsf{LPN}}_{s,\tau}$, values $a$,
   $b$ such that $k = ab$ and $n \geq a2^b$
2: Output: values $s_1, \dots, s_b$

3: Partition the positions $\{1, \dots, k\} \setminus \{1, \dots, b\}$ into disjoint $q_1 \cup \dots \cup q_{a-1}$ with $q_i$ of
   size $b$
4: for $i = 1$ to $a - 1$ do                                        ▷ Reduction phase
5:     Partition $V = V_1 \cup \dots \cup V_{2^b}$ s.t. vectors in $V_j$ have the the same bit values at
   positions in $q_i$
6:     foreach $V_j$
7:         Choose a random $(v^*, c^*) \in V_j$ as a representative vector
8:         Replace each $(v, c)$ by $(v, c) \oplus (v^*, c^*)$, $(v, c) \in V_j$ for $(v, c) \neq (v^*, c^*)$
9:         Discard $(v^*, c^*)$ from $V_j$
10:    end foreach
11:    $V = V_1 \cup \dots \cup V_{2^b}$
12: end for
13: Discard from $V$ all queries $(v, c)$ such that $\mathsf{HW}(v) \neq 1$
14: Partition $V = V_1 \cup \dots \cup V_b$ s.t. vectors in $V_j$ have a bit 1 on position $j$
15: foreach position $i$                                            ▷ Solving phase
16:     $s_i = \mathsf{majority}(c)$, for all $(v, c) \in V_i$
17: end foreach
18: return $s_1, \dots, s_b$
```

After $a - 1$ iterations, we are left with at least $n - (a-1)2^b$ queries, and a secret of size of b effective bits at positions $1, \dots, b$. The goal is to keep only the queries that have Hamming weight 1 (step 13 of Algorithm 3.1). Given $n-(a-1)2^b$ queries and one bit position $j \in \{1,\dots,k\}\backslash\{q_1 \cup \dots \cup q_{a-1}\}$, only $n' = \frac{n-(a-1)2^b}{2^b}$ will have a single non-zero bit on position j and 0 on all the others. These queries represent the input to the solving phase. The bias does not change since we do not alter the original queries. The complexity for performing this step for $n-(a-1)2^b$ queries is $\mathcal{O}(b(n-(a-1)2^b))$ as the algorithm just checks if the queries have Hamming weight 1.

The bit c is part of the query also: it gets updated during the xoring operations but we do not consider this bit in partitioning or when computing the Hamming weight of a query. Later on, the information stored in this bit will be used to recover bits of the secret.

Remark 1. *Given that we have performed the xor between pairs of queries, we note that the noise bits are no longer independent. In the analysis of* BKW**, this was overlooked by Levieil and Fouque [LF06].*[1] *The original* BKW *[BKW00] algorithm overcomes this problem in the following manner: each query that has Hamming weight* 1 *is obtained with a fresh set of queries. Given* $a2^b$ *queries, the algorithm runs the xoring process and is left with* 2^b *vectors. From these* 2^b *queries, with a probability of* $1-(1-2^{-b})^{2^b} \approx 1-\frac{1}{e}$*, where* $e = 2.718$*, there is one with Hamming weight* 1 *on a given position* i*. In order to obtain more such queries the algorithm repeats this process with fresh queries. This means that for guessing one bit of the secret, the original algorithm requires* $n = a \cdot 2^b \cdot \frac{1}{1-1/e} \cdot n'$ *queries, where* n' *denotes the number of queries needed for the solving phase. This is larger than* $n = 2^b n' + (a-1)2^b$ *which is the number of queries given by Levieil and Fouque [LF06]. We implemented and ran* BKW* *as described in Algorithm 3.1 and we discovered that this dependency does not affect the performance of the algorithm. I.e., the number of queries computed by the theory that ignores the dependency of the error bits matches the practical results. The theoretical and practical results are presented in the next chapter, in Section 4.2. Given our practical experiments, we keep the "heuristic" assumption of independence and the algorithm as described in [LF06] which we called* BKW**. Thus, we assume from now on the independence of the noise bits and the independence of the queries.*

Another discussion on the independence of the noise bits is presented in [Fit14]. There, we can see what is the probability to have a collision, i.e. two queries that share an error bit, among the queries formed during the xoring steps.

We can repeat the algorithm a times, with the same queries, to recover all the k bits. The total time complexity for the reduction phase is $\mathcal{O}(ka^2n)$, as we perform the steps described above a times (instead of $\mathcal{O}(kan)$ as given in [LF06]). However, by making the selection of a and b adaptive with ab near to the remaining number of bits to recover, we can show that the total complexity is dominated by the one of recovering the first block. So, we can typically concentrate on the algorithm to recover a single block. We provide a more complete analysis in Section 3.4.

[1] Definition 2 of [LF06] assumes independence of samples but Lemma 2 of [LF06] shows the reduction without proving independence.

Solving phase. The BKW solving method recovers the 1-bit secret by applying the majority rule. The queries from the reduction phase are of the form $c'_j = s_i \oplus d'_j$, $d'_j \leftarrow \mathsf{Ber}_{(1-\delta^{2^{a-1}})/2}$ and s_i being the i^{th} bit of the secret s. Given that the probability for the noise bit to be set to 1 is smaller than $\frac{1}{2}$, in more than half of the cases, these queries will be s_i. Thus, we decide that the value of s_i is given by the majority rule (steps 14-16 of Algorithm 3.1). By applying the Hoeffding bounds [Hoe63], we find how many queries are needed, such that the probability of guessing incorrectly one bit of the secret is bounded by some constant θ, with $0 < \theta < 1$.

The time complexity of performing the majority rule is linear in the number of queries.

Complexity analysis. With their analysis, Levieil and Fouque [LF06] obtain the following result:

Theorem 3.2 (Th. 1 from [LF06])**.** *For $k = a \cdot b$, the* BKW^* *algorithm heuristically $(n = 20 \cdot \ln(4k) \cdot 2^b \cdot \delta^{-2^a}, t = \mathcal{O}(kan), m = kn, \theta = \frac{1}{2}, b)$-solves the* LPN *problem.*

New Analysis on BKW^*. We improve the number of queries needed for the majority rule. Recall that the queries are of the form $c'_j = s_i \oplus d'_j$, $d'_j \leftarrow \mathsf{Ber}_{(1-\delta')/2}$. The majority of these queries will most likely be $c'_j = s_i$. It is intuitive to see that the majority rule fails when more than half of the noise bits are 1 for a given bit. Any wrong guess of a bit gives a wrong value of the k-bit secret s. In order to bound the probability of such a scenario, we could use the Hoeffding bounds [Hoe63] with $X_j = d_j$ (See Section 2.3). We have $\Pr[X_j = 1] = \frac{1-\delta'}{2}$. For $X = \sum_{j=1}^{n'} X_j$, we have $E(X) = \frac{(1-\delta')n'}{2}$ and we apply Theorem 2.5 with $\lambda = \frac{\delta n'}{2}$, $\alpha_j = 0$ and $\beta_j = 1$ and we obtain

$$\Pr[\mathsf{incorrect\ guess\ on}\ s_i] = \Pr\left[X \geq \frac{n'}{2}\right] \leq e^{-\frac{n'\delta'^2}{2}}.$$

As discussed in Remark 1, the assumption of independence is heuristic.

Using the above results for every bit $1, \ldots, b$, we can bound by a constant θ, the probability that we guess incorrectly a block of s, with $0 < \theta < 1$. Using the union bound, we get that $n' = 2\delta'^{-2}\ln(\frac{b}{\theta})$. Given that $n' = \frac{n-(a-1)2^b}{2^b}$ and that $\delta' = \delta^{2^{a-1}}$, we obtain that we need $n = 2^{b+1}\delta^{-2^a}\ln(\frac{b}{\theta}) + (a-1)2^b$ queries in total. This is the result presented by Bogos et al. [BTV16]. It was further improved with the use of Central Limit Theorem (CLT) in ASIACRYPT'16 [BV16b]. With $X_j = d_j$ and $X = \sum_{j=1}^{n'} X_j$ we have that $E(X_1) = \frac{1-\delta'}{2}$ and $\mathsf{Var}(X_1) = E(X_1^2) - E(X_1)^2 = \frac{1-\delta'^2}{4}$. If we apply Theorem 2.6, we obtain

$$\Pr[\mathsf{incorrect\ guess\ on}\ s_i] = \Pr\left[X \geq \frac{n'}{2}\right] \approx 1 - \varphi\left(\frac{\sqrt{n'}\delta'}{\sqrt{1-\delta'^2}}\right).$$

and that $n' \geq \frac{1-\delta'^2}{\delta'^2}\left(\varphi^{-1}\left(1 - \frac{\theta}{b}\right)\right)^2$. With this new bound, we obtain the following result.

Table 3.1: BKW* query complexity ($\log_2$) - our theory (Th. 3.3)

τ	k				
	32	48	64	80	100
0.01	9.71	11.69	14.84	17.67	20.80
0.10	14.73	19.02	23.19	27.32	32.43
0.20	18.70	23.93	30.11	33.97	39.09
0.25	20.81	26.05	32.23	37.33	43.30
0.40	27.33	35.56	42.80	47.91	55.02

Table 3.2: BKW* query complexity ($\log_2$) - original theory (Th. 3.2)

τ	k				
	32	48	64	80	100
0.01	14.56	16.60	19.68	22.59	25.64
0.10	19.75	23.87	27.95	32.00	37.06
0.20	23.50	28.61	34.69	38.64	43.70
0.25	25.60	30.72	36.79	41.85	47.90
0.40	31.89	40.00	47.37	52.43	59.48

Theorem 3.3. *For $k \leq a{\cdot}b$, the* BKW* *algorithm heuristically* ($n = 2^b \frac{1-\delta^{2^a}}{\delta^{2^a}} \left(\varphi^{-1}\left(1-\frac{\theta}{b}\right)\right)^2 + (a-1)2^b, t = \mathcal{O}(kan), m = kn, \theta, b$)*-solves the* LPN *problem.*[2]

We note that we obtained the above result using the union bound. One could make use of the independence of the noise bits and obtain $n = 2^b \frac{1-\delta^{2^a}}{\delta^{2^a}} \left(\varphi^{-1}\left((1-\theta)^{\frac{1}{b}}\right)\right)^2 + (a-1)2^b$, but this would bring a very small improvement.

In terms of query complexity, we compare our theoretical results (i.e. Theorem 3.3) with the ones from [LF06] (i.e. Theorem 3.2) in Table 3.1 and Table 3.2. We provide the $\log_2(n)$ values for k varying from 32 to 100 and we take different Bernoulli noise parameters that vary from 0.01 to 0.4. Overall, our theoretical results bring an improvement of a factor 20 over the results of [LF06].

We note that our BKW* algorithm, for which we have stated the above theorem, follows the steps from Algorithm 3.1 for $k = a \cdot b$. For $k < a \cdot b$ the algorithm is a bit different. In this case, we have $a-1$ blocks of size b and an incomplete block of size smaller than b. During the reduction phase, we first partition the incomplete block and then apply $(a-2)$ reduction steps for the complete blocks. We finally have b bits to recover. Other than this small change, the algorithm remains the same.

3.4 LF1

During the solving phase, the BKW algorithm recovers the value of the secret bit by bit. Given that we are interested only in queries with Hamming weight 1, many queries are discarded at the end of the reduction phase. As first noted in [LF06], this can be improved by using a Walsh-Hadamard transform instead of the majority rule. This

[2]The term $(a-1)2^b$ is not included in Theorem 3.2 (Theorem 1 from [LF06]). This factor represents the number of queries lost during the reduction phase.

improvement of BKW is denoted in [LF06] by LF1. Again, we present the algorithm in pseudo-code in Algorithm 3.2. As in BKW*, we can concentrate on the complexity to recover the first block.

Reduction phase. The reduction phase for LF1 follows the same steps as in BKW* in obtaining new queries as 2^{a-1} xors of initial queries in order to reduce the secret to size b. At this step, the algorithm does not discard queries anymore but proceeds directly with the solving phase (see Steps 3-11 of Algorithm 3.2). We now have $n' = n-(a-1)2^b$ queries after this phase.

Solving phase. The solving phase consists in applying a Walsh-Hadamard transform in order to recover b bits of the secret at once (Steps 13-15 in Algorithm 3.2). We can recover the b-bit secret by computing the Walsh-Hadamard transform of the function $f(x) = \sum_i 1_{v'_i = x}(-1)^{c'_i}$, where (v'_i, c'_i) are the queries after the reduction phase. Writing the LPN instance as $A's^T \oplus d' = c'$, where A' contains the v'_i vectors, the Walsh-Hadamard transform is

$$\begin{aligned}\hat{f}(\nu) &= \sum_x (-1)^{\langle \nu, x\rangle} f(x) = \sum_x (-1)^{\langle \nu, x\rangle} \sum_i 1_{v'_i = x}(-1)^{c'_i} \\ &= \sum_i (-1)^{\langle v'_i, \nu\rangle + c'_i} = n' - 2\mathsf{HW}(A'\nu^T + c').\end{aligned}$$

For $\nu = s$, the correct secret, we have $\hat{f}(s) = n' - 2 \cdot \mathsf{HW}(d')$, where d' represents the noise vector after the reduction phase. We know that most of the noise bits are set to 0. So, $\hat{f}(s)$ is large and we suppose it is the largest value in the table of $\hat{f}$. Thus, we have to look at the maximum value of the Walsh-Hadamard transform in order to recover the value of s. A naive implementation of a Walsh-Hadamard transform would give a complexity of 2^{2b} since we apply it on a space of size 2^b. Since we apply a fast Walsh-Hadamard transform, we get a time complexity of $b2^b$ [CT65].

Complexity analysis. The following theorem states the complexity of LF1:

Theorem 3.4 (Th. 2 from [LF06]). *For $k = a \cdot b$ and $a > 1$, the* LF1 *algorithm heuristically $(n = (8b+200)\delta^{-2^a} + (a-1)2^b, t = \mathcal{O}(kan + b2^b), m = kn + b2^b, \theta = \frac{1}{2}, b)$-solves the* LPN *problem.*[3]

BKW* vs. LF1. We can see that compared to BKW*, LF1 brings a significant improvement in the number of queries needed. As expected, the factor 2^b disappeared as we did not discard any query at the end of the reduction phase. There is an increase in the

[3]The term $b2^b$ in the time complexity is missing in [LF06]. While in general kan is the dominant term, in the special case where $a = 1$ (thus we apply no reduction step) a complexity of $\mathcal{O}(kan)$ would be wrong since, in this case, we apply the Walsh-Hadamard transform on the whole secret and the term $k2^k$ dominates the final complexity.

Algorithm 3.2 LF1 Algorithm

```
1:  Input: a set $V$ of $n$ queries $(v_i, c_i) \in \{0,1\}^{k+1}$ from the LPN oracle, values $a$, $b$ such
    that $k = ab$
2:  Output: values $s_1, \dots, s_b$

3:  Partition the positions $\{1, \dots, k\} \setminus \{1, \dots, b\}$ into disjoint $q_1 \cup \dots \cup q_{a-1}$ with $q_i$ of
    size $b$
4:  for $i = 1$ to $a - 1$ do                                              ▷ Reduction phase
5:      Partition $V = V_1 \cup \dots \cup V_{2^b}$ s.t. vectors in $V_j$ have the the same bit values at
    positions in $q_i$
6:      foreach $V_j$
7:          Choose a random $(v^*, c^*) \in V_j$ as a representative vector
8:          Replace each $(v, c)$ by $(v, c) \oplus (v^*, c^*)$, $(v, c) \in V_j$ for $(v, c) \neq (v^*, c^*)$
9:          Discard $(v^*, c^*)$ from $V_j$
10:     end foreach
11:     $V = V_1 \cup \dots \cup V_{2^b}$
12: end for
13: $f(x) = \sum_{(v,c)\in V} 1_{v_{1,\dots,b}=x}(-1)^c$                  ▷ Solving phase
14: $\hat{f}(\nu) = \sum_x (-1)^{\langle \nu, x\rangle} f(x)$                ▷ Walsh transform of $f(x)$
15: $(s_1, \dots, s_b) = \mathsf{arg\ max}(\hat{f}(\nu))$
16: return $s_1, \dots, s_b$
```

time and memory complexity because of the fast Walsh-Hadamard transform, but these terms are not the dominant ones.

New Analysis on LF1. We improve the analysis done on the number of queries needed for the Walsh transform. The failure probability for LF1 is bounded by the probability that there is another vector $\nu \neq s$ such that that $\hat{f}(\nu) > \hat{f}(s)$. This is equivalent to $\mathsf{HW}(A'\nu^T + c') \leq \mathsf{HW}(A's^T + c')$. Recall that $A's^T + c' = d'$. We define $x = s + \nu$ so that $A'\nu^T + c' = A'x^T + d'$. As A' is uniformly distributed, independent from d', and x is fixed and non-zero, $A'x^T + d'$ is uniformly distributed, so we can rewrite the inequality as $\mathsf{HW}(y) \leq \mathsf{HW}(d')$, for a random y.

Using again the Hoeffding's bounds, we need to have $n' = 8\ln(\frac{2^{k'}}{\theta})\delta'^{-2}$ [BTV16] queries in order to bound the probability of guessing wrongly the k'-bit secret by θ. We can improve further by applying directly the Central Limit Theorem. Let $X_1, X_2, \dots, X_{n'}$ be i.i.d random independent variables with $X_j = y_j - d'_j$, $\Pr(X_j \in [-1, 1]) = 1$ and $X = \sum_{i=0}^{n'} X_i$. We have $E(X_1) = \frac{\delta'}{2}$ and $\mathsf{Var}(X_1) = \frac{2-\delta'^2}{4}$. Using Theorem 2.6 we obtain:

$$\Pr[X \leq 0] \approx \varphi\left(\frac{\sqrt{n'}E(X_1)}{\sqrt{\mathsf{Var}(X_1)}}\right).$$

We can bound the probability of incorrectly guessing one block of s by θ to have

$$\Pr[\text{incorrect guess on one block}] \approx 1 - (1 - \Pr[X \leq 0])^{2^{k'}-1} \leq \theta$$

and obtain that we need to have $n' \geq \frac{2-\delta'^2}{\delta'^2}\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^{k'}-1}}\right)$. We can derive the approximation of Selçuk [Sel08] that $n' \geq 4\ln(\frac{2^{k'}}{\theta})\delta'^{-2}$.

The total number of queries will be $n = n' + (a-1)2^b$ and we have $\delta' = \delta^{2^{a-1}}$, $k' = b$. Complexity of the $\mathsf{WHT}(k')$ is $\mathcal{O}(k'2^{k'}\frac{\log_2 n'+1}{2} + k'n')$ as we use the fast Walsh Hadamard Transform[4,5]. Similar to BKW, we obtain the following theorem:

Theorem 3.5. *For $k \leq a\cdot b$ and $n' = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^b-1}}\right)\right)^2$, the* $\mathsf{LF1}$ *algorithm heuristically ($n = n' + (a-1)2^b, t = \mathcal{O}(kan + b2^b\frac{\log_2 n'+1}{2} + bn'), m = kn + b2^b, \theta, b$)-solves the* LPN *problem.*

By comparing the term $(8b+200)\delta^{-2^a}$ in Theorem 3.4 with our value of $\frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^b-1}}\right)\right)^2$, one might check that our term is roughly a factor 4 smaller than that of [LF06] for practical values of a and b. For example, for a $\mathsf{LPN}_{768,0.01}$ instance (with $a = 11$, $b = 70$), our analysis requires 2^{67} queries for the solving phase while the Levieil and Fouque analysis requires 2^{69} queries.

Recovery of the first block dominates the complexity. For BKW^* and $\mathsf{LF1}$ we stated their complexity in terms of recovering one block of the secret. We show how the cost of solving one block of the secret dominates the total cost of recovering s. The main intuition is that after recovering a first block of k' secret bits, we can apply a simple back substitution mechanism and consider solving a $\mathsf{LPN}_{k-k',\tau}$ problem. The same strategy is applied by [ACF+15, DTV15] when solving LWE. Note that this is simply a generalisation of the classic Gaussian elimination procedure for solving linear systems, where we work over blocks of bits.

Specifically, let $k_1 = k$ and $k_i = k_{i-1} - k'_{i-1}$ for $i > 1$ and $k'_{i-1} < k_{i-1}$. Now, suppose we were able to $(n_i, t_i, m_i, \theta_i, k'_i)$-solve an $\mathsf{LPN}_{k_i,\tau}$ instance (meaning we recover a block of size k'_i from the secret of size k_i with probability θ_i, in time t_i and with memory m_i). One can see that for $k_{i+1} < k_i$ we need less queries to solve the new instance (the number of queries is dependent on the size k_{i+1} and on the noise level). With a smaller secret, the time complexity will decrease. Having a shorter secret and less queries, the memory needed is also smaller. Then, we can (n, t, m, θ, k)-solve the problem $\mathsf{LPN}_{k,\tau}$ (i.e recover s completely), with $n = \max(n_1, n_2, \ldots)$, $\theta = \theta_1 + \theta_2 + \ldots$, $t = t_1 + k'_1 n_1 + t_2 + k'_2 n_2 \ldots$ (the

[4]The second term $k'n'$ illustrates the cost of constructing the function f. In cases where $n' > 2^{k'}$ this is the dominant term and it should not be ignored. This was missing in several works [GJL14, BTV16]. For the instance $\mathsf{LPN}_{592,0.125}$ from Guo et al. [GJL14] this makes a big difference as $k' = 64$ and $n' = 2^{69}$; the complexity of WHT with the second term is 2^{75} vs 2^{70} [GJL14]. Given that is must be repeated 2^{13} (as 35 bits of the secret are guessed), the cost of WHT is 2^{88}.

[5]Normally, the values $\hat{f}(\nu)$ have an order of magnitude of $\sqrt{n'}$ so we have $\frac{1}{2}\log_2 n'$ bits.

terms $k'_i n_i$ are due to query updates by back substitution) and $m = \max(m_1, m_2, \ldots)$. Finally, by taking $\theta_i = 3^{-i}$, we obtain $\theta \le \frac{1}{2}$ and thus recover the full secret s with over 50% probability.

It is easily verified that for all the algorithms we consider, we have $n = n_1$, $m = m_1$, and t is dominated by t_1. We provide an example on a concrete LPN instance in Appendix A.1.

3.5 LF2

LF2 is a heuristic algorithm, also introduced in [LF06], that applies the same Walsh-Hadamard transform as LF1, but has a different reduction phase. We provide the pseudocode for LF2 in Algorithm 3.3.

Algorithm 3.3 LF2 Algorithm

```
1: **Input**: a set $V$ of $n$ queries $(v_i, c_i) \in \{0,1\}^{k+1}$ from the LPN oracle, values $a$, $b$ such
   that $k = ab$
2: **Output**: values $s_1, \ldots, s_b$

3: Partition the positions $\{1, \ldots, k\} \setminus \{1, \ldots, b\}$ into disjoint $q_1 \cup \ldots \cup q_{a-1}$ with $q_i$ of
   size $b$
4: for $i = 1$ to $a - 1$ do                                        ▷ Reduction phase
5:     Partition $V = V_1 \cup \ldots \cup V_{2^b}$ s.t. vectors in $V_j$ have the the same bit values at
       positions in $q_i$
6:     foreach $V_j$
7:         $V'_j = \emptyset$
8:         foreach pair $(v, c), (v', c') \in V_j$, $(v, c) \neq (v', c')$
9:             $V'_i = V'_i \cup (v \oplus v', c \oplus c')$
10:        end foreach
11:    end foreach
12:    $V = V'_1 \cup \ldots \cup V'_{2^b}$
13: end for
14: $f(x) = \sum_{(v,c) \in V} 1_{v_{1,\ldots,b} = x} (-1)^c$                          ▷ Solving phase
15: $\hat{f}(\nu) = \sum_x (-1)^{\langle \nu, x \rangle} f(x)$            ▷ compute the Walsh transform of $f(x)$
16: $(s_1, \ldots, s_b) = \text{arg max}(\hat{f}(\nu))$
17: return $s_1, \ldots, s_b$
```

Reduction phase. Similarly to BKW* and LF1, the n queries are grouped into equivalence classes. Two queries are in the same equivalence class if they have the same value on a window of b bits. In each equivalence class, we perform the xor of all the pairs from that class. Thus, we do not choose one representative vector that is discarded afterwards.

Solving phase. This phase works like in LF1, i.e. we use the Walsh-Hadamard transform and find its maximum in order to recover a block of the secret s.

New Analysis on LF2. There is no formal result for LF2 stated in [LF06]. Using the new bounds for LF1, we can state the number of queries needed for the solving phase. For the reduction phase, we can analyse how the number of queries fluctuate. When we do the xor of all the pairs from each class, the expected new number of queries is $E(\sum_{i<j} 1_{v_i \text{ matches } v_j \text{ on the } b\text{-bit block}}) = \frac{n(n-1)}{2^{b+1}}$ which improves previous results [BTV16][6]. When $n \approx 1 + 2^{b+1}$, the number of queries are maintained. For $n > 1 + 2^{b+1}$, the number of queries will increase.

With this analysis, we have the following result.

Theorem 3.6. *For $k \leq a{\cdot}b$, $n' = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^b-1}}\right)\right)^2$ and $n = 1+2^{b+1} \geq n'$, the* LF2 *algorithm heuristically $(n, t = \mathcal{O}(kan + b2^{b\frac{\log_2 n'+1}{2}} + bn'), m = kn + b2^b, \theta, b)$-solves the* LPN *problem.*

One can observe that we may allow n to be smaller than $1 + 2^{b+1}$. Given that the solving phase may require less than $1 + 2^{b+1}$, we could start with less queries, decrease the number of queries during the reduction and end up with the exact number of queries needed for the solving phase.

In a scenario where the attacker has access to a restricted number of queries, this heuristic algorithm helps in increasing the number of queries. With LF2, the attacker might produce enough queries to recover the secret s.

3.6 Covering Code Algorithm

The new algorithm [GJL14] that was presented at ASIACRYPT'14 (and received the best paper award), introduces a new type of reduction. There is a difference between [GJL14] and what was presented at the ASIACRYPT conference. We concentrate here on [GJL14] and on the suggestions we provided to the authors.

Reduction phase. The first step of this algorithm is to transform the LPN instance where the secret s is randomly chosen to an instance where the secret has now a Bernoulli distribution. This method was described in [Kir11, ACPS09, BL12].

Given n queries from the LPN oracle: $(\bar{v_1}, c_1)$, $(\bar{v_2}, c_2), \ldots, (\bar{v_n}, c_n)$, select k linearly independent vectors $\bar{v}_{i_1}, \ldots, \bar{v}_{i_k}$. Construct the $k \times k$ target matrix M that has on its columns the aforementioned vectors, i.e. $M = [\bar{v}_{i_1}^T \bar{v}_{i_2}^T \ldots \bar{v}_{i_k}^T]$. Compute $(M^T)^{-1}$ the inverse of M^T, where M^T is the transpose of M. We can rewrite the k queries corresponding to the selected vectors as $M^T s^T + d'$, where d' is the k-bit column vector

[6]In Bogos et al. [BTV16], the approximation for the number of queries was $\frac{\frac{n}{2^b}\left(\frac{n}{2^b}-1\right)}{2}$, which is less favourable.

$d' = (d_{i_1}, d_{i_2}, \ldots, d_{i_k})^T$. We denote $c' = M^T s^T + d'$. For any $\bar{v}_j$ that is not used in matrix M do the following computation:

$$\bar{v}_j (M^T)^{-1} c' + c_j = \langle \bar{v}_j (M^T)^{-1}, d' \rangle + d_j.$$

We discard the matrix M. From the initial set of queries, we have obtained a new set where the secret value is d'. This can be seen as a reduction to a sparse secret. The complexity of this transform is $\mathcal{O}(k^3 + nk^2)$ by the schoolbook matrix inversion algorithm. This can be improved as follows: for a fixed χ, one can split the matrix $(M^T)^{-1}$ in $a' = \lceil \frac{k}{\chi} \rceil$ parts $\begin{bmatrix} M_1 \\ M_2 \\ \cdots \\ M_{a'} \end{bmatrix}$ of χ rows. By pre-computing $\bar{v} M_i$ for all $\bar{v} \in \{0,1\}^\chi$, the operation of performing $\bar{v}_j (M^T)^{-1}$ takes $\mathcal{O}(ka')$. The pre-computation takes $\mathcal{O}(2^\chi)$ and is negligible if the memory required by the BKW reduction is bigger. With this pre-computation the complexity is $\mathcal{O}(nka')$.

Afterwards, the algorithm follows the usual BKW reduction steps where the size of the secret is reduced to k' by the xoring operation. Again, the vector of k bits is seen as being split into blocks of size b. The BKW reduction is applied a times. Thus, we have $k' = k - ab$.

The secret s of k' bits is split into 2 parts: one part denoted s_2 of k'' bits and the other part, denoted s_1, of $k' - k''$ bits. The next step in the reduction is to guess value of s_1 by making an assumption on its Hamming weight: $\mathsf{HW}(s_1) \leq w_0$. The remaining queries are of the form $(v_i, c_i = \langle v_i, s_2 \rangle \oplus d_i)$, where $v_i, s_2 \in \{0,1\}^{k''}$ and $d_i \in \mathsf{Ber}_{\frac{1-\delta^{2^a}}{2}}$. Thus, the problem is reduced to a secret of k'' bits.

At this point, the algorithm approximates the v_i vectors to the nearest codeword g_i in a $[k'', \ell]$ linear code where k'' is the size and ℓ is the dimension. By observing that g_i can be written as $g_i = g'_i G$, where G is the generating matrix of the code, we can write the equations in the form

$$c_i = \langle v_i, s_2 \rangle \oplus d_i = \langle g'_i G, s_2 \rangle \oplus \langle v_i - g_i, s_2 \rangle \oplus d_i = \langle g'_i, s'_2 \rangle \oplus d'_i$$

with $s'_2 = s_2 G^T$ and $d'_i = \langle v_i - g_i, s_2 \rangle \oplus d_i$, where g'_i, s'_2 have length ℓ. If the code has a covering radius of ρ, $v_i - g_i$ is a random vector of weight bounded by ρ, while s_2 is a vector of some small weight bounded by w_1, with some probability. So, $\langle v_i - g_i, s_2 \rangle$ is biased and we can analyse d'_i in place of d_i.

In [GJL14], the authors approximate the bias of $\langle v_i - g_i, s_2 \rangle$ to $\delta' = \left(1 - 2\frac{\rho}{k''}\right)^{w_1}$, as if all bits were independent. As discussed later, this approximation is far from good.

No query is lost during this covering code operation and now the secret is reduced to ℓ bits. We now have $n' = n - k - a2^b$ queries at the end of the reduction phase.

Solving phase. The solving phase of this algorithm follows the same steps as LF1, i.e. it employs a fast Walsh-Hadamard transform. One should notice that the solving phase recovers ℓ linear relations between the bits of the secret and not actual bits of the secret.

Complexity analysis. Recall that in the algorithm two assumptions are made regarding the Hamming weight of the secret: that s_2 has a Hamming weight bounded by w_1 and that s_1 has a Hamming weight bounded by w_0. This holds with probability $\Pr(w_0, k' - k'') \cdot \Pr(w_1, k'')$ where

$$\Pr(w, m) = \sum_{i=0}^{w} (1-\tau)^{m-i} \tau^i \binom{m}{i}. \tag{3.1}$$

The total complexity is given by the complexity of one iteration to which we add the number of times we have to repeat the iteration. We state below the result from [GJL14]:

Theorem 3.7 (Th 1. from [GJL14]). [7]
Let n be the number of samples required and $a, a', b, w_0, w_1, \ell, k', k''$ be the algorithm parameters. For the $\mathsf{LPN}_{k,\tau}$ instance, the number of bit operations required for a successful run of the new attack is equal to

$$t = \frac{t_{\textsf{sparse reduction}} + t_{\textsf{bkw reduction}} + t_{\textsf{guess}} + t_{\textsf{covering code}} + t_{\textsf{Walsh transform}}}{\Pr(w_0, k' - k'')\Pr(w_1, k'')},$$

where

- $t_{\textsf{sparse reduction}} = nka'$ *is the cost of reducing the* LPN *instance to a sparse secret*
- $t_{\textsf{bkw reduction}} = (k+1)an$ *is the cost of the* BKW *reduction steps*
- $t_{\textsf{guess}} = n' \sum_{i=0}^{w_0} \binom{k'-k''}{i} i$ *is the cost of guessing $k' - k''$ bits and $n' = n - k - a2^b$ represents the number of queries at the end of the reduction phase*
- $t_{\textsf{covering code}} = (k'' - \ell)(2n' + 2^\ell)$ *is the cost of the covering code reduction and n' is again the number of queries*
- $t_{\textsf{Walsh transform}} = \ell 2^\ell \sum_{i=0}^{w_0} \binom{k'-k''}{i}$ *is the cost of applying the fast Walsh-Hadamard transform for every guess of $k' - k''$ bits*

under the condition that $n - a2^b > \frac{1}{\delta^{2^{a+1}} \cdot \delta'^2}$, where $\delta = 1 - 2\tau$ and $\delta' = \left(1 - 2\frac{\rho}{k''}\right)^{w_1}$ and ρ is the smallest integer, s.t. $\sum_{i=0}^{\rho} \binom{k''}{i} > 2^{k''-\ell}$.

The condition $n - a2^b > \frac{1}{\delta^{2^{a+1}} \cdot \delta'^2}$ proposed in [GJL14] imposes a lower bound on the number of queries needed in the solving phase for the fast Walsh-Hadamard transform. In our analysis, we will see that this is underestimated: the Central Limit Theorem dictates a larger number of queries.

[7]This theorem includes imprecisions. It is updated in Theorem 3.8.

New Analysis on Covering Code. We describe the shortcomings of the covering code algorithm when computing the bias introduced by the covering code method. We present a solution to this that was adopted also by the authors of this algorithm [GJL14].

Recall that the algorithm first reduces the size of the secret to k'' bits by running BKW reduction steps. Then it approximates the v_i vector to the nearest codeword g_i in a $[k'', \ell]$ linear code with G as generator matrix. The noisy inner products can be rewritten as

$$c_i = \langle g_i' G, s_2 \rangle \oplus \langle v_i - g_i, s_2 \rangle \oplus d_i = \langle g_i', s_2 G^T \rangle \oplus d_i' = \langle g_i', s_2' \rangle \oplus d_i',$$

where g_i is such that $g_i = g_i' G$, $s_2' = s_2 G^T$ and $d_i' = \langle g_i - v_i, s_2 \rangle \oplus d_i$.

Given that the code has a covering radius of ρ and that the Hamming weight of s_2 is bounded by w_1, the bias of $\langle g_i - v_i, s_2 \rangle$ is computed as $\delta' = \left(1 - 2\frac{\rho}{k''}\right)^{w_1}$ in ASIACRYPT'14 [GJL14], where k'' is the size of s_2. We stress that this approximation is far from good.

Indeed, with the $[3, 1]$ repetition code given as an example in their paper [GJL14], the xor of two error bits is unbiased. Even worse: the xor of the three bits has a negative bias. So, when using the code obtained by 25 concatenations of this repetition code and $w_1 = 6$, with some probability of 36%, we have at least two error bits falling in the same concatenation and the bias makes this approach fail.

We can do the same computation with the concatenation of five $[23, 12]$ Golay codes with $w_1 = 15$, as suggested in [GJL14]. With probability 0.21%, the bias is zero or negative so the algorithm fails. With probability 8.3%, the bias is too low.

In any case, we cannot assume the error bits to be independent. When the code has optimal covering radius ρ, we can actually find an explicit formula for the bias of $\langle v_i - g_i, s_2 \rangle$ assuming that s_2 has weight w_1:

$$\Pr[\langle v_i - g_i, s_2 \rangle = 1 | \mathsf{HW}(s_2) = w_1] = \frac{1}{S(k'', \rho)} \sum_{i \leq \rho, i \text{ odd}} \binom{w_1}{i} S(k'' - w_1, \rho - i) \quad (3.2)$$

where $S(k'', \rho)$ is the number of k''-bit strings with weight at most ρ.

To solve $\mathsf{LPN}_{512,0.125}$, the authors of ASIACRYPT'14 [GJL14] propose the following parameters

$$a = 6 \quad a' = 9 \quad b = 63 \quad \ell = 64 \quad k'' = 124 \quad w_0 = 2 \quad w_1 = 16$$

and obtain $n = 2^{66.3}$ and a complexity of $2^{79.92}$. With these parameters, the paper [GJL14] approximated the bias to $\left(1 - 2\frac{\rho}{k''}\right)^{w_1} = 2^{-5.91}$ (with $\rho = 14$). With our exact formula, the bias should rather be of $2^{-7.05}$. So, n should be multiplied by 4.82 (the square of the ratio).

Also, we stress that all this assumes the construction of a code with optimal radius coverage, such as the Golay codes, or the repetition codes of odd length and dimension 1. But these codes do not exist for all $[k'', \ell]$. If we use concatenations of repetition codes, given as an example in [GJL14], the formula for the bias changes. Given ℓ concatenations of the $[k_i, 1]$ repetition code, with $k_1 + \cdots + k_\ell = k''$, $k_i \approx \frac{k''}{\ell}$ and $1 \le i \le \ell$, we would have to split the secret s_2 in chunks of $k_1, \ldots, k_\ell$ bits. We take $w_{11} + \cdots + w_{1\ell} = w_1$ where w_{1i} is the weight of s_2 on the i^{th} chunk. In this case the bias for each repetition code is

$$\delta_i = 1 - 2 \times \frac{1}{S(k_i, \rho_i)} \sum_{j \le \rho_i, j \text{ odd}} \binom{w_{1i}}{j} S(k_i - w_{1i}, \rho_i - j), \tag{3.3}$$

where $\rho_i = \lfloor \frac{k_i}{2} \rfloor$.

The final bias is

$$\delta' = \delta_1 \cdots \delta_\ell. \tag{3.4}$$

We emphasize that the value of n is underestimated in [GJL14]. Indeed, with $n' = \mathsf{bias}^{-2}$, the probability that $\arg\max(\hat{f}(\nu)) = s'_2$ is too low in $\mathsf{LF1}$. To have a constant probability of success θ, our analysis says that we should multiply n' by $4\ln(\frac{2^\ell}{\theta})$ if we apply the Hoeffding's bound. We use below Central Limit Theorem to define the value of n'. For $\mathsf{LPN}_{512,0.125}$ with the parameters presented above and $\theta = \frac{1}{3}$, this is 181.

When presenting their algorithm at ASIACRYPT'14, the authors of [GJL14] updated their computation by using our suggested formulas for the bias and the number of queries. In order to obtain a complexity smaller than 2^{80}, they further improved their algorithm by the following observation: instead of assuming that the secret s_2 has a Hamming weight smaller or equal to w_1, the algorithm takes now into account all the Hamming weights that would give a good bias for the covering code reduction. I.e., the algorithm takes into account all the Hamming weights w for which $\delta' > \epsilon_{\mathsf{set}}$, where ϵ_{set} is a preset bias. The probability of a good secret changes from $\Pr(w_1, k'')$ to $\Pr(\mathsf{HW})$ that we define below. They further adapted the algorithm by using the $\mathsf{LF2}$ reduction steps. With these changes, they suggest the following parameters for $\mathsf{LPN}_{512,0.125}$:

$$a = 5 \quad b = 62 \quad \ell = 60 \quad k'' = 180 \quad w_0 = 2 \quad \epsilon_{\mathsf{set}} = 2^{-14.18}$$

Using two $[90, 30]$ linear codes, they obtain that $n = 2^{63.6} = 3 \cdot 2^b$ queries are needed, the memory used is of $m = 2^{72.6}$ bits and the time complexity is $t = 2^{79.7}$. Thus, this algorithm gives better performance than $\mathsf{LF2}$ and shows that this LPN instance does not offer a security of 80 bits.

With all the above observations we update Theorem 3.7 as follows.

Theorem 3.8. *Let $a, a', b, w_0, w_1, \ell, k', k'', \epsilon_{\mathsf{set}}$ be the algorithm parameters. The covering code* $(n = \frac{2-\delta^{2^{a+1}}\epsilon_{\mathsf{set}}^2}{\delta^{2^{a+1}}\epsilon_{\mathsf{set}}^2}\left(\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^\ell - 1}}\right)\right)^2 + a2^b$ *,*$t, m = kn + 2^{k''-\ell} + \ell 2^\ell, \theta, \ell)$*-solves the* LPN *problem* [8]*, where $\delta = 1 - 2\tau$ and ϵ_{set} is a preset bias. The code chosen for the*

[8]This n corresponds to covering code reduction using $\mathsf{LF1}$. For $\mathsf{LF2}$ reduction steps we need to have

covering code reduction step can be expressed as the concatenation of one or more linear codes. The time t complexity can be expressed as

$$t = \frac{t_{\textsf{sparse reduction}} + t_{\textsf{bkw reduction}} + t_{\textsf{guess}} + t_{\textsf{covering code}} + t_{\textsf{Walsh transform}}}{\Pr(w_0, k' - k'') \Pr(\textsf{HW})},$$

where

- $t_{\textsf{sparse reduction}} = nka'$ *is the cost of reducing the* LPN *instance to a sparse secret*
- $t_{\textsf{bkw reduction}} = (k+1)an$ *is the cost of the* BKW *reduction steps*
- $t_{\textsf{guess}} = n' \sum_{i=0}^{w_0} \binom{k'-k''}{i} i$ *is the cost of guessing $k' - k''$ bits and $n' = n - k - a2^b$ represents the number of queries at the end of the reduction phase*
- $t_{\textsf{covering code}} = (k'' - \ell)(2n' + 2^\ell)$ *is the cost of the covering code reduction and n' is again the number of queries*
- $t_{\textsf{Walsh transform}} = \left(\ell 2^{\ell \frac{\log_2(n-a2^b)+1}{2}} + \ell(n - a2^b)\right) \sum_{i=0}^{w_0} \binom{k'-k''}{i}$ *is the cost of applying the fast Walsh-Hadamard transform for every guess of $k' - k''$ bits*
- $\Pr(\textsf{HW}) = \sum_{w_i} (1-\tau)^{k''-w_i} \tau^{w_i} \binom{k''}{w_i}$ *where w_i is chosen such that the bias δ' (computed following (3.3) and (3.4)), which depends on w_i and the covering radius ρ of the chosen code, is larger than $\epsilon_{\textsf{set}}$*

3.7 LF(4)

The algorithm that was presented at EUROCRYPT'16 [ZJW16], introduces a new type of reduction that is denoted LF(4). The authors also claim to improve the complexity of the existing reduction steps through precomputation. We focus here on the structure of the algorithm of [ZJW16] and discuss its issues.

Reduction phase. Given an LPN instance, where the secret has size k, the first step of the reduction phase is to do a sample selection. It keeps from the LPN oracle only the samples that have 0 bits on an entire given window of c bits. In this way, we obtain an LPN instance where the secret has size $k - c$. This step is called *sample selection.* This idea is present also in the BKW algorithm. The next step is to change the distribution of the secret from uniform to a binomial distribution. The same steps as in Section 3.6 are used in order to change to a secret that follows the noise distribution. This step is entitled *Gaussian elimination.*

With the new secret, the algorithm is further reducing the size of the secret by xoring pairs of vectors that have the same value on a window of b bits. This can be done by applying the BKW reduction step (LF1), LF2 or a new method which is named LF(4)

$n = 1 + 2^{b+1} + k \geq \frac{2 - \delta^{2^{a+1}} \epsilon_{\textsf{set}}^2}{\delta^{2^{a+1}} \epsilon_{\textsf{set}}^2} \left(\varphi^{-1} \left(1 - (1-\theta)^{\frac{1}{2^\ell - 1}} \right) \right)^2.$

based on Wagner's algorithm [Wag02]. This new method finds four vectors from a list which, when they are xored, give a zero vector. By performing this step (either with $\mathsf{LF1}$, $\mathsf{LF2}$ or $\mathsf{LF}(4)$) t times, we end up with an $\mathsf{LPN}_{k-c-t*b,\eta'}$ instance, where $\eta' = \eta^{2^t}$ for $\mathsf{LF1}$ and $\mathsf{LF2}$ and $\eta' = \eta^{4^t}$ for $\mathsf{LF}(4)$. The use of the xoring is named *collision procedure* in [ZJW16].

Using the fact that the distribution of the secret is the same as the one of the initial noise, the algorithm is further guessing k_1 bits of the secret. More precisely, it tries all possible values of the k_1 bit vector that have a Hamming weight smaller than w_1. The probability that the secret has a Hamming weight smaller than w_1 is given in (3.1). Thus, to make the algorithm succeed, we must run it several times. Afterwards, it is applying the covering code reduction to further reduce to a $\mathsf{LPN}_{\ell,\eta'}$ instance where we use a $[k_2, \ell]$ code and $k_2 = k - c - t \cdot b - k_1$.

Solving phase. The solving phase consists in applying the Walsh transform on the remaining ℓ bits. Given that the algorithm is guessing k_1 bits of the secret, the Walsh transform has to be instantiated $\sum_{i=0}^{w_1} \binom{k_1}{i}$ times.

Complexity analysis. We state below the result from [ZJW16].

Theorem 3.9 ([ZJW16])**.** *Let N be the initial number of queries from the LPN oracle and $c, f, t, a, u, w_1, \ell, k_1, k_2$ be the parameters of the algorithm. In order to solve a $\mathsf{LPN}_{k,\tau}$ instance the overall time complexity of the algorithm is*

$$C = C_0 + \frac{PC_{11} + C_1 + C_2 + C_3 + C_4 + C_5}{\Pr(w_1, k_1)},$$

where $\Pr(w_1, k_1)$ is computed as in (3.1) and

- *$C_0 = N = 2^c n$ is the sample selection complexity and n represents the number of queries after the sample reduction*
- *$C_1 = (n - k + c)(a + \lceil (k - c)/u \rceil)$ is the cost of Gaussian elimination, where a and u are two parameters that are used in the optimization of this step*
- *$PC_{11} = (2^s - s - 1)(k - c)a$ is the cost to update the queries after the Gaussian elimination*
- *$C_2 = \sum_{i=1}^{t} \lceil \frac{k-c+1-ib}{f} \rceil (n - k - c - i2^b)$ is the cost for collision procedure when using $\mathsf{LF1}$ as reduction method and f is a parameter used in the pre-computation of the collision procedure*
- *$C_2 = \sum_{i=1}^{t} \lceil \frac{k-c+1-ib}{f} \rceil n[i]$ is the cost for collision procedure when using $\mathsf{LF2}$ as reduction method, where $n[i] = n[i-1] - 2^b$ and $n[0]$ represents the number of queries before the reduction*

- $C_2 = \sum_{i=1}^{t}(\lceil \frac{k-c+1-ib}{f} \rceil n[i] + (2^b n[i])^{1/3})$ *is the cost for collision procedure when using* LF(4) *as reduction method, where* $n[i] = \binom{n[i-1]}{4}2^{-b}$ *and* $n[0]$ *represents the number of queries before the reduction* [9]
- $C_3 = m\sum_{i=1}^{w_1}\binom{k_1}{i}$ *is the cost of the partial secret guessing and* m *is the cost of performing the xor*
- $C_4 = mh$ *is the cost of the covering code reduction method*
- $C_5 = \ell 2^\ell \sum_{i=0}^{w_1}\binom{k_1}{i}$ *is the cost of applying the Walsh transform for every guess of the secret*

Revised Analysis on LF(4). We re-evaluate the work by Zhang et al. [ZJW16]. While this new algorithm claims to improve all the previous results, we have discovered issues in its analysis. We review the LF(4) reduction method and find that this method is misleading.

The authors of the EUROCRYPT'16 paper [ZJW16] introduce a new reduction technique that they call LF(4) which is based on Wagner's algorithm [Wag02]. As aforementioned, this reduction method finds four vectors from a list which, when they are xored, give a zero. They use an information theoretic estimate on the required number of vectors for a solution to exist and the complexity results from Wagner.

If we have n vectors of b bits, information theory says we need $n \approx (4! \cdot 2^b)^{\frac{1}{4}}$ for one collision. For this n, there is an algorithm of complexity $\frac{n^2}{2} \approx \sqrt{6} \cdot 2^{b/2}$ to find a solution (make a list of XOR of two strings then look for a collision). Wagner's algorithm works with the better time complexity of $2^{b/3}$ but needs a larger $n \approx 2^{b/3}$ because it looks for solutions such that the XOR of the first two vectors starts with $\frac{b}{3}$ zero bits. For this, it makes a list of XOR of two vectors colliding on their first $b/3$ bits then look for collisions of their XOR on the remaining bits. What the authors of [ZJW16] use is a data complexity of $2^{b/4}$ and a time complexity of $2^{b/3}$ by invoking Wagner. So, they mix up the two algorithms: they take the best data complexity of the two and the best time complexity of the two without realising that this applies to two different ones. We will now exemplify this.

For their algorithm, they need more than one solution so instead of having a complexity of $2^{b/3}$, they will have $2^{b/3}n_{sol}^{1/3}$ with Wagner's algorithm, where n_{sol} represents the number of solutions for LF(4). So, for n_{sol} collisions Wagner's algorithms will require $n \approx n_{sol}^{\frac{1}{3}}2^{\frac{b}{3}}$ and have a complexity of $2^{b/3}n_{sol}^{1/3}$. For the information theory algorithm, n becomes $n = (4! \cdot n_{sol}2^b)^{\frac{1}{4}}$ and the algorithm has a complexity of $\sqrt{6} \cdot n_{sol}^{\frac{1}{2}}2^{b/2}$.

The results from [ZJW16] show that for an $\mathsf{LPN}_{512,1/8}$ for $n \approx n_{sol} = 2^{53.5}$ and $b = 156$ the complexity is $2^{72.8}$. By fixing the number of queries we want to have at the end and the length on the segment we want to find collisions, i.e. n_{sol} and b, the Wagner algorithm would require $n \approx 2^{70}$ and would have a complexity of about $\mathbf{2^{70}}$. For the information theory algorithm, we would have $n \approx \mathbf{2^{53.5}}$ and a complexity of about 2^{106}.

[9] This part is incorrect as commented below

They claim that their results are confirmed by simulations, but the results they provide miss the time complexity of the simulations.

There are also inconsistencies in the data complexity and the complexity of the Gaussian elimination step, but this analysis is outside the scope of this thesis. Details about this can be found in ePrint [BV16a]. What we are interested in is analysing the efficiency of the new reduction or solving steps.

The conclusion regarding $\mathsf{LF}(4)$ is that it does not bring improvements. The computations with the correct $\mathsf{LF}(4)$ give worse results than $\mathsf{LF1}$ and $\mathsf{LF2}$. This is because an $\mathsf{LF}(4)$ with Wagner's algorithm means 2 $\mathsf{LF2}$ consecutive reductions and the $\mathsf{LF}(4)$ with the information theory algorithm approach brings no improvement.

3.8 Other LPN Solving Algorithms

Most LPN-based encryption schemes use τ as a function of k, e.g. $\tau = \frac{1}{\sqrt{k}}$ [Ale03, DP12]. The bigger the value of k, the lower the level of noise. For $k = 768$, we have $\tau \approx 0.036$. For such a value, we say that the noise is sparse. Given that these LPN instances are used in practice, we study how we can construct other algorithms that take advantage of this extra information.

The first two algorithms presented in this section bring new ideas for the solving phase. The third one provides a method to recover the whole secret and does not need any reduction phase.

We maintain the notations used until now: n' queries remain after the reduction phase, the bias is δ' and the block size is k'.

For these solving algorithms, we assume that the secret is sparse. Even if the secret is not sparse, we can just assume that the noise is sparse. We can transform an LPN instance to an instance of LPN where the secret is actually a vector of noise bits by the method presented in [Kir11]. The details of this transform were given in Section 3.6 for the covering codes algorithm.

We denote by δ_s the bias of the secret, i.e. $\Pr[s_i = 1] = \frac{1-\delta_s}{2}$ for any $1 \leq i \leq k$. We can take $\delta_s = \delta$.

The assumption we make is that the Hamming weight of the k'-bit length secret s is in a given range. On average, we have that $\mathsf{HW}(s) = k'(\frac{1-\delta_s}{2})$, so an appropriate range is $\left[0, k'(\frac{1-\delta_s}{2}) + \frac{\sigma}{2}\sqrt{k'}\right]$, where σ is constant. We denote $k'(\frac{1-\delta_s}{2})$ by E_{HW} and $\frac{\sigma}{2}\sqrt{k'}$ by dev. Thus, we are searching in the range $[0, E_{\mathsf{HW}} + \mathsf{dev}]$. We can bound the probability that the secret has a Hamming weight outside the range by using the Hoeffding bound [Hoe63].

Let $X_1, X_2, \ldots, X_{k'}$ be independent random variables that correspond to the secret bits, i.e. $\Pr[X_i = 1] = \frac{1-\delta_s}{2}$ and $\Pr[X_i \in [0,1]] = 1$. We have $E(X) = \frac{1-\delta_s}{2}k'$. Using Theorem 2.5, we get that

$$\Pr[\mathsf{HW}(s) \text{ not in range}] = \Pr\left[\mathsf{HW}(s) - \frac{(1-\delta_s)}{2}k' \geq \sigma\sqrt{\frac{k'}{4}}\right] \leq e^{-\frac{\sigma^2}{2}}.$$

If we want to bound by $\theta/2$ the probability that $\mathsf{HW}(s)$ is not in the correct range for one block, we obtain that $\sigma = \sqrt{2\ln(\frac{2}{\theta})}$.

3.8.1 Exhaustive Search on Sparse Secret

We have $S = \sum_{i=0}^{E_{\mathsf{HW}}+\mathsf{dev}} \binom{k'}{i}$ vectors ν with Hamming weight in our range. One first idea would be to perform an exhaustive search on the sparse secret. We denote this algorithm by Search_1. For every such value ν, we compute $\mathsf{HW}(A\nu^T + c)$ (See Algorithm 3.4). In order to compute the Hamming weight, we have to compute the multiplication between A and all ν which have a Hamming weight in the correct range. This operation would take $\mathcal{O}(Sn'k')$ time but we can save a k' factor by the following observation done in [BLP11]: computing $A\nu^T$, with $\mathsf{HW}(\nu) = i$ means xoring i columns of A. If we have the values of $A\nu^T$ for all ν where $\mathsf{HW}(\nu) = i$ then we can compute $A\nu'^T$ for $\mathsf{HW}(\nu') = i+1$ by adding one extra column to the previous results.

We use here a similar reasoning done for the Walsh-Hadamard transform. When $\nu = s$, the value of $\mathsf{HW}(As^T + c)$ is equal to $\mathsf{HW}(d)$ and we assume that this is the smallest value as we have more noise bits set to 0 than 1. Thus, going through all possible values of ν and keeping the minimum will give us the value of the secret. The time complexity of Search_1 is the complexity of computing the Hamming weight, i.e. $O(Sn')$.

Algorithm 3.4 Search_1 - solving phase

```
1: **Input**: a matrix $A$ and vector $c$ s.t. $As + d = c$ from the reduction phase, values $k'$,
   $E_{\mathsf{HW}}$, $\mathsf{dev}$
2: **Output**: value $s$

3: Set $\mathsf{min_HW} = k'$
4: Set $s = \perp$
5: **for** each $\nu$ s.t. $\mathsf{HW}(\nu) \in [0, E_{\mathsf{HW}} + \mathsf{dev}]$ **do**
6:     Compute $h = \mathsf{HW}(A\nu + c)$
7:     **if** $h < \mathsf{min_HW}$ **then**
8:         $\mathsf{min_HW} = h$
9:         $s = \nu$
10:    **end if**
11: **end for**
12: **return** $s$
```

Besides Search_1, which requires a matrix multiplication for each trial, we also discovered that a Walsh transform can be used for a sparse secret. We call this algorithm Search_2. The advantage is that a Walsh transform is faster than a naive exhaustive search and thus, improves the time complexity. We thus compute the fast Walsh-Hadamard transform and search the maximum of $\hat{f}$ only for those S values with Hamming weight in the correct range (See Algorithm 3.5). Given that we apply a Walsh transform, we get that the complexity of this solving algorithm is $\mathcal{O}(k'2^{k'\frac{\log_2(n')+1}{2}})$. So, it is more interesting than Search_1 when $Sn' > k'2^{k'\frac{\log_2(n')+1}{2}}$.

Algorithm 3.5 Search_2 - solving phase

```
1: Input: a matrix $A$ and vector $c$ s.t. $As + d = c$ from the reduction phase, values $k'$,
   $E_{\mathsf{HW}}$, $\mathsf{dev}$
2: Output: value $s$

3: $f(x) = \sum_{(a_i)\in A} 1_{a_{i1,\ldots,ik'}=x}(-1)^{c_i}$
4: $\hat{f}(\nu) = \sum_u (-1)^{\nu u} f(w)$                ▷ Walsh transform of $f(x)$
5: $(s_{p_1}, \ldots, s_{p_b}) = \mathsf{arg\ max}(\hat{f}(\nu))$
6:     where $\mathsf{HW}(s_{p_1}, \ldots, s_{p_b}) \in [0, E_{\mathsf{HW}} + \mathsf{dev}]$
7: return $s$
```

For both algorithms, the failure probability is given by the scenario where there exists another sparse value $\nu \neq s$ such that $\mathsf{HW}(A\nu^T + c) \leq \mathsf{HW}(As^T + c)$. We apply a similar analysis with the one done for WHT, where now we have S vectors in total. We obtain that we need $n' = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta/2)^{\frac{1}{S-1}}\right)\right)^2$. Another failure scenario, that we take into account into our analysis, occurs when the secret has a Hamming weight outside our range. Thus, we can say that with $\sigma = \sqrt{2\ln(\frac{2}{\theta})}$ and $n = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta/2)^{\frac{1}{S-1}}\right)\right)^2 + (a-1)2^b$, the probability of failure is smaller than θ.

Complexity analysis. Taking $n = n' + (a-1)2^b$, $k' = b$, $\delta' = \delta^{2^{a-1}}$ and $\delta_s = \delta$, we obtain the following theorems for Search_1 and Search_2:

Theorem 3.10. *Let $S = \sum_{i=0}^{E_{\mathsf{HW}}+\mathsf{dev}} \binom{b}{i}$ where $E_{\mathsf{HW}} = b(\frac{1-\delta_s}{2})$ and $\mathsf{dev} = \frac{\sigma}{2}\sqrt{b}$ and let $n' = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta/2)^{\frac{1}{S-1}}\right)\right)^2$. For $k \leq a \cdot b$ and a secret s with bias δ_s, the* Search_1 *algorithm heuristically ($n = n' + (a-1)2^b, t = \mathcal{O}(kan + n'S), m = kn + b\binom{b}{E_{\mathsf{HW}}+\mathsf{dev}}, \theta, b$)-solves the* LPN *problem.*

Theorem 3.11. *Let $S = \sum_{i=0}^{E_{\mathsf{HW}}+\mathsf{dev}} \binom{b}{i}$ where $E_{\mathsf{HW}} = b(\frac{1-\delta_s}{2})$ and $\mathsf{dev} = \frac{\sigma}{2}\sqrt{b}$ and let $n' = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta/2)^{\frac{1}{S-1}}\right)\right)^2$. For $k \leq a \cdot b$ and a secret s with bias δ_s, the* Search_2 *algorithm heuristically ($n = n' + (a-1)2^b, t = \mathcal{O}(kan + b2^b\frac{\log_2 n'+1}{2} + bn'), m = kn, \theta, b$)-solves the* LPN *problem.*

Here, we take the probability, that any of the two failure scenarios to happen, to be each $\theta/2$. A search for the optimal values such that their sum is θ, brings a very little improvement to our results. We expect to require less queries for exhaustive search compared to LF1. As the asymptotic time complexity of Search_2 is the same as LF1 and the number of queries is smaller, we expect to see that this algorithm runs faster than LF1.

3.8.2 Meet in the Middle on Sparse Secret (MITM)

While MITM is a widely used technique for analysing block ciphers, stream ciphers and hash functions [GJNS14, NS16, Sas13, SKS$^+$13, WSK$^+$12, SWS$^+$12, Iso11, IS12, CNV13, GLRW10, DSP07, BR10], we try to use it for LPN and we use a mask in order to remove the effect of the noise. The idea of MITM is used also for solving the LWE problem [AFFP14, BG14, APS15].

Given that $As^T + d = c$, we split s into s_1 and s_2 and rewrite the equation as $A_1 s_1^T + d = A_2 s_2^T + c$. With this split, we try to construct a meet-in-the-middle attack by looking for $A_2 s_2^T + c$ close to $A_1 s_1^T$. The secret s has size k' and we split it into s_1 of size k_1 and s_2 of size k_2 such that $k_1 + k_2 = k'$. We consider that both s_1 and s_2 are sparse. Thus the Hamming weight of s_i lies in the range $\left[0, k_i(\frac{1-\delta_s}{2}) + \frac{\sigma'}{2}\sqrt{k_i}\right]$. We denote $k_i(\frac{1-\delta_s}{2}) + \frac{\sigma'}{2}\sqrt{k_i}$ by $\mathsf{max}_{\mathsf{HW}}(k_i)$. In order to bound the probability that both estimates are correct, we use the same bound shown in Section 3.8.1 and obtain that $\sigma' = \sqrt{2\ln(\frac{4}{\theta})}$.

For our MITM attack, we have a pre-computation phase. We compute and store $A_1 s_1^T$ for all $S_1 = \sum_{i=0}^{\mathsf{max}_{\mathsf{HW}}(k_1)} \binom{k_1}{i}$ possible values for s_1. We do the same for s_2, i.e compute $A_2 s_2^T + c$ for all $S_2 = \sum_{i=0}^{\mathsf{max}_{\mathsf{HW}}(k_2)} \binom{k_2}{i}$ vectors s_2. The pre-computation phase takes $(S_1 + S_2)n'$ steps in total. Afterwards, we pick ξ bit positions and hope that the noise d has only values of 0 on these positions. If this is true, then we could build a mask μ that has Hamming weight ξ such that $d \wedge \mu = 0$. The probability for this to happen is $(\frac{1+\delta'}{2})^\xi = e^{-\xi \ln \frac{2}{1+\delta'}}$.

We build our meet-in-the-middle attack by constructing a hash table where we store, for all s_2 values, $A_2 s_2^T + c$ at position $h((A_2 s_2^T + c) \wedge \mu)$. We have S_2 vectors s_2, so we expect to have $S_2 2^{-\xi}$ vectors on each position of the hash table. For all S_1 values of s_1, we check for collisions, i.e. $h((A_1 s_1^T) \wedge \mu) = h((A_2 s_2^T + c) \wedge \mu)$. If this happens, we check if $A_1 s_1^T$ xored with $A_2 s_2^T + c$ gives a vector d with a small Hamming weight. Remember that with the pre-computed values we can compute d with only one xor operation. If the resulting vector has a Hamming weight in our range, then we believe we have found the correct s_1 and s_2 values and we can recover the value of s. Given that $A_1 s_1^T + A_2 s_2^T + d = c$, we expect to have $(A_2 s_2^T + c) \wedge \mu = A_1 s_1^T \wedge \mu$ only when $d \wedge \mu = 0$. The condition $d \wedge \mu = 0$ holds with a probability of $(\frac{1+\delta'}{2})^\xi$ so we have to repeat our algorithm $(\frac{2}{1+\delta'})^\xi$ times in order to be sure that our condition is fulfilled. The steps of the meet-in-the-middle algorithm are illustrated in Algorithm 3.6.

As for exhaustive search, we have two scenarios that could result in a failure. One scenario is when s_1 or s_2 have a Hamming weight outside the range. The second one happens when there is another vector $\nu \neq s$ such that $\mathsf{HW}(A_1 \nu_1^T + A_2 \nu_2^T + c) \leq \mathsf{HW}(A_1 s_1^T + A_2 s_2^T + c)$ and $(A_1 \nu_1^T + A_2 \nu_2^T + c) \wedge \mu = 0$.

Complexity analysis. The time complexity of constructing the MITM attack is $(S_1 + S_2)n' + ((S_1 + S_2)\xi + S_1 S_2 2^{-\xi} n') \cdot (\frac{2}{1+\delta'})^\xi$. We include here the cost of the pre-computation phase and the actual MITM cost. We obtain that the time complexity is $\mathcal{O}((S_1 + S_2)n' + (S_1 + S_2)\xi(\frac{2}{1+\delta'})^\xi + S_1 S_2 n'(\frac{1}{1+\delta'})^\xi)$. Taking again $n' = n - (a-1)2^b$, $k' = b$, $\delta' = \delta^{2^{a-1}}$,

Algorithm 3.6 MITM Algorithm - solving phase

```
1: Input: a matrix $A$ and vector $c$ s.t. $As + d = c$ from the reduction phase, values $k'$,
   $\delta'$, $\mathsf{max}_{\mathsf{HW}}(\cdot)$, $\xi$, hash function $h$
2: Output: the value $s$

3: Split $s$ in $s_1$ and $s_2$ and $A$ into $A_1$ and $A_2$          ▷ $s_i \in \{0,1\}^{k_i}$ s.t. $k_1 + k_2 = k'$
4:         such that $A_1 s_1^T + d + A_2 s_2^T = c$
5: for $j = 1$ to $(\frac{2}{1+\delta'})^\xi$ do
6:     Construct mask $\mu$ s.t. $\mathsf{HW}(\mu) = \xi$
7:     for each $\nu_2$ s.t. $\mathsf{HW}(\nu_2) \in [0, \mathsf{max}_{\mathsf{HW}}(k_2)]$ do
8:         Store $A_2\nu_2^T + c$ at position $h((A_2\nu_2^T + c) \wedge \mu)$
9:     end for
10:    for each $\nu_1$ s.t. $\mathsf{HW}(\nu_1) \in [0, \mathsf{max}_{\mathsf{HW}}(k_1)]$ do
11:        if $h((A_1\nu_1^T) \wedge \mu) = h((A_2\nu_2^T + c) \wedge \mu)$ then
12:            if $\mathsf{Hw}(A_1\nu_1^T + A_2\nu_2^T + c) \in [0, \mathsf{max}_{\mathsf{HW}}(k')]$ then
13:                Construct $s$ from $\nu_1$ and $\nu_2$
14:            end if
15:        end if
16:    end for
17: end for
18: return $s$
```

$\delta_s = \delta$, we obtain the following result for MITM.

Theorem 3.12. *Let* $n' = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta/2)^{\frac{1}{S_1 S_2 - 1}}\right)\right)^2$*. Take* k_1 *and* k_2 *values such that* $b = k_1 + k_2$*. Let* $S_j = \sum_{i=0}^{\mathsf{max}_{\mathsf{HW}}(k_j)} \binom{k_j}{i}$ *where* $\mathsf{max}_{\mathsf{HW}}(k_j) = k_j(\frac{1-\delta_s}{2}) + \frac{\sigma'}{2}\sqrt{k_j}$ *for* $j \in \{1,2\}$*. For* $k \leq a \cdot b$ *and a secret* s *with bias* δ_s*, the MITM algorithm heuristically* $(n = n' + (a-1)2^b, t = \mathcal{O}(kan + (S_1+S_2)n' + (S_1+S_2)\xi(\frac{2}{1+\delta^{2^{a-1}}})^\xi + S_1 S_2 n'(\frac{1}{1+\delta^{2^{a-1}}})^\xi),$ $m = kn + S_2 + (S_1+S_2)n', \theta, b)$*-solves the* LPN *problem.*

3.8.3 Gaussian Elimination

In the case of a sparse noise, one may try to recover the secret s by using Gaussian elimination. It is well known that LPN with noise 0, i.e. $\tau = 0$, is an easy problem. This idea was used in [CTIN09] in order to mount a passive attack on HB and HB$^+$ protocols. If we are given $\Theta(k)$ queries for which the noise is 0, one can just run Gaussian elimination and in $\mathcal{O}(k^3)$ recover the secret s. For a $\mathsf{LPN}_{k,\tau}$ instance, the event of having no noise for k queries happens with a probability $p_{\mathsf{nonoise}} = (1-\tau)^k$.

We design the following algorithm for solving LPN: first, we have no reduction phase. For each k new queries, we assume that the noise is 0. We recover an ν through Gaussian elimination. We must test if this value is the correct secret by computing the Hamming weight of $A'\nu^T + c'$, where A' is the matrix that contains n' fresh queries and c' is the vector containing the corresponding noisy inner products. We expect to have a Hamming weight in the range $[0, (\frac{1-\delta}{2})n' + \sigma\frac{\sqrt{n'}}{2}]$, where σ is a constant.

If we want to bound by $\theta/2$ the probability that the Hamming weight of the noise is not in the correct range, for the correct secret, we obtain that $\sigma = \varphi^{-1}(1-\theta/2)\sqrt{1-\delta^2}$.

For a $\nu \neq s$, we use the Central Limit Theorem to bound that $\mathsf{HW}(A'\nu^T + c')$ is in the correct range. Let $X_1, \ldots, X_{n'}$ be the i.i.d random variables that correspond to $X_i = \langle v_i, \nu\rangle \oplus c_i$. Let $X = X_1 + \ldots + X_{n'}$. We have $E(X_1) = \frac{1}{2}$ and $\mathsf{Var}(X_1) = \frac{1}{4}$. Using Theorem 2.6, we obtain

$$\begin{aligned}\Pr[\mathsf{failure}] &= \Pr[\exists \nu s.t.\ \mathsf{HW}(A'\nu^T + c') \text{ in correct range}]\\ &= \left(1 - \left(1 - \Pr[X \leq (\frac{1-\delta}{2})n' + \frac{\sigma\sqrt{n'}}{2}]\right)^{2^k-1}\right)\\ &= \left(1 - \left(1 - \varphi(\sigma - \delta\sqrt{n'})\right)^{2^k-1}\right)\end{aligned}$$

If we bound this probability of failure by $\theta/2$ we obtain that we need at least $n' = \left(\sigma - \varphi^{-1}(1-(1-\theta/2)^{\frac{1}{2^k-1}})\right)^2\delta^{-2}$ queries besides the k that are used for the Gaussian elimination.

As aforementioned, with a probability of $p_{\mathsf{nonoise}} = (1-\tau)^k$, the Gaussian elimination will give the correct secret. Thus, we have to repeat our algorithm $\frac{1}{p_{\mathsf{nonoise}}}$ times.

Complexity analysis. The computation of the Hamming weight has a cost of $\mathcal{O}(n'k^2)$. Given that we run the Gaussian elimination and the verification step $\frac{1}{p_{\mathsf{nonoise}}}$ times, we obtain the following theorem for this algorithm:

Theorem 3.13. *Let $n' = \left(\sigma - \varphi^{-1}(1-(1-\theta/2)^{\frac{1}{2^k-1}})\right)^2\delta^{-2}$. The Gaussian elimination algorithm ($n = \frac{k+2}{(1-\tau)^k} + n', t = \mathcal{O}\left(\frac{n'k^2+k^3}{(1-\tau)^k}\right), m = k^2 + n'k, \theta, k$)-solves the* LPN *problem.*[10]

Remark 2. *Notice that this algorithm recovers the whole secret at once and the only assumption we make is that the noise is sparse. We do not need to run the transform such that we have a sparse secret and there are no queries lost during the reduction phase.*

Remark 3. *In the extreme case where $(1-\tau)^k > \theta$, the Gaussian elimination algorithm can just assume that k queries have noise 0 and retrieve the secret s without verifying that this is the correct secret.*

3.9 Solving LPN with a Polynomial Number of Samples

So far, all the algorithms presented here require access to a subexponential number of LPN queries. In real applications, e.g. passive attack on LPN-based encryption schemes,

[10]Given that we receive uniformly distributed vectors from the LPN oracle, from $n+2$ vectors v we expect to have n linearly independent ones.

an adversary has a limited access to these queries. Thus, it is of interest to see what is the complexity of solving LPN when n, the initial number of queries, is polynomial. Lyubashevsky presented in [Lyu05] a subexponential algorithm for solving LPN when $n = k^{1+\epsilon}$, for $\epsilon > 0$. We present in this Section the main results from [Lyu05] and later in this thesis (in Chapter 6) we will prove that one can do better.

Pre-reduction phase. The aim of this phase is to construct enough LPN-like samples from the $n = k^{1+\epsilon}$ LPN initial queries. The new queries are going to be given as input to the BKW algorithm, i.e. following the reduction and solving phase.

A way to increase the number of queries is to construct w combinations of them. At the end, we would have $\binom{n}{w}$ new queries. The following results show that by choosing a correct w, in the end, we have enough samples and that they behave like the LPN ones.

The following Lemma shows that if we make random combinations of the v_i vectors, the new vectors are also uniformly distributed.

Lemma 3.14 (Lemma 1 from [Lyu05])**.** *Let $X \subseteq \{0,1\}^{k^{1+\epsilon}}$ such that $|X| > 2^{2k}$ and let $Y = \{0,1\}^k$. Let $\mathcal{H}$ be the universal family of hash functions from X to Y, where $\mathcal{H} = \{h_v | v = (v_1, \ldots, v_{k^{1+\epsilon}}), v_i \in \{0,1\}^k\}$, and where $h_v(x) = x_1 v_1 \oplus \ldots \oplus x_{k^{1+\epsilon}} v_{k^{1+\epsilon}}$. If h_v is chosen uniformly at random from $\mathcal{H}$, then, with probability at least $1 - 2^{-\frac{k}{4}}$, $d(h_v(x), U) \leq 2^{-\frac{k}{4}}$ where x is chosen uniformly at random from X and U is the uniform distribution over Y.*

Once it is proven that $\mathcal{H}$ is a universal family of hash functions, one can use the Leftover Hash Lemma and prove Lemma 3.14. For this algorithm, X is chosen to be $X = \{x \in \{0,1\}^{k^{1+\epsilon}} | HW(x) = \lceil \frac{2k}{\epsilon \log_2 k} \rceil\}$. Thus, the new queries are a combination of $w = \lceil \frac{2k}{\epsilon \log_2 k} \rceil$ initial queries. The new noisy inner products have a noise of $\tau' \leq \frac{1}{2} - \frac{1}{2}\left(\frac{1-2\tau}{4}\right)^{\frac{2k}{\epsilon \log_2 k}}$.

At the end of this phase, the algorithm gives the new queries to the BKW algorithm.

Reduction phase. This phase follows the same steps that are done in the BKW algorithm.

Solving phase. It uses the majority rule as in BKW.

Complexity analysis. We state the main result from [Lyu05].

Theorem 3.15 (Th. 1 from [Lyu05])**.** *Let $n = k^{1+\epsilon}$ be the number of samples from the LPN oracle and let $w = \lceil \frac{2k}{\epsilon \log_2 k} \rceil$ with $\epsilon > 0$. For $\tau < \frac{1}{2} - 2^{-(\log_2 k^{\sigma})}$ for any constant $\sigma < 1$, the algorithm $(n, t = 2^{\mathcal{O}(\frac{k}{\log_2 \log_2 k})}, m = k\binom{n}{w}, \theta, b)$-solves the LPN problem.*

Chapter 4

Comparing the LPN Solving Algorithms

Having provided a tighter analysis of the LPN solving algorithms, in this chapter, we provide a comparison between the theory and practice with respect to the performance of these algorithms. The personal contribution in this chapter is a joint work with Florian Tramèr and Serge Vaudenay that was published in Journal of Cryptography and Communications [BTV16].

Structure of the Chapter. We present the environment and the libraries we used for the implementation in Section 4.2. We also give the comparison between the theory and the practice for the algorithms described in Sections 3.3 - 3.8. We compare the performance of all these algorithms and show what are the secure parameters that LPN should be instantiated with in Section 4.3.

4.1 Our Contribution

Having a better analysis of the LPN solving algorithms described in the previous chapter, *we provide experimental results and compare them with the practice.* For this, we *implement all the algorithms* that we describe. Our motivation is to study the gap between theory and practice. We discover that our practical results are very close to our theoretical results. Besides comparing theory with practice for each algorithm, *we compare all the algorithms for a family of* LPN *instances where the secret is sparse* and establish that the Gaussian elimination is the best algorithm for this case. As we validate our theory, we can *propose secure parameters for different* LPN *instances, where we vary the size of the secret and the noise level.* We see that depending on the noise level, we have several algorithms that are the most efficient. For a sparse noise the Gaussian elimination is the best. For a a constant noise level, the covering code and the LF2 provide the best results.

4.2 Comparing Theory with Practice

In this section, we compare the theoretical analysis with implementation results of all the LPN solving algorithms described in Sections 3.3 - 3.8.

We implemented the BKW, LF1 and LF2 algorithms as they are presented in [LF06] and in pseudocode in Algorithms 3.1-3.3. The implementation was done in C on a Intel Xeon 3.33Ghz CPU. We used a custom bit library to store and handle bit vectors. Using the OpenMP library[1], we have also parallelized certain crucial parts of the algorithms. The xor-ing in the reduction phases as well as the majority phases for instance, are easily distributed onto multiple threads to speed up the computation. Furthermore, we implemented the exhaustive search and MITM algorithms described in Section 3.8. The various matrix operations performed for the sparse LPN solving algorithms are done with the M4RI library [2]. Regarding the memory model used, we implemented the one described in [LF06] in order to accommodate the LF2 algorithm. The source code of our implementation can be found at `http://lasec.epfl.ch/lpn/lpn_source_code.zip`.

We ran all the algorithms for different LPN instances, where the size of the secret varies from 48 to 100 bits and the Bernoulli parameter τ takes different values from 0.01 to 0.4. A value of $\tau = 0.1$ for a small k as the one we are able to test means that very few, if none, of the queries have the noise bits set on 1. For this sparse case, an exhaustive search is the optimal strategy. Also, $\tau = 0.4$ might also seem to be an extreme case. Still, we provide the query complexity for these extreme cases to fully observe the behaviour of the LPN solving algorithms.

For each LPN instance, we try to find the theoretical number of oracle queries required to get a 50% probability of recovering the full secret while optimizing the time complexity. This means that in half of our instances, we recover the secret correctly. In the other half of the cases, it may happen that one or more bits are guessed wrong. We thus take $\theta = \frac{1}{3}$ as the probability of failure for the first block. We choose a and b that would minimize the time complexity and we apply this split in our theoretical bounds in order to compute the theoretical number of initial queries. We apply the same split in practice and try to minimize the number of initial queries such that we maintain a 50% probability of success. We thus experimented with different values for the original number of oracle samples, and ran multiple instances of the algorithms to approximate the success probability. One can observe that in our practical and theoretical results the a, b parameters are the same and the comparison is consistent. We were limited by the power of our experimental environment and thus, we were not able to provide results for instances that require more than 2^{30} queries.

4.2.1 BKW*

The implementation results for BKW* are presented in Table 4.1. Each entry in the table is of the form $\log_2(n)(a)$, where n is the number of oracle queries that were required to

[1] `http://openmp.org/wp`

[2] `http://m4ri.sagemath.org/`

Table 4.1: BKW* query complexity - practice ($\log_2$)

τ	k			
	48	64	80	100
0.01	11.85(6)	15.01(6)	17.68(7)	20.78(7)
0.10	19.99(4)	23.13(4)	27.30(4)	
0.20	23.84(3)			
0.25	25.95(3)			
0.40				

Table 4.2: BKW* query complexity - theory ($\log_2$)

τ	k			
	48	64	80	100
0.01	11.69(6)	14.84(6)	17.69(7)	20.80(7)
0.10	19.02(4)	23.19(4)	27.32(4)	32.43(4)
0.20	23.94(3)	30.11(3)	33.97(4)	39.09(4)
0.25	26.05(3)	32.23(3)	37.33(3)	43.30(4)
0.40	35.56(2)	42.81(3)	47.91(3)	55.02(3)

obtain a 50% success rate for the full recovery of the secret. The parameter a is the algorithm parameter denoting the number of blocks into which the vectors were split. We take $b = \lceil \frac{k}{a} \rceil$. By maintaining the value of a, we can easily compute the number of queries and the time & memory complexity. In Table 4.2, we present the theoretical results for BKW* obtained by using Theorem 3.3. We can see that our theoretical and practical results are very close. We are a bit too optimistic with the theory for the small values of k, i.e. $k = 48$ and $k = 60$ where we require a bit less number of queries. For larger values, our theory requires more queries than the practice.

If we take the example of $\mathsf{LPN}_{100,0.01}$, we need $2^{20.78}$ queries and our theoretical analysis gives a value of $2^{20.80}$. These two values are very close. We emphasize again that for both the theory and the practice, we use the split that optimizes the time complexity and from this optimal split we derive the number of queries.

Remark 4. *For the BKW* algorithm, we tried to optimize the average final bias of the queries, i.e. obtaining a better value than $\delta^{2^{a-1}}$. Recall that at the beginning of the reduction phase, we order the queries in equivalence classes and then choose a representative vector that is xored with the rest of queries from the same class. One variation of this reduction operation would be to change several times the representative vector. The incentive for doing so is the following: one representative vector that has error vector set on 1 affects the bias δ of all queries, while by choosing several representative vectors this situation may be improved; more than half of them will have error bit on 0. We implemented this new approach, but we found that it does not bring any significant improvement. Another change that was tested was about the majority rule applied during the solving phase. Queries have a worst case bias of $\delta^{2^{a-1}}$ (See Lemma 3.1), but some have a larger bias. So, we could apply a weighted majority rule. This would decrease the number of queries needed for the solving phase. We implemented the idea and discovered that the complexity advantage is very small.*

Remark 5. *In our implementation, we try to optimize the way we split our blocks during the reduction phase. The optimal split might give an a that does not divide k. For the*

Table 4.3: LF1 query complexity - practice ($\log_2$)

τ	k			
	48	64	80	100
0.01	9.78(7)	11.29(8)	13.32(8)	14.99(8)
0.10	13.20(4)	15.52(5)	17.98(5)	21.38(5)
0.20	16.30(4)	18.03(4)	21.04(4)	25.18(4)
0.25	16.20(3)	20.70(4)	22.24(4)	25.93(4)
0.40	23.49(3)	23.97(3)		

Table 4.4: LF1 query complexity - theory ($\log_2$)

τ	k			
	48	64	80	100
0.01	9.90(7)	12.24(8)	13.44(8)	15.94(8)
0.10	13.69(4)	16.12(5)	18.24(5)	22.02(5)
0.20	16.75(4)	18.34(4)	21.66(4)	26.59(4)
0.25	17.01(3)	21.36(4)	22.60(4)	26.64(4)
0.40	23.84(3)	24.82(3)	28.13(3)	35.00(3)

theoretical analysis, we always assume that k can be written as $k = a \cdot b$. When comparing the theory with practice, the results for the theoretical values give an upper bound as we take $b = \lceil \frac{k}{a} \rceil$ and we actually give exact results for a secret of size $a\lceil \frac{k}{a} \rceil \geq k$. This observation holds for all the algorithms we implemented.

4.2.2 LF1

Below, we present the experimental and theoretical results for the LF1 algorithm. As a first observation we can see that, for all instances, this algorithm is a clear optimization over the original BKW* algorithm. As before, each entry is of the form $\log_2(n)(a)$, where n and a are selected to obtain a 50% success rate for the full recovery of the secret and $b = \lceil \frac{k}{a} \rceil$.

Table 4.4 shows our theoretical results for LF1 using Theorem 3.5. When we compare the experimental and the practical results for LF1 (See Table 4.3 and Table 4.4), we can see that the gap between them is of a factor up to 2.

Remark 6. *Recall that in* LF1*, like in all* LPN *solving algorithms, we perform the reduction phase by splitting the queries into a blocks of size b. When this split is not possible, we consider that we have $a - 1$ blocks of size b and a last block shorter of size b' with $b' < b$. By* LF1**, we denote the same* LPN *solving algorithm that makes use of the Walsh transform but where the split of the blocks is done different. We allow now to have a last block larger than the rest. The gain for this strategy may be the following: given that we recover a larger block of the key, we run our solving phase fewer times. Although the complexity of the transform is bigger as we work with a bigger block, the reduction phase has to be applied fewer times. From our experiments, we discover there seems to be no difference between the performance of the two algorithms.*

4.2.3 LF2

We tested the LF2 heuristic on the same instances as for BKW* and LF1. The results are summarized in Table 4.5. To illustrate the performance of the heuristic, we concentrate

Table 4.5: LF2 query complexity - practice ($\log_2$)

τ	k			
	48	64	80	100
0.01	8.97(7)	10.24(7)	12.41(8)	13.15(8)
0.10	12.60(4)	15.12(5)	16.90(5)	20.65(5)
0.20	15.40(3)	16.94(4)	20.47(4)	24.88(4)
0.25	15.92(3)	20.61(4)	21.00(4)	25.40(4)
0.40	19.74(2)	23.52(3)		

on a particular instance, $\mathsf{LPN}_{100,0.1}$ with $a = 5, b = 20$. As derived in [LF06], the LF1 algorithm for this parameter set should require less than $(8 \cdot b + 200) \cdot \delta^{-2^a} \approx 2^{18.77}$ queries for a solving phase and $(a-1) \cdot 2^b + (8 \cdot b + 200) \cdot \delta^{-2^a} \approx 2^{22.13}$ queries overall to achieve a success probability of 50%. Using our theoretical analysis, the LF1 algorithm for this parameter set requires to have $\approx 2^{22.05}$ queries overall and $2^{15.91}$ queries for the solving phase. Our experimental results for LF1 were a bit lower than our theoretical ones: $2^{21.38}$ oracle samples were sufficient. If we use the LF2 heuristic starting with $1 + 2 * 2^b \approx 2^{21}$ samples, we get about the same amount of vectors for the solving phase. In this case, there are no queries lost during reduction. We thus have much more queries than should actually be required for a successful solving phase and correctly solve the problem with success probability close to 100%. So we can try to start with less. By starting off with $2^{20.65}$ queries and thus loosing some queries in each reduction round, we also solved the LPN problem in slightly over 50% of the cases. The gain in total query complexity for LF2 is thus noticeable but not extremely important.

As another example, consider the parameter set $k = 768, \tau = 0.05$ proposed at the end of [LF06]. The values for a, b which minimize the query complexity are $a = 9, b = 86$ ($a \cdot b = 774 > k$). Solving the problem with LF1 should thus require about $2^{85.66}$ vectors for the solving phase and $2^{89.13}$ oracle samples overall. Using LF2, as $1 + 2 * 2^b \approx 2^{87}$ oracle samples would be sufficient, we obtain a reduction by a factor ≈ 4.

Even though LF2 introduces linear dependencies between queries, this does not seem to have any noticeable impact on the success probability in recovering the secret value.

Remark 7. *A general observation for all these three algorithms, shown also by our results, is that the bias has a big impact on the number of queries and the complexity. Recall that the bias has value $\delta^{2^{a-1}}$ at the end of the reduction phase. We can see from our tables that the lower the value of τ, i.e. larger value of $\delta = 1 - 2\tau$, the higher a can be chosen to solve the* LPN *instance. Also, for a constant τ, the higher the size of the secret, the higher a can be chosen.*

Remark 8. *The* LF2 *algorithm is a variation of* LF1 *that offers a different heuristic technique to decrease the number of initial queries. The same trick could be used for* BKW*, *exhaustive search and MITM.*

While the same analysis can be applied for exhaustive search and MITM as for LF2, BKW* *is a special case. We denote by* BKW^2 *this variation of* BKW *where we use the reduction phase from* LF2. *Recall that for* BKW*, *we need to have* $n = 2^b \frac{1-\delta^{2^a}}{\delta^{2^a}} \left(\varphi^{-1}\left(1-\frac{\theta}{b}\right)\right)^2 + (a-1)2^b$ *queries and here the dominant term is* $2^b \frac{1-\delta^{2^a}}{\delta^{2^a}} \left(\varphi^{-1}\left(1-\frac{\theta}{b}\right)\right)^2$. *Thus, we need to start with* $2 \cdot 2^b + \epsilon$, *where* $\epsilon > 1$ *and increase such that at the end of the last iteration of the reduction, we get the required number of queries. This improves the initial number of queries and we have a gain of a factor a for the time complexity. For an* $\mathsf{LPN}_{48,0.1}$ *instance, our implementation of* BKW^2 *requires* $n = 2^{13.82}$ *initial queries and increases it, during the reduction phase, up to* $2^{19.51}$, *the amount of queries needed for the solving phase. Thus, there is an improvement from* $2^{19.99}$ *(See Table 4.1) to* $2^{13.82}$ *and the time complexity is better. While this is an improvement over* BKW*, *it still performs worse than* LF1 *and* LF2.

4.2.4 Exhaustive Search

Recall that for exhaustive search, we have two variants. The results for Search_1 are displayed in Table 4.6 and Table 4.7. For Search_1 we observe that the gap between theory and practice is of a factor smaller than 4. In terms of number of queries, Search_1 brings a small improvement compared to LF1. We will see in the next section the complete comparison between all the implemented algorithms.

Remark 9. *One may observe a larger difference for the* $\mathsf{LPN}_{48,0.4}$ *and* $\mathsf{LPN}_{64,0.25}$ *instances. For example for* $\mathsf{LPN}_{48,0.4}$ *we have* $n = 2^{19.74}$ *(practice) vs.* $n = 2^{24.00}$ *(theory). For this case, the implementation requires* $n = 2^{19.74}$ *initial queries compared with the theory that requires* $n = 2^{24.00}$ *queries. Here, we have* $a = 2$ *and* $b = 24$ *and the term* $(a-1)2^b$ *dominates the query complexity. The discrepancy comes from the worst-case analysis of the reduction phase where we say that at each reduction step we discard* 2^b *queries. With this reasoning, we predict to lose* 2^{24} *queries. If we analyse more closely, we discover that actually in the average-case, we discard only* $2^b \cdot \left[1 - \left(1 - \frac{1}{2^b}\right)^n\right]$ *queries (this is the number of expected non-empty equivalence classes). Thus, with only* $2^{19.74}$ *initial queries, we run the reduction phase and discard* $2^{19.70}$ *queries, instead of* 2^{24}. *We are left with* $2^{14.45}$, *queries which are sufficient for the solving phase. We note that for large* LPN *instances, this difference between worst-case and average-case analysis for the number of deleted queries during reduction rounds becomes negligible.*

The results for Search_2 are displayed in Table 4.8 and Table 4.9.

We notice that for both Search_1 and Search_2 the instances $\mathsf{LPN}_{48,0.01}$ and $\mathsf{LPN}_{68,0.01}$ have a very low number of queries. This is due to the following observation: for $n \leq 68$ linearly independent queries and $\tau = 0.01$ we have that the noise bits are all 0 with a probability larger than 50%. Thus, for $k \leq 64$ we hope that the k queries we receive from

Table 4.6: Search_1 query complexity - practice ($\log_2$)

τ	k			
	48	64	80	100
0.01	5.70(1)	6.12(1)	13.25(8)	14.93(8)
0.10	13.15(4)	16.44(4)	17.93(5)	21.34(5)
0.20	15.54(3)	17.99(4)	21.02(4)	25.15(4)
0.25	16.18(3)	19.88(3)		
0.40	19.74(2)			

Table 4.7: Search_1 query complexity - theory ($\log_2$)

τ	k			
	48	64	80	100
0.01	5.70(1)	6.12(1)	13.28(8)	15.90(8)
0.10	13.69(4)	17.59(4)	18.20(5)	22.02(5)
0.20	17.02(3)	18.31(4)	21.66(4)	26.59(4)
0.25	17.10(3)	23.00(3)	28.00(3)	26.64(4)
0.40	24.00(2)	24.91(3)	28.15(3)	35.00(3)

Table 4.8: Search_2 query complexity - practice ($\log_2$)

τ	k			
	48	64	80	100
0.01	5.70(1)	6.12(1)	13.25(8)	14.93(8)
0.10	13.15(4)	15.36(5)	17.93(5)	21.34(5)
0.20	16.09(4)	17.99(4)	21.02(4)	25.15(4)
0.25	16.18(3)	20.63(4)		
0.40	23.50(2)			

Table 4.9: Search_2 query complexity - theory ($\log_2$)

τ	k			
	48	64	80	100
0.01	5.70(1)	6.12(1)	13.11(8)	15.87(8)
0.10	13.66(4)	15.87(5)	18.17(5)	22.02(5)
0.20	16.58(4)	18.27(4)	21.65(4)	26.59(4)
0.25	17.10(3)	21.26(4)	22.54(4)	26.64(4)
0.40	23.86(3)	24.82(3)	28.13(3)	35.00(3)

Table 4.10: MITM query complexity - practice ($\log_2$)

τ	k			
	48	64	80	100
0.01	5.70(1)	6.12(1)	13.25(8)	14.93(8)
0.10	13.15(4)	16.47(4)		
0.20	15.54(3)			
0.25				
0.40				

Table 4.11: MITM query complexity - theory ($\log_2$)

τ	k			
	48	64	80	100
0.01	5.70(1)	6.12(1)	13.28(8)	15.90(8)
0.10	13.67(4)	17.59(4)	18.20(5)	22.01(5)
0.20	17.02(3)	18.30(4)	21.66(4)	26.59(4)
0.25	17.11(3)	23.01(3)	28.00(3)	26.64(4)
0.40	24.00(2)	32.85(2)	28.14(3)	35.00(3)

the oracle have all the noise set to 0. With k noiseless and linearly independent queries, we can just recover s with Gaussian elimination. This is an application of Remark 3 from Chapter 3.

4.2.5 MITM

In the case of MITM, the experimental and theoretical results are illustrated in Table 4.10 and Table 4.11. There is a very small difference between MITM and exhaustive search algorithms for a sparse secret: in practice, MITM requires just dozens of queries less than Search_1 and Search_2 for the same a and b parameters.

4.2.6 Gaussian Elimination

In the Gaussian elimination algorithm, the only assumption we need to make is that we have a sparse noise. We do not run any reduction technique and the noise is not affected. As the algorithm depends on the probability to have a 0 noise on k linearly independent vectors, the complexity decays very quickly once we are outside the sparse noise scenario. We present in Table 4.12 the theoretical results obtained for this algorithm.

In the next section, we will show the effectiveness of this simple idea in the sparse case scenario and compare it to the other LPN solving algorithms.

Again for $\mathsf{LPN}_{48,0.01}$ and $\mathsf{LPN}_{64,0.01}$ we apply Remark 3.

4.2.7 Covering Codes

The covering code requires the existence of a code with the optimal coverage. For each instance one has to find an optimal code that minimizes the query and time complexity. Unlike the previous algorithms, this algorithm cannot be truly automatized. In practice, we could test only the cases that were suggested in [GJL14]. Thus, we are not able to

Table 4.12: Gaussian elimination query complexity - theory ($\log_2$)

τ	k			
	48	64	80	100
0.01	5.70	6.12	8.23	8.73
0.10	12.96	15.78	18.52	21.87
0.20	21.09	26.65	32.11	38.87
0.25	25.57	32.61	39.56	48.18
0.40	41.01	53.21	65.31	80.37

compare the theoretical and practical values. Nevertheless, we will give theoretical values for different practical parameters in the next section.

4.3 Complexity Analysis of the LPN Solving Algorithms

We have compared our theoretical bounds with our practical results and we saw that there is a small difference between the two. Our theoretical analysis also gives tighter bounds compared with the results from [LF06]. We now extend our theoretical results and compare the asymptotic performance of all the LPN algorithms for practical parameters used by the LPN-based constructions. We consider the family of $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$ instances proposed by some encryption schemes [Ale03, DP12]. Although the covering code cannot be automatized, as for each instance we have to try different codes with different sizes and dimensions, we provide results also for this algorithm. When dealing with the covering code reduction, we always assume the existence of an ideal code and compute the bias introduced by this step. We do not consider here concatenation of ideal codes and we will see that we obtain a worse result for the $\mathsf{LPN}_{512,0.125}$ instance compared with the result from [GJL14], although the difference is small. In the covering code algorithm, we also stick with the BKW reduction steps and do not use the LF2 reduction. As aforementioned, the LF2 reduction brings a small improvement to the final complexity. This does not affect the comparison between all the LPN solving algorithms.

We analyse the time complexity of each algorithm, by which we mean the number of bit operations the algorithm performs while solving an LPN problem. For each algorithm, we consider values of k for which the parameters (a, b) minimising the time complexity are such that $k = a \cdot b$. For the LF2 algorithm, we select the initial number of queries such that we are left with at least $n' = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^b-1}}\right)\right)^2$ queries after the reduction phase. Recall that by Search_1, we denote the standard exhaustive search algorithm and Search_2 is making use of a Walsh-Hadamard transform. The results are illustrated in Figure 4.1. We recall the time complexity and the initial number of queries for each algorithm in Table 4.13, where S represents the number of sparse secrets with $S < 2^b$. For MITM, the values S_1 (resp. S_2) represent the number of possible values

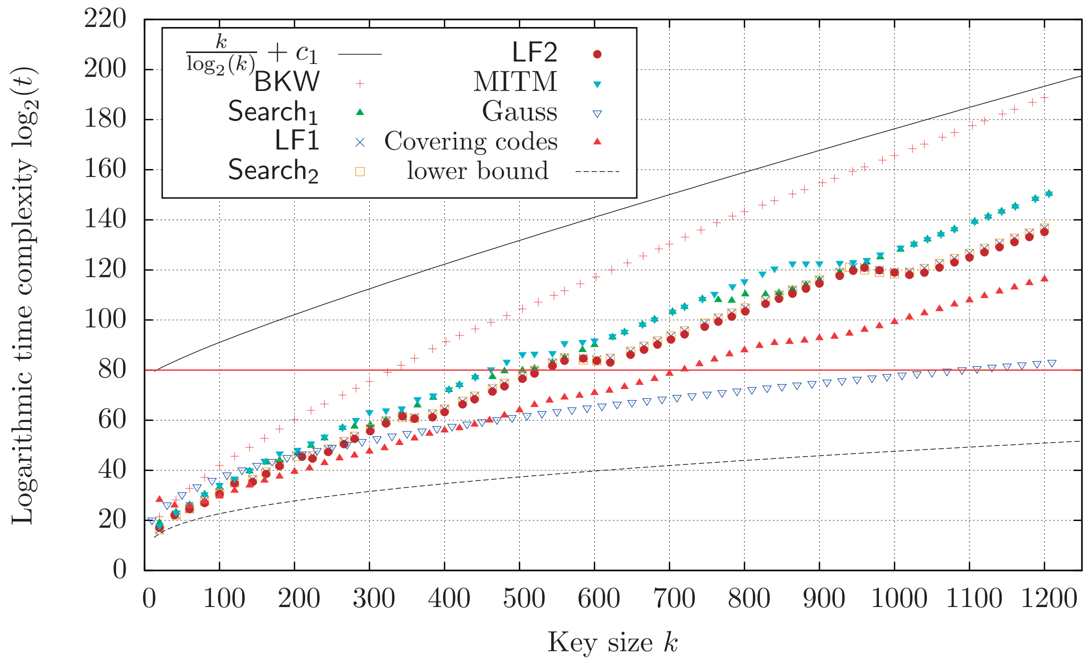

Figure 4.1: Time Complexity of LPN Algorithms on instances $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$

for the first (resp. second) half of the secret, n' represents the number of queries left after the reduction phase and ξ represents the Hamming weight of the mask used. In the case of the covering codes algorithm, all $a, b, a', k', k'', l, w_0, \epsilon_{set}$ are parameters of the algorithm. Recall that θ is $\frac{1}{3}$.

We can bound the logarithmic complexity of all these algorithms by $\frac{k}{\log_2(k)} + c_1$ and $c_3 \log_2(k) + \frac{\sqrt{k}}{\ln(2)} + c_2$ (denoted in Figure 4.1 by lower bound). The lower bound is given by the asymptotic complexity of the Gaussian elimination that can be expressed as $c \log_2 k + \frac{\sqrt{k}}{\ln(2)}$ when $\tau = \frac{1}{\sqrt{k}}$.

The complexity of BKW can be written as $\min_{k=ab}(\mathsf{poly} \cdot 2^b \cdot \delta^{-2^a})$ and for the other algorithms (except the Gaussian elimination), the formula is $\min_{k=ab}(\mathsf{poly} \cdot (2^b + \delta^{-2^a}))$, where poly denotes a polynomial factor. By searching for the optimal a, b values, we have two cases:

- for $a > 1$, we find $a \sim \log_2 \frac{k}{(\log_2 k)^2 \ln \frac{1}{\delta}}$ and $b = \frac{k}{a}$ and obtain that 2^b dominates δ^{-2^a}. For $\delta = 1 - \frac{2}{\sqrt{k}}$ we obtain the complexity $\mathsf{poly} \cdot 2^{\frac{k}{\log_2(k)}}$.
- for $a = 1$, we have that for
  - BKW the complexity is $\mathsf{poly} \cdot 2^k$
  - LF1, LF2, Search$_2$ the complexity is $\mathsf{poly} + k2^k$

– Search_1, MITM the complexity is $\mathsf{poly} \cdot S_r$ and $\mathsf{poly} \cdot S_{r'}^2$, respectively, where we define S_r to be $\#\{v \in \{0,1\}^k \mid HW(v) \leq r\}$. We need to bound the value of S_r. By induction we can show that $S_r \leq \frac{k}{k-r-1} \cdot \frac{k^r}{r!}$. For $\tau \approx \frac{1}{\sqrt{k}}$, we have that $r \approx (1+\frac{\sigma}{2})\sqrt{k}$ and $r' \approx (\frac{1}{2} + \frac{\sigma}{2\sqrt{2}})\sqrt{k}$. We obtain that the complexity for both algorithms is $\mathsf{poly} \cdot 2^{\gamma\sqrt{k}\log_2 k + \mathcal{O}(\sqrt{k})}$, where γ is a constant. This is not better than $2^{\frac{k}{\log_2(k)}}$ for $k < 200\,000$, but asymptotically this gives a better complexity.

For Gaussian elimination, the complexity is $\frac{\mathsf{poly}}{(1-\tau)^k}$ which is $\mathsf{poly} \cdot 2^{\sqrt{k}}$ for $\tau = \frac{1}{\sqrt{k}}$.

Table 4.13: Query & Time complexity for LPN solving algorithms for recovering the first b bits

LPN algorithm	Query complexity(n) and Time complexity(t)
BKW	$n = 2^b \frac{1-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-\frac{\theta}{b}\right)\right)^2 + (a-1)2^b$ $t = kan$
LF1	$n = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^b-1}}\right)\right)^2 + (a-1)2^b$ $t = kan + b2^b\frac{\log_2(n-(a-1)2^b)+1}{2} + b(n-(a-1)2^b)$
LF2	$n = 1 + 2^{b+1} \geq \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^b-1}}\right)\right)^2$ $t = kan + b2^b\frac{\log_2(n-(a-1)2^b)+1}{2} + b(n-(a-1)2^b)$
Search_1	$n = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta/2)^{\frac{1}{S-1}}\right)\right)^2 + (a-1)2^b$ $t = kan + S(n-(a-1)2^b)$
Search_2	$n = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta/2)^{\frac{1}{S-1}}\right)\right)^2 + (a-1)2^b$ $t = kan + b2^b\frac{\log_2(n-(a-1)2^b)+1}{2} + b(n-(a-1)2^b)$
MITM	$n = \frac{2-\delta^{2^a}}{\delta^{2^a}}\left(\varphi^{-1}\left(1-(1-\theta/2)^{\frac{1}{S_1S_2-1}}\right)\right)^2 + (a-1)2^b$ $t = kan + (S_1+S_2)n' + (S_1+S_2)\xi(\frac{2}{1+\delta^{2^{a-1}}})^\xi + S_1S_2n'(\frac{1}{1+\delta^{2^{a-1}}})^\xi$
Gauss	$n = \frac{k+2}{(1-\tau)^k} + \left(\sigma - \varphi^{-1}(1-(1-\theta/2)^{\frac{1}{2^k-1}})\right)^2\delta^{-2}$ $t = \frac{\left(n-\frac{k+2}{(1-\tau)^k}\right)k^2+k^3}{(1-\tau)^k}$
Covering codes	$n = \frac{2-\delta^{2^{a+1}}\epsilon_{\mathsf{set}}^2}{\delta^{2^{a+1}}\epsilon_{\mathsf{set}}^2}\left(\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^\ell-1}}\right)\right)^2 + a2^b$ $t = nka' + \mathsf{comp} + \mathsf{comp}_{\mathsf{solving}}$, where $\mathsf{comp} = (k+1)an + (n-a2^b)\sum_{i=0}^{w_0}\binom{k'-k''}{i}i + (k''-\ell)(2(n-a2^b)+2^\ell)$ $\mathsf{comp}_{\mathsf{solving}} = \left(\ell 2^\ell\frac{\log_2(n-a2^b)+1}{2} + \ell(n-a2^b)\right)\sum_{i=0}^{w_0}\binom{k'-k''}{i}$

We see in Figure 4.1 that in some cases, increasing the value of k may result in a decrease in time complexity. The reason for this is that we are considering LPN instances where the noise parameter τ takes value $\frac{1}{\sqrt{k}}$. Thus, as k grows, the noise is reduced, which leads to an interesting trade-off between the complexity of the solving phase and the complexity of the reduction phase of the various algorithms. This behaviour does not seem to occur for the BKW algorithm. In this case, the query complexity $n = 2^b \frac{1-\delta^{2^a}}{\delta^{2^a}} \left(\varphi^{-1}\left(1-\frac{\theta}{b}\right)\right)^2 + (a-1)2^b$ is largely dominated by the first term, which grows exponentially not only in terms of the noise parameter, but also in terms of the block size b.

Remark 10 ($\mathsf{LF1}$ vs. Search_2). *As shown in Figure 4.1, the overall complexity of the* $\mathsf{LF1}$ *and* Search_2 *algorithms is quasi identical. From Theorems 3.5 and 3.11, we deduce that for the same parameters (a, b), the* Search_2 *algorithm should perform better as long as $S < 2^{b-1}$. This is indeed the case for the instances we consider here, although the difference in complexity is extremely small.*

We can see clearly that for the $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$ family of instances, the Gaussian elimination outperforms all the other algorithms for $k > 500$. For no $k < 1000$, the $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$ offers an 80 bit security. This requirement is achieved for $k = 1090$.

Selecting secure parameters. We remind that for each algorithm we considered, our analysis made use of a heuristic assumption of query and noise independence after reduction. In order to propose security parameters, we simply consider the algorithm which performs best under this assumption.

By taking all the eight algorithms described in this work, Tables 4.14-4.21 display the logarithmic time complexity for various LPN parameters. For instance, the $\mathsf{LF2}$ algorithm requires 2^{79} steps to solve a $\mathsf{LPN}_{384,0.25}$ instance.

We recall here the result from [GJL14]: an instance $\mathsf{LPN}_{512,0.125}$ offers a security of 79.7. We obtain a value of 82. The difference comes mainly from the use of $\mathsf{LF2}$ reduction in [GJL14] and from a search of optimal concatenation of linear codes.

When comparing all the algorithms, we have to keep in mind that the Gaussian elimination recovers the whole secret, while for the rest of the algorithms, we give the complexity to recover a block of the secret. Still, this does not affect our comparison as we have proven in Section 3.4 that the complexity of recovering the first block dominates the total complexity.

We highlight in red the best values obtained for different LPN instances. We observe the following behaviour: for a sparse case scenario, i.e. $\tau = 0.05$ for $k \geq 576$ or $\tau = \frac{1}{\sqrt{k}} < 0.05$, the Gaussian elimination offers the best performance. Once we are outside the sparse case scenario, we have that $\mathsf{LF2}$ and the covering code algorithms are the best ones. The covering code proves to be better than $\mathsf{LF2}$ for a level of noise of 0.125. While the performance of the covering code reduction highly depends on the sparseness of the noise, $\mathsf{LF2}$ has a more general reduction phase and is more efficient for noise parameters of 0.25 and 0.4. Also, for a $\tau > 0.05$ the covering code is better than the Gaussian elimination.

Thus, for different scenarios, there are different algorithms that prove to be efficient. This comparison clearly shows that for the family of instances $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$ neither the BKW, nor its variants are the best ones. One should use the Gaussian elimination algorithm.

Table 4.14: Bit security of LPN against BKW

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	68	88	97	106	114	122	139	197
0.05	66	88	98	108	117	126	145	215
0.125	78	104	116	127	137	148	170	252
0.25	93	123	136	149	162	175	201	294
0.4	114	147	163	179	195	212	243	346

Table 4.15: Bit security of LPN against $\mathsf{LF1}$

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	50	63	71	79	84	88	102	145
0.05	50	62	71	79	87	95	102	160
0.125	56	72	78	89	98	107	125	176
0.25	65	83	89	100	110	121	143	199
0.4	76	93	104	117	130	142	168	230

Table 4.16: Bit security of LPN against LF2

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	47	59	67	75	79	84	98	141
0.05	47	59	67	75	83	91	98	156
0.125	53	69	75	85	95	104	122	173
0.25	63	79	87	98	108	119	140	197
0.4	75	90	102	115	128	140	165	228

Table 4.17: Bit security of LPN against Search_1

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	54	69	74	79	87	95	104	147
0.05	51	69	76	81	87	95	111	160
0.125	64	80	89	100	110	121	135	199
0.25	80	108	119	130	142	152	175	258
0.4	107	140	156	171	187	203	235	335

Table 4.18: Bit security of LPN against Search_2

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	50	63	71	79	83	88	102	145
0.05	50	62	71	79	87	95	102	160
0.125	56	72	78	88	98	107	125	176
0.25	65	83	89	100	110	121	143	199
0.4	76	93	104	117	130	142	168	230

Table 4.19: Bit security of LPN against $MITM$

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	55	69	75	79	87	95	108	151
0.05	51	69	78	82	87	95	111	160
0.125	64	80	89	100	110	121	138	199
0.25	81	108	120	132	143	155	177	261
0.4	107	140	156	172	188	204	235	336

Table 4.20: Bit security of LPN against Gaussian elimination

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	49	56	59	62	64	67	71	85
0.05	44	56	61	66	72	77	87	127
0.125	75	102	115	127	140	153	179	279
0.25	133	188	215	242	269	296	350	565
0.4	218	314	362	410	457	505	600	979

Table 4.21: Bit security of LPN against Covering codes

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	44	55	60	66	69	74	86	123
0.05	42	54	60	64	73	78	88	132
0.125	53	68	75	82	90	96	109	162
0.25	69	87	96	106	115	125	139	203
0.4	94	110	123	136	149	161	179	281

As we have shown, there still remains a small gap between the theoretical and practical results for the algorithms we analysed. It thus seems reasonable to take a safety margin when selecting parameters to achieve a certain level of security.

Based on this analysis, we could recommend the LPN instances $\mathsf{LPN}_{512,0.25}$, $\mathsf{LPN}_{640,0.125}$, $\mathsf{LPN}_{1280,0.05}$ or $\mathsf{LPN}_{1280,\frac{1}{\sqrt{1280}}}$ to achieve 80 bit security for different noise levels.

Chapter 5

Optimizations on LPN Solving Algorithms

In this chapter, we explain how to automatize the process of finding optimal LPN solving algorithms. The previous work from the LPN literature, presented in Chapters 3 and 4, does not explain how to choose the parameters and what is the best use of the existing tools, i.e. reduction methods, for optimising the total complexity of solving LPN. The personal contribution in this chapter is a joint work with Serge Vaudenay that was published in [BV16b].

Structure of the Chapter. In Section 5.2, we describe the main tools used to solve LPN, i.e. the reduction techniques and the most efficient solving technique. We carefully analyse the complexity of each step. Section 5.3 studies the failure probability of the entire algorithm and validates the use of the average bias in the analysis. Section 5.4 introduces the bias computation for perfect and quasi-perfect codes. We provide an algorithm to find good codes. The algorithm that searches the optimal strategy to solve LPN is presented in Sections 5.5 and 5.6. We illustrate and compare our results in Section 5.7.

5.1 Our Contribution

As a first contribution, we *analyze the existing* LPN *algorithms and study the operations that are used in order to reduce the size of the secret.*

Second, we *improve the theory behind the covering code reduction and show the link with perfect and quasi-perfect codes.* Using the average bias of covering codes allows us to use arbitrary codes and even random ones. Using the algorithm to construct optimal concatenated codes based on a pool of elementary ones allows us to improve complexities (In [GJL14], only a hypothetical code was assumed to be close to a perfect code; in [ZJW16], only the concatenation of perfect codes are used).

Third, we *optimize the order and the parameters used by the operations that reduce the size of the secret such that we minimize the time complexity required.* We *design a "meta-algorithm" that combines the reduction steps and finds the optimal strategy to solve* LPN. We *automatize the process of finding* LPN *solving algorithms, i.e. given a random* LPN *instance, our algorithm provides the description of the steps that optimize the time complexity.* In our formalization, we call such algorithms "optimal chains". We perform a security analysis of LPN based on the results obtained by our algorithm and compare our results with the existing ones. We discover that we improve the complexity compared with the results from [BTV16, LF06, ZJW16] and [GJL14].

5.2 Reduction and Solving Techniques

Recall that the LPN solving algorithms have a common structure: given an $\mathsf{LPN}_{k,\tau}$ instance with a secret s, they reduce the original LPN problem to a new LPN problem where the secret s' is of size $k' \leq k$ by applying several *reduction techniques*. Then, they recover s' using a *solving method*. The queries are updated and the process is repeated until the whole secret s is recovered. As before, we compute the complexity of the algorithm such that the probability of success is at least 50%. We present here the list of reduction and solving techniques used in the existing LPN solving algorithms. The reduction methods were presented in the previous chapters as steps in different LPN solving algorithms. For a clear comparison, we gather and list here all these methods. In a further section, we combine the reduction techniques such that we find the optimal reduction phases for solving different LPN instances.

We assume, for all the reduction steps, that we start with n queries, that the size of the secret is k, the bias of the secret bits is δ_s and the bias of the noise bits is δ. After applying a reduction step, we will end up with n' queries, size k' and biases δ' and δ'_s. Note that δ_s averages over all secrets although the algorithm runs with one target secret. As it will be clear below, the complexity of all reduction steps only depends on k, n, and the parameters of the steps but not on the biases. Actually, only the probability of success is concerned with biases. We see in Section 5.3 that the probability of success of the overall algorithm is not affected by this approach. Actually, we will give a formula to compute a value which approximates the average probability of success over the key based on the average bias.

We have the following reduction steps:

- *sparse-secret* changes the secret distribution (this reduction step was described also in Section 3.6). In the formal definition of LPN, we take the secret s to be a random row vector of size k. When other reduction steps or the solving phase depends on the distribution of s, one can transform an LPN instance with a random s to a new one where s has the same distribution as the initial noise, i.e. $s \leftarrow \mathsf{Ber}_\tau^k$.

[1]`http://des.cse.nsysu.edu.tw/asiacrypt2014/doc/1-1_SolvingLPNUsingCoveringCodes.pdf`

The reduction performs the following steps: from the n queries select k of them: $(v_{i_1}, c_{i_1}), \dots, (v_{i_k}, c_{i_k})$ where the row vectors v_{i_j}, with $1 \leq j \leq k$, are linearly independent. Construct the matrix M as $M = [v_{i_1}^T \cdots v_{i_k}^T]$ and rewrite the k queries as $sM + d' = c'$, where $d' = (d_{i_1}, \dots, d_{i_k})$. With the rest of $n - k$ queries we do the following:

$$c'_j = \langle v_j (M^T)^{-1}, c' \rangle \oplus c_j = \langle v_j (M^T)^{-1}, d' \rangle \oplus d_j = \langle v'_j, d' \rangle \oplus d_j \ .$$

We have $n-k$ new queries (v'_j, c'_j) where the secret is now d'. In Guo et al. [GJL14], the authors use an algorithm which is inappropriately called "the four Russians algorithm" [ADKF70]. This way, the complexity should be of $\mathcal{O}\left(\min_{\chi \in \mathbb{N}} \left(kn' \lceil \frac{k}{\chi} \rceil + k^3 + k\chi 2^\chi\right)\right)$.[1] Instead, the Bernstein algorithm [Ber] works in $\mathcal{O}\left(\frac{n'k^2}{\log_2 k - \log_2 \log_2 k} + k^2\right)$. We use the best of the two, depending on the parameters. Thus, we have:

> *sparse-secret* : $k' = k$; $n' = n - k$; $\delta' = \delta$; $\delta'_s = \delta$
> Complexity: $\mathcal{O}\left(\min_{\chi \in \mathbb{N}} \left(\frac{n'k^2}{\log_2 k - \log_2 \log_2 k} + k^2, kn' \lceil \frac{k}{\chi} \rceil + k^3 + k\chi 2^\chi\right)\right)$

- *partition-reduce*(b) is used by the BKW algorithm (See Section 3.3). Recall that the queries received from the oracle are of the form $(v, \langle v, s \rangle \oplus d)$. In this reduction, the v vectors are sorted in equivalence classes according to their values on a block of b bits. These b positions are chosen randomly. Two queries in the same equivalence class have the same values on the b positions. In each equivalence class, we choose a representative vector and xor it with the rest of the queries. This will give vectors with only 0 on this window of b bits. Afterwards, the representative vector is dropped. This operation reduces the secret to $k - b$ effective bits (since b bits of the secret are always xored with zero). The new bias is δ^2 as the new queries are xor of two initial ones and the number of queries is reduced by 2^b (as there are 2^b equivalence classes).

  > *partition-reduce*(b) : $k' = k - b$; $n' = n - 2^b$; $\delta' = \delta^2$; $\delta'_s = \delta_s$
  > Complexity: $\mathcal{O}(kn)$

  The next reduction, *xor-reduce*(b), is always better than *partition-reduce*(b). Nevertheless, we keep this operation in our analysis for backward compatibility with existing algorithms (e.g. to fill our Table 5.4 with the algorithms from [GJL14]).

- *xor-reduce*(b) was first used by the LF2 algorithm (See Section 3.5). The queries are grouped in equivalence classes according to the values on b random positions. In each equivalence class, we perform the xoring of every pair of queries. The size of the secret is reduced by b bits and the new bias is δ^2. The expected new number of queries is $\frac{n(n-1)}{2^{b+1}}$. For $n \approx 1 + 2^{b+1}$, the number of queries are maintained, i.e. $n' = n$.

[1]but the $k^3 + k\chi 2^\chi$ terms is missing in [GJL14].

> *xor-reduce*(b) : $k' = k - b$; $n' = \frac{n(n-1)}{2^{b+1}}$; $\delta' = \delta^2$; $\delta'_s = \delta_s$
> Complexity: $\mathcal{O}(k \cdot \max(n, n'))$

- *drop-reduce*(b) is a reduction used only by the BKW algorithm (See Section 3.3). It consists in dropping all the queries that are not 0 on a window of b bits. Again, these b positions are chosen randomly. In average, we expect that half of the queries are 0 on a given position. For b bits, we expect to have $\frac{n}{2^b}$ queries that are 0 on this window. The bias is unaffected and the secret is reduced by b bits.

  > *drop-reduce*(b) : $k' = k - b$; $n' = \frac{n}{2^b}$; $\delta' = \delta$; $\delta'_s = \delta_s$
  > Complexity: $\mathcal{O}(n)$

  The complexity of $n(1 + \frac{1}{2} + \ldots + \frac{1}{2^{b-1}}) = \mathcal{O}(n)$ comes from the fact that we do not need to check all the b bits: once we find a 1 we can stop and just drop the corresponding query.

- *code-reduce*(k, k', params) is a method used by the covering code algorithm presented in ASIACRYPT'14 [GJL14] (this reduction step was described also in Section 3.6). In order to reduce the size of the secret, one uses a linear code $[k, k']$ (which is defined by params) and approximates the v_i vectors to the nearest codeword g_i. We assume that decoding is done in linear time for the code considered. (For the considered codes, decoding is indeed based on table look-ups.) The noisy inner product becomes:

$$\begin{aligned}
\langle v_i, s \rangle \oplus d_i &= \langle g'_i G, s \rangle \oplus \langle v_i - g_i, s \rangle \oplus d_i \\
&= \langle g'_i, sG^T \rangle \oplus \langle v_i - g_i, s \rangle \oplus d_i \\
&= \langle g'_i, s' \rangle \oplus d'_i,
\end{aligned}$$

where G is the generator matrix of the code, $g_i = g'_i G$, $s' = sG^T \in \{0,1\}^{k'}$ and $d'_i = \langle v_i - g_i, s \rangle \oplus d_i$. We denote by $\mathsf{bc} = E((-1)^{\langle v_i - g_i, s \rangle})$ the bias of $\langle v_i - g_i, s \rangle$. We will see in Section 5.4 how to construct a $[k, k']$ linear code making bc as large as possible.

Here, bc averages the bias over the secret although s is fixed by *sparse-secret*. It gives the correct average bias δ over the distribution of the key. We will see that it allows to approximate the expected probability of success of the algorithm.

By this transform, no query is lost.

> *code-reduce*(k, k', params) : k'; $n' = n$; $\delta' = \delta \cdot \mathsf{bc}$
> δ'_s depends on δ_s and G
> Complexity: $\mathcal{O}(kn)$

The way δ'_s is computed is a bit more complicated than for the other types of reductions. However, δ_s only plays a role in the *code-reduce* reduction, and we will not consider algorithms that use more than one *code-reduce* reduction.

It is easy to notice that with each reduction operation the number of queries decreases or the bias is getting smaller. In general, for solving LPN, one tries to lose as few queries as possible while maintaining a large bias. We will study in Chapter 5.5 what is a good combination of these reductions.

After applying the reduction steps, we assume we are left with an $\mathsf{LPN}_{k',\delta'}$ instance where we have n' queries. The original BKW algorithm was using a final solving technique based on majority decoding. Since the LF1 algorithm, we use a better solving technique based on the Walsh Hadamard Transform (WHT).

WHT recovers a block of the secret by computing the fast Walsh Hadamard transform on the function $f(x) = \sum_i 1_{v_i=x}(-1)^{\langle v_i,s\rangle \oplus d_i}$ (this analysis was presented also in Section 3.4). The Walsh-Hadamard transform is

$$\hat{f}(\nu) = \sum_x (-1)^{\langle \nu,x\rangle} f(x) = \sum_i (-1)^{\langle v_i,s+\nu\rangle \oplus d_i} .$$

For $\nu = s$, we have $\hat{f}(s) = \sum_i (-1)^{d_i}$. For a positive bias, we know that most of the noise bits are set to 0. It is the opposite when the bias is negative. So, $|\hat{f}(s)|$ is large and we suppose it is the largest value in the table of $\hat{f}$. Using the Central Limit Theorem we obtain

$$n' \geq (2\delta'^{-2} - 1) \cdot \left(\varphi^{-1}\left(1 - (1-\theta)^{\frac{1}{2^{k'}-1}}\right)\right)^2 . \tag{5.1}$$

WHT(k');

Requires $n' \geq (2\delta'^{-2} - 1) \cdot \left(\varphi^{-1}\left(1 - (1-\theta)^{\frac{1}{2^{k'}-1}}\right)\right)^2$

Complexity: $\mathcal{O}(k' 2^{k'} \frac{\log_2 n' + 1}{2} + k'n')$

As we can see in the formulas of each possible step, the computations of k', n', and of the complexity do not depend on the secret weight. Furthermore, the computation of biases is always linear. So, the correct average bias (over the distribution of the key made by the *sparse-secret* transform) is computed. Only the computation of the success probability is non-linear, but we discuss about this in the next section. As it only matters in WHT, we will see in Section 5.3 that the approximation is justified.

5.3 On Approximating the Probability of Success

Approximating n by using Central Limit Theorem. In order to approximate the number of queries needed to solve the LPN instance, we consider when the Walsh Hadamard Transform fails to give the correct secret. We first assume that the bias is positive. We have a failure when for another $\bar{s} \neq s$, we have that $\hat{f}(\bar{s}) > \hat{f}(s)$. Following the analysis from Section 3.4, we let $y = A'\bar{s}^T + c'^T$ and $d' = A's^T + c'^T$. We have $\hat{f}(\bar{s}) = \sum_i (-1)^{y_i} = n' - 2 \cdot \mathsf{HW}(y)$ and similarly, $\hat{f}(s) = n' - 2 \cdot \mathsf{HW}(d')$. So, $\hat{f}(\bar{s}) > \hat{f}(s)$

translates to $\mathsf{HW}(y) \leq \mathsf{HW}(d')$. Therefore

$$\Pr[\hat{f}(\bar{s}) > \hat{f}(s)] = \Pr\left[\sum_{i=1}^{n'} (y_i - d'_i) \leq 0\right].$$

For each $\bar{s}$, we take y as a uniformly distributed random vector and we let $\delta'(s)$ be the bias introduce with a fixed s for d'_i (we recall that our analysis computes $\delta' = E(\delta'(s))$ over the distribution of s). Let $X_1, \ldots, X_{n'}$ be random variable corresponding to $X_i = y_i - d'_i$. Since $E(y_i) = \frac{1}{2}$, $E(d'_i) = \frac{1}{2} - \frac{\delta'(s)}{2}$ and y_i and d'_i are independent, we have that $E(X_i) = \frac{\delta'(s)}{2}$ and $\mathsf{Var}(X_i) = \frac{2-\delta'(s)^2}{4}$. By using the Central Limit Theorem (See Theorem 2.6) we obtain that

$$\Pr[X_1 + \ldots + X_{n'} \leq 0] \approx \varphi\left(Z(s)\right) \text{ with } Z(s) = -\frac{\delta'(s)}{\sqrt{2 - \delta'(s)^2}}\sqrt{n'}$$

where φ can be calculated by $\varphi(x) = \frac{1}{2} + \frac{1}{2}\mathsf{erf}(\frac{x}{\sqrt{2}})$ and erf is the Gauss error function. For $\delta'(s) < 0$, the same analysis with $\hat{f}(\bar{s}) < \hat{f}(s)$ gives the same result. Applying the reasoning for any $s' \neq s$, we obtain that the failure probability is

$$p(s) = 1 - (1 - \varphi(Z(s)))^{2^{k'}-1}, \text{if } \delta'(s) > 0$$
$$\text{and } p(s) = 1 - \frac{1}{2^{k'}}, \text{if } \delta'(s) \leq 0.$$

We deduce the following (for $\theta < \frac{1}{2}$)

$$p(s) \leq \theta \Leftrightarrow \sqrt{n'} \geq -\sqrt{2\delta'(s)^{-2} - 1}\,\varphi^{-1}\left(1 - (1-\theta)^{\frac{1}{2^{k'}-1}}\right) \text{ and } \delta'(s) > 0\,.$$

As a condition for our WHT step, we adopt the inequality in which we replace $\delta'(s)$ by δ'. We give a heuristic argument below to show that it implies that $E(p(s)) \leq \theta$, which is what we want.

Note that if we use the approximation $\varphi\left(Z\right) \approx -\frac{1}{Z\sqrt{2\pi}}e^{-\frac{Z^2}{2}}$ for $Z \to -\infty$, we obtain the condition $n' \geq 2(2\delta'^{-2} - 1)\ln(\frac{2^{k'}-1}{\theta})$. So, our analysis brings an improvement of factor two over the Hoeffding bound method used by Bogos et al. [BTV16] that requires $n' \geq 8\delta'^{-2}\ln(\frac{2^{k'}}{\theta})$.

On the validity of using the bias average. The above computation is correct when using $\delta'(s)$ but we use $\delta' = E(\delta'(s))$ instead. If no *code-reduce* step is used, $\delta'(s)$ does not depend on s and we do have $\delta'(s) = \delta'$. However, when a *code-reduce* is used, the bias depends on the secret which is obtained after the *sparse-secret* step. For simplicity, we let s denote this secret. The bias $\delta'(s)$ is actually of form $\delta'(s) = \delta^{2^x}\mathsf{bc}(s)$ where x is the number of *xor-reduce* and *partition-reduce* steps and $\mathsf{bc}(s)$ is the bias introduced by *code-reduce* depending on s. The values of $\delta'(s)$, $Z(s)$, and $p(s)$ are already defined above. We define $Z = -\frac{\delta'}{\sqrt{2-\delta'^2}}\sqrt{n'}$ and $p = 1-(1-\varphi(Z))^{2^{k'}-1}$. Clearly, $E(p(s))$ is the average failure probability over the distribution of the secret obtained after *sparse-secret*.

Our method ensures that $\delta' = E(\delta'(s))$ over the distribution of s. Since δ' is typically small (after a few *xor-reduce* steps, δ^{2^x} is indeed very small), we can consider $Z(s)$ as a linear function of $\delta'(s)$ and have $E(Z(s)) \approx Z$. This is confirmed by experiment. We make the **heuristic approximation** that

$$E\left(1-(1-\varphi(Z(s)))^{2^{k'}-1}\right) \approx 1-(1-\varphi(E(Z(s))))^{2^{k'}-1} \approx 1-(1-\varphi(Z))^{2^{k'}-1}$$

So, $E(p(s)) \approx p$.[2]

We did some experiments based on some examples in order to validate our heuristic assumption. Our results show indeed that $E(Z(s)) \approx Z$. There is a small gap between $E(p(s))$ and p but this does not affect our results. Actually, we are in a phase transition region so any tiny change in the value of n' makes $E(p(s))$ change a lot. We include these results in Appendix A.2. Thus, ensuring that $p \leq \theta$ with the above analysis based on the average bias ensures that the expected failure probability is bounded by θ.

Both the choice of the codes and the secret have a big influence on our approximation. We believe that choosing larger codes can give us the best results. Splitting the reduction *code-reduce* in several small codes can give a smaller bias (as a multiplication of several biases) and in the worst case can give a negative or a 0 bias. In Appendix A.3 we prove that for a repetition codes $[k, 1]$, with k odd, for a secret with even Hamming weight, we always introduce a bias of 0. A larger code would eliminate such scenarios and could give a better bias.

5.4 Bias of the Code Reduction

In this section, we present how to compute the bias introduced by a *code-reduce*. Recall that the reduction *code-reduce*(k, k') introduces a new noise:

$$\langle v_i, s\rangle \oplus d_i = \langle g'_i, s'\rangle \oplus \langle v_i - g_i, s\rangle \oplus d_i,$$

where $g_i = g'_i G$ is the nearest codeword of v_i and $s' = sG^T$. Note that g_i is not necessarily unique, specially if the code is not perfect. We take $g_i = \mathsf{Decode}(v_i)$ obtained from an arbitrary decoding algorithm. Then the noise $\mathtt{bc}$ can be computed by the following formula:

[2]Note that Zhang et al. [ZJW16] implicitly does the same assumption as they use the average bias as well.

$$\begin{aligned}\mathsf{bc} = E((-1)^{\langle v_i - g_i, s\rangle}) &= \sum_{e\in\{0,1\}^k} \Pr[v_i - g_i = e] E((-1)^{\langle e,s\rangle}) \\ &= \sum_{w=0}^{k} \sum_{\substack{e\in\{0,1\}^k,\\ \mathsf{HW}(e)=w}} \Pr[v_i - g_i = e]\delta_s^w = E\left(\delta_s^{\mathsf{HW}(v_i-g_i)}\right)\end{aligned}$$

for a δ_s-sparse secret. (We recall that the *sparse-secret* reduction step randomizes the secret.) So, the probability space is over the distribution of v_i *and* the distribution of s. Later, we consider $\mathsf{bc}(s) = E((-1)^{\langle v_i - g_i, s\rangle})$ over the distribution over v_i only. (In the work of Guo et al. [GJL14], only $\mathsf{bc}(s)$ is considered. In Zhang et al. [ZJW16], our bc was also considered.) In the last expression of bc, we see that the ambiguity in decoding does not affect bc as long as the Hamming distance $\mathsf{HW}(v_i - \mathsf{Decode}(v_i))$ is not ambiguous. This is a big advantage of averaging in bc as it allows to use non-perfect codes. From this formula, we can see that the decoding algorithm $v_i \to g_i$ making $\mathsf{HW}(v_i - g_i)$ minimal makes bc maximal. In this case, we obtain

$$\mathsf{bc} = E\left(\delta_s^{d(v_i,C)}\right), \tag{5.2}$$

where C is the code and $d(v_i, C)$ denotes the Hamming distance of v_i from C.

We recall that for a perfect code C the covering radius is $\rho = \left\lfloor \frac{D-1}{2} \right\rfloor$ and for quasi-perfect code, we have $\rho = \left\lfloor \frac{D-1}{2} \right\rfloor + 1$, where D is the minimum distance of C (See Section 2.6).

Theorem 5.1. *We consider a $[k, k', D]$ linear code C, where k is the length, k' is the dimension, and D is the minimum distance. For any integer r and any positive bias δ_s, we have*

$$\mathsf{bc} \le 2^{k'-k} \sum_{w=0}^{r} \binom{k}{w} (\delta_s^w - \delta_s^{r+1}) + \delta_s^{r+1}$$

where bc is a function of δ_s defined by (5.2). Equality for any δ_s such that $0 < \delta_s < 1$ implies that C is perfect or quasi-perfect. In that case, the equality is reached when taking the packing radius $r = R = \left\lfloor \frac{D-1}{2} \right\rfloor$.

By taking r as the largest integer such that $\sum_{w=0}^{r} \binom{k}{w} \le 2^{k-k'}$ (which is the packing radius $R = \left\lfloor \frac{D-1}{2} \right\rfloor$ for perfect and quasi-perfect codes), we can see that if a perfect $[k, k']$ code exists, it makes bc maximal. Otherwise, if a quasi-perfect $[k, k']$ code exists, it makes bc maximal.

Proof. Let decode be an optimal deterministic decoding algorithm. The formula gives us that

$$\mathsf{bc} = 2^{-k} \sum_{g \in C} \sum_{v \in \mathsf{decode}^{-1}(g)} \delta_s^{\mathsf{HW}(v-g)} .$$

We define $\mathsf{decode}_w^{-1}(g) = \{v \in \mathsf{decode}^{-1}(g); \mathsf{HW}(v-g) = w\}$ and $\mathsf{decode}_{>r}^{-1}(g)$ the union of all $\mathsf{decode}_w^{-1}(g)$ for $w > r$. For all r, we have

$$\begin{aligned}
&\sum_{v \in \mathsf{decode}^{-1}(g)} \delta_s^{\mathsf{HW}(v-g)} \\
=& \sum_{w=0}^{r} \binom{k}{w} \delta_s^w + \sum_{w=0}^{r} \left(\#\mathsf{decode}_w^{-1}(g) - \binom{k}{w} \right) \delta_s^w + \sum_{w>r} \delta_s^w \#\mathsf{decode}_w^{-1}(g) \\
\leq& \sum_{w=0}^{r} \binom{k}{w} \delta_s^w + \sum_{w=0}^{r} \left(\#\mathsf{decode}_w^{-1}(g) - \binom{k}{w} \right) \delta_s^w + \delta_s^{r+1} \#\mathsf{decode}_{>r}^{-1}(g) \\
\leq& \sum_{w=0}^{r} \binom{k}{w} \delta_s^w + \delta_s^{r+1} \left(\#\mathsf{decode}^{-1}(g) - \sum_{w=0}^{r} \binom{k}{w} \right)
\end{aligned}$$

where we used $\delta_s^w \leq \delta_s^{r+1}$ for $w > r$, $\#\mathsf{decode}_w^{-1}(g) \leq \binom{k}{w}$ and $\delta_s^w \geq \delta_s^{r+1}$ for $w \leq r$. We further have equality if and only if the ball centered on g of radius r is included in $\mathsf{decode}^{-1}(g)$ and the ball of radius $r+1$ contains $\mathsf{decode}^{-1}(g)$. By summing over all $g \in C$, we obtain the result.

So, the equality case implies that the packing radius is at least r and the covering radius is at most $r+1$. Hence, the code is perfect or quasi-perfect. Conversely, if the code is perfect or quasi-perfect and r is the packing radius, we do have equality.

□

So, for quasi-perfect codes, we can compute

$$\mathsf{bc} = 2^{k'-k} \sum_{w=0}^{R} \binom{k}{w} (\delta_s^w - \delta_s^{R+1}) + \delta_s^{R+1} \tag{5.3}$$

with $R = \left\lfloor \frac{D-1}{2} \right\rfloor$. For perfect codes, the formula simplifies to

$$\mathsf{bc} = 2^{k'-k} \sum_{w=0}^{R} \binom{k}{w} \delta_s^w . \tag{5.4}$$

5.4.1 Bias of a Repetition Code

Given a $[k, 1]$ repetition code, the optimal decoding algorithm is the majority decoding. We have $D = k$, $k' = 1$, $R = \left\lfloor \frac{k-1}{2} \right\rfloor$. For k odd, the code is perfect so $\rho = R$. For k

even, the code is quasi-perfect so $\rho = R + 1$. Using (5.3) and (5.4), we obtain

$$\mathsf{bc} = \begin{cases} \sum_{w=0}^{\frac{k-1}{2}} \frac{1}{2^{k-1}} \binom{k}{w} \delta_s^w & \text{if } k \text{ is odd} \\ \\ \sum_{w=0}^{\frac{k}{2}-1} \frac{1}{2^{k-1}} \binom{k}{w} \delta_s^w + \frac{1}{2^k} \binom{k}{k/2} \delta_s^{\frac{k}{2}} & \text{if } k \text{ is even} \end{cases}$$

We give below the biases obtained for some $[k, 1]$ repetition codes.

$[k, 1]$	bias
$[2, 1]$	$\frac{1}{2}\delta_s + \frac{1}{2}$
$[3, 1]$	$\frac{3}{4}\delta_s + \frac{1}{4}$
$[4, 1]$	$\frac{3}{8}\delta_s^2 + \frac{1}{2}\delta_s + \frac{1}{8}$
$[5, 1]$	$\frac{5}{8}\delta_s^2 + \frac{5}{16}\delta_s + \frac{1}{16}$
$[6, 1]$	$\frac{5}{16}\delta_s^3 + \frac{15}{32}\delta_s^2 + \frac{3}{16}\delta_s + \frac{1}{32}$
$[7, 1]$	$\frac{35}{64}\delta_s^3 + \frac{21}{64}\delta_s^2 + \frac{7}{64}\delta_s + \frac{1}{64}$
$[8, 1]$	$\frac{35}{128}\delta_s^4 + \frac{7}{16}\delta_s^3 + \frac{7}{32}\delta_s^2 + \frac{1}{16}\delta_s + \frac{1}{128}$
$[9, 1]$	$\frac{63}{128}\delta_s^4 + \frac{21}{64}\delta_s^3 + \frac{9}{64}\delta_s^2 + \frac{9}{256}\delta_s + \frac{1}{256}$
$[10, 1]$	$\frac{63}{256}\delta_s^5 + \frac{105}{256}\delta_s^4 + \frac{15}{64}\delta_s^3 + \frac{45}{512}\delta_s^2 + \frac{5}{256}\delta_s + \frac{1}{512}$

5.4.2 Bias of a Perfect Code

In previous works [GJL14, ZJW16], the authors assume a perfect code. In this case, $\sum_{w=0}^{R} \binom{k}{w} = 2^{k-k'}$ and we can use (5.4) to compute bc. There are not so many binary linear codes which are perfect. Except the repetition codes with odd length, the only ones are the trivial codes $[k, k, 1]$ with $R = \rho = 0$ and $\mathsf{bc} = 1$, the Hamming codes $[2^\ell - 1, 2^\ell - \ell - 1, 3]$ for $\ell \geq 2$ with $R = \rho = 1$, and the Golay code $[23, 12, 7]$ with $R = \rho = 3$.

For the Hamming codes, we have

$$\mathsf{bc} = 2^{-\ell} \sum_{w=0}^{1} \binom{2^\ell - 1}{w} \delta_s^w = \frac{1 + (2^\ell - 1)\delta_s}{2^\ell} .$$

For the Golay code, we obtain

$$\mathsf{bc} = 2^{-11} \sum_{w=0}^{3} \binom{23}{w} \delta_s^w = \frac{1 + 23\delta_s + 253\delta_s^2 + 1771\delta_s^3}{2^{11}} .$$

Previously [BTV16, GJL14], the value bc_w of $\mathsf{bc}(s)$ for any s of Hamming weight w was approximated to

$$\mathsf{bc}_w = 1 - 2\frac{1}{S(k, \rho)} \sum_{\substack{i \leq \rho, \\ i \text{ odd}}} \binom{w}{i} S(k - w, \rho - i),$$

where w is the Hamming weight of the k-bit secret and $S(k', \rho)$ is the number of k'-bit strings with weight at most ρ. Intuitively, the formula counts the number of $v_i - g_i$ that produce an odd number of xor with the 1's of the secret (See 3.2 and [BTV16, GJL14]). So, Guo et al. [GJL14] assumes a fixed value for the weight w of the secret and considers the probability that w is not correct. If w is lower, the actual bias is larger but if w is larger, the computed bias is overestimated and the algorithm fails.

For instance, with a $[3, 1]$ repetition code, the correct bias is $\mathsf{bc} = \frac{3}{4}\delta_s + \frac{1}{4}$ following our formula. With a fixed w, it is of $\mathsf{bc}_w = 1 - \frac{w}{2}$ [BTV16, GJL14]. The probability of w to be correct is $\binom{k}{w}\tau^w(1-\tau)^{k-w}$. We take the example of $\tau = \frac{1}{3}$ so that $\delta_s = \frac{1}{3}$.

w	bc_w	$\Pr[w]$	$\Pr[w], \tau = \frac{1}{3}$
0	1	$(1-\tau)^3$	0.2963
1	$\frac{1}{2}$	$3\tau(1-\tau)^2$	0.4444
2	0	$3\tau^2(1-\tau)$	0.2222
3	$-\frac{1}{2}$	τ^3	0.0370

So, by taking $w = 1$, we have $\delta = \mathsf{bc}_w = \frac{1}{2}$ but the probability of failure is about $\frac{1}{4}$. Our approach uses the average bias $\delta = \mathsf{bc} = \frac{1}{2}$.

5.4.3 Using Quasi-Perfect Codes

If C' is a $[k-1, k', D]$ perfect code with $k' > 1$ and if there exists some codewords of odd length, we can extend C', i.e., add a parity bit and obtain a $[k, k']$ code C. Clearly, the packing radius of C is at least $\lfloor \frac{D-1}{2} \rfloor$ and the covering radius is at most $\lfloor \frac{D-1}{2} \rfloor + 1$. For $k' > 1$, there is up to one possible length for making a perfect code of dimension k'. So, C is a quasi-perfect code, its packing radius is $\lfloor \frac{D-1}{2} \rfloor$ and its covering radius is $\lfloor \frac{D-1}{2} \rfloor + 1$.

If C' is a $[k+1, k', D]$ perfect code with $k' > 1$, we can puncture it, i.e., remove one coordinate by removing one column from the generating matrix. If we chose to remove a column which does not modify the rank k', we obtain a $[k, k']$ code C. Clearly, the packing radius of C is at least $\lfloor \frac{D-1}{2} \rfloor - 1$ and the covering radius is at most $\lfloor \frac{D-1}{2} \rfloor$. For $k' > 1$, there is up to one possible length for making a perfect code of dimension k'. So, C is a quasi-perfect code, its packing radius is $\lfloor \frac{D-1}{2} \rfloor - 1$ and its covering radius is $\lfloor \frac{D-1}{2} \rfloor$.

Hence, we can use extended Hamming codes $[2^\ell, 2^\ell - \ell - 1]$ with packing radius 1 for $\ell \geq 3$, punctured Hamming codes $[2^\ell - 2, 2^\ell - \ell - 1]$ with packing radius 0 for $\ell \geq 3$, the extended Golay code $[24, 12]$ with packing radius 3, and the punctured Golay code $[22, 12]$ with packing radius 2.

There actually exist many constructions for quasi-perfect linear binary codes. We list a few in Table 5.1. We took codes listed in the existing literature [CKJS85, Table 1], [PW72, p. 122], [GS85, p. 47], [EM05, Table 1], [CHLL97, p. 313], and [BBDF08, Table I]. In Table 5.1, k, k', D, and R denote the length, the dimension, the minimum distance, and the packing radius, respectively.

Table 5.1: Perfect and Quasi-Perfect Binary Linear Codes

name	type	$[k, k', D]$	R	comment	ref.
	P	$[k, k, 1]$, $k \geq 1$	0	$[*, \ldots, *]$	
r	P	$[k, 1, k]$, k odd	$\frac{k-1}{2}$	repetition code	
H	P	$[2^\ell - 1, 2^\ell - \ell - 1, 3]$, $\ell \geq 3$,	1	Hamming code	
G	P	$[23, 12, 7]$	3	Golay code	
	QP	$[k, k-1, 1]$	0	$[*, \ldots, *, 0]$	
r	QP	$[k, 1, k]$, k even	$\frac{k}{2} - 1$	repetition code	
eG	QP	$[24, 12, 8]$	3	extended Golay code	
pG	QP	$[22, 12, 6]$	2	punctured Golay code	
eH	QP	$[2^\ell, 2^\ell - \ell - 1, 4]$, $\ell \geq 2$	1	extended Hamming code	
	QP	$[2^\ell - 1, 2^\ell - \ell, 1]$, $\ell \geq 2$,	0	Hamming with an extra word	
pH	QP	$[2^\ell - 2, 2^\ell - \ell - 1, 2]$, $\ell \geq 2$	0	punctured Hamming	
HxH	QP	$[2 * (2^\ell - 1), 2 * (2^\ell - \ell - 1)]$, $\ell \geq 2$	1	Hamming $\times$ Hamming	[CKJS85]
upack	QP	$[2^\ell - 2, 2^\ell - \ell - 2, 3]$, $\ell \geq 3$	1	uniformly packed	[CKJS85]
2BCH	QP	$[2^\ell - 1, (2^\ell - 1) - (2 * \ell)]$, $\ell \geq 3$	2	2-e.c. BCH	[CKJS85]
Z	QP	$[2^\ell + 1, (2^\ell + 1) - (2 * \ell)]$, $\ell > 3$ even	2	Zetterberg	[CKJS85]
rGop	QP	$[2^\ell - 2, (2^\ell - 2) - (2 * \ell)]$, $\ell > 3$ even	2	red. Goppa	[CKJS85]
iGop	QP	$[2^\ell, (2^\ell) - (2 * \ell)]$, $\ell > 2$ odd	2	irred. Goppa	[CKJS85]
Mclas	QP	$[2^\ell - 1, (2^\ell - 1) - 2 * \ell]$, $\ell > 2$ odd	2	Mclas	[CKJS85]
S	QP	$[5, 2]$, $[9, 5]$, $[10, 5]$, $[11, 6]$	1	Slepian	[PW72]
S	QP	$[11, 4]$	2	Slepian	[PW72]
FP	QP	$[15, 9]$, $[21, 14]$, $[22, 15]$, $[23, 16]$	1	Fontaine-Peterson	[PW72]
W	QP	$[19, 10]$, $[20, 11]$, $[20, 13]$, $[23, 14]$	2	Wagner	[PW72]
P	QP	$[21, 12]$	2	Prange	[PW72]
FP	QP	$[25, 12]$	3	Fontaine-Peterson	[PW72]
W	QP	$[25, 15]$, $[26, 16]$, $[27, 17]$, $[28, 18]$, $[29, 19]$, $[30, 20]$, $[31, 20]$	1	Wagner	[PW72]
GS	QP	$[13, 7]$, $[19, 12]$	1	GS85	[GS85]
BBD	QP	$[7, 3, 3]$, $[9, 4, 4]$, $[10, 6, 3]$, $[11, 7, 3]$, $[12, 7, 3]$, $[12, 8, 3]$, $[13, 8, 3]$, $[13, 9, 3]$, $[14, 9, 3]$, $[15, 10, 3]$, $[16, 10, 3]$, $[17, 11, 4]$, $[17, 12, 3]$, $[18, 12, 4]$, $[18, 13, 3]$, $[19, 13, 3]$, $[19, 14, 3]$, $[20, 14, 4]$	1	BBD08	[BBDF08]
BBD	QP	$[22, 13, 5]$	2	BBD08	[BBDF08]

5.4.4 Finding the Optimal Concatenated Code

The linear code $[k, k']$ is typically instantiated by a concatenation of elementary codes for practical purposes. By "concatenation" of m codes $C_1, \dots, C_m$, we mean the code formed by all $g_{i,1} \| \cdots \| g_{i,m}$ obtained by concatenating any set of $g_{i,j} \in C_j$. Decoding $v_1 \| \cdots \| v_m$ is based on decoding each $v_{i,j}$ in C_j independently. If all C_j are small, this is done by a table lookup. So, concatenated codes are easy to implement and to decode. For $[k, k']$ we have the concatenation of $[k_1, k'_1], \dots, [k_m, k'_m]$ codes, where $k_1 + \cdots + k_m = k$ and $k'_1 + \cdots + k'_m = k'$. Let v_{ij}, g_{ij}, s'_j denote the j^{th} part of v_i, g_i, s' respectively, corresponding to the concatenated $[k_j, k'_j]$ code. The bias of $\langle v_{ij} - g_{ij}, s_j \rangle$ in the code $[k_j, k'_j]$ is denoted by bc_j. As $\langle v_i - g_i, s \rangle$ is the xor of all $\langle v_{ij} - g_{ij}, s_j \rangle$, the total bias introduced by this operation is computed as $\mathsf{bc} = \prod_{j=1}^{m} \mathsf{bc}_j$ and the combination $\mathsf{params} = ([k_1, k'_1], \dots, [k_m, k'_m])$ is chosen such that it gives the highest bias.

The way these params are computed is the following: we start by computing the biases for all elementary codes. I.e. we compute the biases for all codes from Table 5.1. We may add random codes that we found interesting. (For these, we use (5.2) to compute bc.)[3] Next, for each $[i, j]$ code, we check to see if there is a combination of $[i - n, j - m]$,$[n, m]$ codes that give a better bias, where $[n, m]$ is either a repetition code, a Golay code or a Hamming code. We illustrate below the algorithm to find the optimal concatenated code. This algorithm was independently proposed by Zhang et al. [ZJW16] (with perfect codes only).

Algorithm 5.1 Finding the optimal params and bias

```
1: Input: $k$
2: Output: table for the optimal bias for each $[i, j]$ code, $1 \leq j < i \leq k$

3: initialize all $\mathsf{bias}(i, j) = 0$
4: initialize $\mathsf{bias}(1, 1) = 1$
5: initialize the bias for all elementary codes
6: for all $j : 2$ to $k$ do
7:     for all $i : j + 1$ to $k$ do
8:         for all elementary code $[n, m]$ do
9:             if $|\mathsf{bias}(i - n, j - m) \cdot \mathsf{bias}(n, m)| > |\mathsf{bias}(i, j)|$ then
10:                $\mathsf{bias}(i, j) = \mathsf{bias}(i - n, j - m) \cdot \mathsf{bias}(n, m)$
11:                $\mathsf{params}(i, j) = \mathsf{params}(i - n, j - m) \cup \mathsf{params}(n, m)$
12:            end if
13:        end for
14:    end for
15: end for
```

Using $\mathcal{O}(k)$ elementary codes, this procedure takes $\mathcal{O}(k^3)$ time and we can store all params for any combination $[i, j]$, $1 \leq j < i \leq k$ with $\mathcal{O}(k^2)$ memory.

[3]The random codes that we used are provided in Appendix A.4 of the thesis.

5.5 The Graph of Reduction Steps

Having in mind the reduction methods described in Section 5.2, we formalize an LPN solving algorithm in terms of finding the best chain in a graph. The intuition is the following: in an LPN solving algorithm, we can see each reduction step as an edge from a $(k, \log_2 n)$ instance to a new instance $(k', \log_2 n')$ where the secret is smaller, $k' \leq k$, we have more or less queries and the noise has a different bias. For example, a *xor-reduce*(b) reduction turns an $(k, \log_2 n)$ instance with bias δ into $(k', \log_2 n')$ with bias δ' where $k' = k - b$, $n' = \frac{n(n-1)}{2^{b+1}}$ and $\delta' = \delta^2$. By this representation, the reduction phase represents a chain in which each edge is a reduction type moving from LPN with parameters (k, n) to LPN with parameters (k', n') and that ends with an instance (k_i, n_i) used to recover the k_i-bit length secret by a solving method. The chain terminates by the fast Walsh-Hadamard solving method.

We formalize the reduction phase as a chain of reduction steps in a graph $G = (V, E)$. The set of vertices V is composed of $V = \{1, \dots, k\} \times L$ where L is a set of real numbers. For instance, we could take $L = \mathbb{R}$ or $L = \mathbb{N}$. For efficiency reasons, we could even take $L = \{0, \dots, \eta\}$ for some bound η. Every vertex saves the size of the secret and the logarithmic number of queries; i.e. a vertex $(k, \log_2 n)$ means that we are in an instance where the size of the secret is k and the number of queries available is n. An edge from one vertex to another is given by a reduction step. An edge from $(k, \log_2 n)$ to a $(k', \log_2 n')$ has a label indicating the type of reduction and its parameters (e.g. *xor-reduce*$(k - k')$ or *code-reduce*(k, k', params)). This reduction defines some α and β coefficients such that the bias δ' after reduction is obtained from the bias δ before the reduction by

$$\log_2 \delta'^2 = \alpha \log_2 \delta^2 + \beta$$

where $\alpha, \beta \in \mathbb{R}$.

We denote by $\lceil \lambda \rceil_L$ the smallest element of L which is at least equal to λ and by $\lfloor \lambda \rfloor_L$ the largest element of L which is not larger than λ. In general, we could use a rounding function $\mathsf{Round}_L(\lambda)$ such that $\mathsf{Round}_L(\lambda)$ is in L and approximates λ.

The reduction steps described in Section 5.2 can be formalized as follows:

- *sparse-secret*: $(k, \log_2 n) \to (k, \mathsf{Round}_L(\log_2 (n - k)))$ and $\alpha = 1, \beta = 0$
- *partition-reduce*(b): $(k, \log_2 n) \to (k - b, \mathsf{Round}_L(\log_2 (n - 2^b)))$ and $\alpha = 2, \beta = 0$
- *xor-reduce*(b): $(k, \log_2 n) \to (k - b, \mathsf{Round}_L(\log_2 \left(\frac{n(n-1)}{2^{b+1}}\right)))$ and $\alpha = 2, \beta = 0$
- *drop-reduce*(b): $(k, \log_2 n) \to (k - b, \mathsf{Round}_L(\log_2 (\frac{n}{2^b})))$ and $\alpha = 1, \beta = 0$
- *code-reduce*(k, k', params): $(k, \log_2 n) \to (k', \log_2 n)$ and $\alpha = 1, \beta = \log_2 \mathsf{bc}^2$, where bc is the bias introduced by the covering code reduction using a $[k, k']$ linear code defined by params.

Below, we give the formal definition of a reduction chain.

Definition 5.2 (Reduction chain)**.** *Let*

$$\mathcal{R} = \{\textit{sparse-secret}, \textit{partition-reduce}(b), \textit{xor-reduce}(b), \textit{drop-reduce}(b), \textit{code-reduce}(k, k', \mathsf{params})\}$$

for $k, k', b \in \mathbb{N}$ *. A* ***reduction chain*** *is a sequence*

$$(k_0, \log_2 n_0) \xrightarrow{e_1} (k_1, \log_2 n_1) \xrightarrow{e_2} \ldots \xrightarrow{e_i} (k_i, \log_2 n_i),$$

where the change $(k_{j-1}, \log_2 n_{j-1}) \to (k_j, \log_2 n_j)$ *is performed by one reduction from* $\mathcal{R}$*, for all* $0 < j \leq i$*.*

A chain is ***simple*** *if it is accepted by the automaton from Figure 5.1* [4]*.*

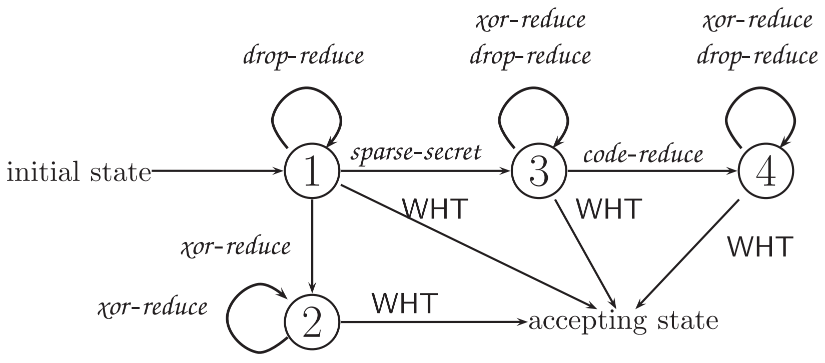

Figure 5.1: Automaton accepting simple chains

Remark: Restrictions for simple chains are modelled by the automaton in Figure 5.1. We restrict to simple chains as they are easier to analyze. Indeed, *sparse-secret* is only used to raise δ_s to make *code-reduce* more effective. And, so far, it is hard to analyze sequences of *code-reduce* steps as the first one may destroy the uniform and high δ_s for the next ones. This is why we exclude multiple *code-reduce* reductions in a simple chain. So, we use up to one *sparse-secret* reduction, always one before *code-reduce*. And *sparse-secret* occurs before δ decreases. For convenience, we will add a state of the automaton to the vertex in V.

Definition 5.3 (Exact chain)**.** *An* ***exact chain*** *is a simple reduction chain for* $L = \mathbb{R}$*. I.e.* Round_L *is the identity function.*

A chain which is not exact is called **rounded**.

For solving LPN we are interested in those chains that end with a vertex $(k_i, \log_2 n_i)$ which allows to call a WHT solving algorithm to recover the k_i-bit secret. We call these chains valid chains and we define them below.

[4]We simplify the automaton by removing any *partition-reduce*; *xor-reduce* is always better than *partition-reduce*.

Definition 5.4 (Valid reduction chain). *Let*

$$(k_0, \log_2 n_0) \xrightarrow{e_1} (k_1, \log_2 n_1) \xrightarrow{e_2} \cdots \xrightarrow{e_i} (k_i, \log_2 n_i)$$

be a reduction chain with $e_j = (\alpha_j, \beta_j, .)$. *Let* δ_j *be the bias corresponding to the vertex* $(k_j, \log_2 n_j)$ *iteratively defined by* $\delta_0 = \delta$ *and* $\log_2 \delta_j^2 = \alpha_j \log_2 \delta_{j-1}^2 + \beta_j$ *for* $j = 1, \ldots, i$. *We say the chain is a* ***θ-valid reduction chain*** *if* n_i *satisfies (5.1) from page 59 for* $\delta' = \delta_i$ *and* $n' = n_i$.

The *time complexity* of a chain $(e_1, \ldots, e_i)$ is simply the sum of the complexity of each reduction step $e_1, e_2, \ldots, e_i$ and WHT. We further define the **max-complexity** of a chain which is the maximum of the complexity of each reduction step and WHT. The max-complexity is a good approximation of the complexity. Our goal is to find a chain with optimal complexity. What we achieve is that, *given a set L*, we find a *rounded* chain with optimal *max-complexity* up to some given precision.

5.5.1 Towards Finding the Best LPN Reduction Chain

In this subsection, we present the algorithm that helps finding the optimal valid chains for solving LPN. As aforementioned, we try to find the valid chain with optimal max-complexity for solving an $\mathsf{LPN}_{k,\tau}$ instance in our graph G.

The first step of the algorithm is to construct the directed graph $G = (V, E)$. We take the set of vertices $V = \{1, \ldots, k\} \times L \times \{1, 2, 3, 4\}$ which indicate the size of the secret, the logarithmic number of queries and the state in the automaton in Figure 5.1. Each edge $e \in E$ represents a reduction step and is labelled with the following information: $(k_1, \log_2 n_1, st) \overset{\alpha,\beta,t}{\rightarrow} (k_2, \log_2 n_2, st')$ where t is one of the reduction steps and α and β save information about how the bias is affected by this reduction step.

The graph has $\mathcal{O}(k \cdot |L|)$ vertices and each vertex has $\mathcal{O}(k)$ edges. So, the size of the graph is $\mathcal{O}(k^2 \cdot |L|)$.

Thus, we construct the graph G with all possible reduction steps and from it, we try to see what is the optimal simple rounded chain in terms of max-complexity. We present in Algorithm 5.2 the procedure to construct the graph G that contains all possible reduction steps with a time complexity bounded by 2^η (As explained below, Algorithm 5.2 is not really used).

The procedure of finding the optimal valid chain is illustrated in Algorithm 5.3. The procedure of finding a chain with upper bounded max-complexity is illustrated in Algorithm 5.4.

Algorithm 5.4 receives as input the parameters k and τ for the LPN instance, the parameter θ which represents the bound on the failure probability in recovering the secret. Parameter η represents an upper bound for the logarithmic complexity of each reduction step. Given η, we build the graph G which contains all possible reductions with time complexity smaller than 2^η (Step 4). Note that we do not really call Algorithm 5.2. Indeed, we do not need to store the edges of the graph. We actually keep a way to

Algorithm 5.2 Construction of graph G

```
1: Input: $k, \tau, L, \eta$
2: Output: graph $G = (V, E)$ containing all the reduction steps that have a complexity smaller than $2^\eta$

3: $V = \{1, \ldots, k\} \times L \times \{1, \ldots, 4\}$
4: $E$ is the set of all $((i, \eta_1, st), (j, \eta_2, st'))$ labelled by $(\alpha, \beta, t)$ such that there is a $st \xrightarrow{t} st'$ transition in the automaton and for
5: $t =$ sparse-secret:
6: for all $\eta : 1$ such that $\mathsf{lcomp} \leq \eta$ do set the edge
7:     where $i = k$, $(j, \eta_2) = (i, \mathsf{Round}_L(\log_2(2^{\eta_1} - i)))$, $\alpha = 1$, $\beta = 0$, $\mathsf{lcomp} = \min_x \log_2(\frac{(2^{\eta_1}-i)i^2}{log_2 i - log_2(log_2 i)} + i^2, i(2^{\eta_1} - i)\lceil \frac{i}{2^x} \rceil + i^3 + i2^{x+2^x})$
8: end for
9: $t =$ partition-reduce:
10: for all $(i, \eta_1, b)$ such that $b \geq 1$ and $\mathsf{lcomp} \leq \eta$ do set the edge
11:     where $(j, \eta_2) = (i - b, \mathsf{Round}_L(\log_2(\eta_1 - 2^b)))$, $\alpha = 2$, $\beta = 0$, $\mathsf{lcomp} = \log_2 i + \eta_1$
12: end for
13: $t =$ xor-reduce:
14: for all $(i, \eta_1, b)$ such that $b \geq 1$ and $\mathsf{lcomp} \leq \eta$ do set the edge
15:     where $(j, \eta_2) = (i - b, \mathsf{Round}_L(\eta_1 - 1 + \log_2(\frac{2^{\eta_1}-1}{2^b})))$, $\alpha = 2$, $\beta = 0$, $\mathsf{lcomp} = \log_2 i + \max(\eta_1, \eta_2)$
16: end for
17: $t =$ drop-reduce:
18: for all $(i, \eta_1, b)$ such that $b \geq 1$ and $\mathsf{lcomp} \leq \eta$ do set the edge
19:     where $(j, \eta_2) = (i - b, \mathsf{Round}_L(\eta_1 - b))$, $\alpha = 1$, $\beta = 0$, $\mathsf{lcomp} = \log_2 b + \eta_1$
20: end for
21: $t =$ code-reduce:
22: for all $(i, \eta_1, j)$ such that $j < i$ and $\mathsf{lcomp} \leq \eta$ do set the edge
23:     where $\eta_2 = \eta_1$, $\alpha = 1$, $\beta = \log_2 \mathsf{bc}^2$, $\mathsf{lcomp} = \log_2 i + \eta_1$, $\mathsf{bc}$ is the bias from the optimal $[i, j]$ code
24: end for
```

enumerate all edges going to a given vertex (in Step 12) by using the rules described in Algorithm 5.2.

For each vertex, we iteratively define Δ^{st} and Best^{st}, the best reduction step to reach a vertex and the value of the corresponding error bias. The best reduction step is the one that maximizes the bias. We define these values iteratively until we reach a vertex from which the WHT solving algorithm succeeds with complexity bounded by 2^η. Once we have reached this vertex, we construct the chain by going backwards, following the Best pointers.

We easily prove what follows by induction.

Lemma 5.5. *At the end of the iteration of Algorithm 5.4 for (j, η_2, st'), $\Delta^{st'}_{j,\eta_2}$ is the maximum of $\log_2 \delta^2$, where δ is the bias obtained by a Round_L-rounded simple chain from a vertex of form $(k, \eta_1, 0)$ to (j, η_2, st') with max-complexity bounded by 2^η ($\Delta^{st'}_{j,\eta_2} = -\infty$*

Algorithm 5.3 Search for a rounded chain with optimal max-complexity

```
1: Input: $k, \tau, \theta, \mathsf{precision}$
2: Output: a valid simple rounded chain in which rounding uses a given precision

3: set found = bruteforce                          ▷ found is the best found algorithm
4: set increment = $k$
5: set $\eta = k$                                  ▷ $2^\eta$ is a bound on the max-complexity
6: repeat
7:     set increment ← $\frac{1}{2}$increment
8:     define $L = \{0, \mathsf{precision}, 2 \times \mathsf{precision}, \ldots\} \cap [0, \eta - \mathsf{increment}]$
9:     run (out, success) = Search($k, \tau, \theta, L, \eta - \mathsf{increment}$) with Algorithm 5.4
10:    if success then
11:        set found = out
12:        set $\eta = \eta - \mathsf{increment}$
13:    end if
14: until increment ≤ precision
15: output found
```

if there is no such chain).

Lemma 5.6. *If there exists a simple* Round_L*-rounded chain c ending on state* (k_j, η_j, st_j) *and max-complexity bounded by* 2^η*, there exists one* c' *such that* $\Delta^{st_i}_{i,\eta_i} = \log_2 \delta_i^2$ *at each step.*

Proof. Let c'' be a simple chain ending on (k_j, η_j, st_j) with $\Delta^{st_j}_{j,\eta_j} = \log_2 \delta_j^2$.
Let $(k_{j-1}, \eta_{j-1}, st_{j-1})$ be the preceding vertex in c''. We apply Lemma 5.6 on this vertex by induction to obtain a chain c'''. Since the complexity of the last edge does not depend on the bias and $\alpha \geq 0$ in the last edge, we construct the chain c', by concatenating c''' with the last edge of c''. □

Theorem 5.7. *Algorithm 5.4 finds a θ-valid simple* Round_L*-rounded chain for* $\mathsf{LPN}_{k,\tau}$ *with max-complexity bounded by* 2^η *if there exists one.*

Proof. We use Lemma 5.6 and the fact that increasing δ^2 keeps constraint (5.1) valid. □

If we used $L = \mathbb{R}$, Algorithm 5.4 would always find a valid simple chain with bounded max-complexity when it exists. Instead, we use rounded chains and hope that rounding still makes us find the optimal chain.

So, we build Algorithm 5.3. In this algorithm, we look for the minimal η for which Algorithm 5.4 returns something by a divide and conquer algorithm. First, we set η as being in the interval $[0, k]$ where the solution for $\eta = k$ corresponds to a brute-force search. Then, we cut the interval in two pieces and see if the lower interval has a solution. If it does, we iterate in this interval. Otherwise, we iterate in the other interval. We stop once the amplitude of the interval is lower than the requested precision. The complexity of Algorithm 5.3 is of $\log_2 \frac{k}{\mathsf{precision}}$ calls to Algorithm 5.4.

Algorithm 5.4 Search for a best LPN reduction chain with max-complexity bounded to η

```
1: **Input**: $k, \tau, \theta, L, \eta$
2: **Output**: a valid simple rounded chain with max-complexity bounded to $\eta$

3: $\delta = 1 - 2\tau$
4: Construct the graph $G$ using Algorithm 5.2 with parameters $k, \tau, L, \eta$
5: **for all** $\eta_1 \in L$ **do**
6:     set $\Delta^0_{k,\eta_1} = \log_2 \delta^2$, $\mathsf{Best}^0_{k,\eta_1} = \bot$
7:     set $\Delta^{st}_{k,\eta_1} = -\infty$, $\mathsf{Best}^{st}_{k,\eta_1} = \bot$    ▷ $\Delta^{st}$ stores the best bias for a vertex $(k, \eta_1, st)$ in a chain, and $\mathsf{Best}^{st}$ is the edge ending to this vertex in this chain
8: **end for**
9: **for** $j : k$ downto 1 **do**    ▷ Search for the optimal chain
10:     **for** $\eta_2 \in L$ in decreasing order **do**
11:         set $\Delta^{st}_{j,\eta_2} = 0$, $\mathsf{Best}^{st}_{j,n_2} = \bot$ for all $st$
12:         **foreach** st' and each edge $e$ to $(j, \eta_2, st')$
13:             set $(i, \eta_1, st)$ to the origin of $e$ and $\alpha$ and $\beta$ as defined by $e$
14:             **if** $\alpha\Delta^{st}_{i,\eta_1} + \beta \geq \Delta^{st'}_{j,\eta_2}$ **then** set $\Delta^{st'}_{j,\eta_2} = \alpha\Delta^{st}_{i,\eta_1} + \beta$, $\mathsf{Best}^{st'}_{j,n_2} = e$
15:             **end if**
16:         **end foreach**
17:         **if** $\eta_2 > 1 - \Delta^{st'}_{j,\eta_2} + 2\log_2\left(-\varphi^{-1}(1-(1-\theta)^{\frac{1}{2^j-1}})\right)$ and $j + \log_2 j \leq \eta$ **then**
18:             Construct the chain $c$ ending by $\mathsf{Best}^{st'}_{j,\eta_2}$ and output $(c, \mathsf{true})$
19:         **end if**
20:     **end for**
21: **end for**
22: output $(\bot, \mathsf{false})$
```

Theorem 5.8. *Algorithm 5.3 finds a θ-valid simple* Round_L*-rounded chain for* $\mathsf{LPN}_{k,\tau}$ *with parameter* $\mathsf{precision}$*, with optimal rounded max-complexity, where the rounding function approximates* $\log_2$ *up to* $\mathsf{precision}$ *if there exists one.*

Proof. Algorithm 5.3 is a divide-and-conquer algorithm to find the smallest η such that Algorithm 5.4 finds a valid simple Round_L-rounded chain of max-complexity bounded by 2^η. □

We can see that the complexity of Algorithm 5.4 is of $\mathcal{O}\left(k^2 \cdot |L|\right)$ iterations as vertices have k possible values for the secret length and $|L|$ possible values for the logarithmic number of equations. So, it is linear in the size of the graph. Furthermore, each type of edge to a fixed vertex has $\mathcal{O}(k)$ possible origins. The memory complexity is $\mathcal{O}(k \cdot |L|)$, mainly to store the $\Delta_{k,\eta}$ and $\mathsf{Best}_{k,\eta}$ tables. We also use Algorithm 5.1 which has a complexity $\mathcal{O}(k^3)$ but we run it only once during precomputation. Algorithm 5.3 sets $|L| \sim \frac{k}{\mathsf{precision}}$. So, the complexity of Algorithm 5.3 is $\mathcal{O}\left(k^3 + \frac{k^3}{\mathsf{precision}} \times \log \frac{k}{\mathsf{precision}}\right)$.

5.6 Chains with a Guessing Step

In order to further improve our valid chain, we introduce a new reduction step to our algorithm. As it is done in previous works [GJL14, BL12], we guess part of the bits of the secret. More precisely, we assume that b bits of the secret have a Hamming weight smaller or equal to w. The influence on the whole algorithm is more complicated: it requires to iterate the WHT step $\sum_{i=0}^{w}\binom{w}{i}$ times. The overall complexity must further be divided by $\sum_{i=0}^{w}\binom{w}{i}\left(\frac{1-\delta_s}{2}\right)^{i}\left(\frac{1+\delta_s}{2}\right)^{w-i}$. Note that this generalized *guess-secret* step was used in Guo et al. [GJL14].

We formalize this step as following:

- *guess-secret*(b, w) guesses that b bits of the secret have a Hamming weight smaller or equal to w. The b positions are chosen randomly. The number of queries remains the same, the noise is the same and the size of the secret is reduced by b bits. Thus, for this step we have

guess-secret(b, w) : $k' = k - b$; $n' = n$; $\delta' = \delta$; $\delta'_s = \delta$
Complexity: $\mathcal{O}(nb)$ (included in *sparse-secret*) and
the Walsh transform has to be iterated $\sum_{i=0}^{w}\binom{w}{i}$ times and
the complexity of the whole algorithm is divided by
$\sum_{i=0}^{w}\binom{w}{i}\left(\frac{1-\delta_s}{2}\right)^{i}\left(\frac{1+\delta_s}{2}\right)^{w-i}$

This step may be useful for a sparse secret, i.e. τ is small, as then we reduce the size of the secret with a very small cost. In order to accommodate this new step, we would have to add a transition from state 3 to state 3 in the automaton that accepts the simple chains (See Figure 5.1).

To find the optimal chain using *guess-secret*(b, w), we have to make a loop over all possible b and all possible w. We run the full search $\mathcal{O}(k^2)$ times. The total complexity is thus $\mathcal{O}\left(\frac{k^5}{\mathsf{precision}} \times \log \frac{k}{\mathsf{precision}}\right)$.

5.7 Results

We illustrate in this section the results obtained by running Algorithm 5.4 for different LPN instances. They vary from taking $k = 32$ to $k = 768$, with the noise levels: $0.05, 0.1, 0.125, 0.2$ and 0.25. In Table 5.2, we display the logarithmic time complexity we found for solving LPN without using *guess-secret*.[5]

[5]Complete results are provided in Appendix A.5 and A.6.

τ	k						
	32	48	64	100	256	512	768
0.05	$13.9^{11.3}_{0.1}$	$14.5^{13.0}_{0.1c}$	$16.0^{14.4}_{0.1c}$	$20.5^{18.5}_{0.1c}$	$36.7^{34.4}_{0.1c}$	$57.8^{55.1}_{0.1c}$	$76.6^{74.0}_{0.1c}$
0.1	$15.0^{12.7}_{0.1}$	$18.6^{16.4}_{0.1}$	$21.6^{19.4}_{0.1c}$	$27.6^{25.4}_{0.1c}$	$46.7^{44.2}_{0.1c}$	$73.7^{70.9}_{0.1c}$	$99.0^{96.0}_{0.1c}$
0.125	$15.7^{13.5}_{0.1}$	$19.3^{17.0}_{0.1}$	$22.9^{20.5}_{0.1}$	$28.9^{26.3}_{0.1}$	$49.9^{47.3}_{0.1c}$	$78.8^{76.2}_{0.1c}$	$105.9^{103.0}_{0.1c}$
0.2	$17.0^{14.8}_{0.1}$	$21.2^{19.2}_{0.1}$	$24.4^{22.0}_{0.1}$	$32.1^{29.7}_{0.1}$	$56.3^{53.8}_{0.1c}$	$89.0^{86.4}_{0.1c}$	$121.0^{118.2}_{0.1c}$
0.25	$18.4^{16.3}_{0.1}$	$22.3^{20.4}_{0.1}$	$26.9^{24.6}_{0.1}$	$32.9^{30.7}_{0.1}$	$59.5^{56.9}_{0.1}$	$94.7^{92.0}_{0.1c}$	$127.3^{124.6}_{0.1c}$

entry of form $a^{b}_{c\cdots}$: $a = \log_2$ complexity, $b = \log_2$ max-complexity,
$c =$ precision; subscript c means that a *code-reduce* is used

Table 5.2: Logarithmic time complexity on solving LPN without *guess-secret*

τ	k						
	32	48	64	100	256	512	768
0.05	$11.8^{10.9}_{0.1cg13o}$	$13.0^{12.5}_{0.1cg23o}$	$14.4^{13.7}_{0.1cg38o}$	$17.2^{16.2}_{0.1cg75o}$	$30.1^{28.0}_{0.1cg178o}$	$49.6^{47.3}_{1cg417o}$	$68.1^{66.0}_{1cg682o}$
0.1	$12.4^{11.6}_{0.1cg23o}$	$15.2^{14.2}_{0.1cg37o}$	$17.7^{16.8}_{0.1cg52o}$	$24.0^{22.1}_{0.1cg77o}$	$46.0^{43.5}_{0.1cg100o}$	$73.7^{71.1}_{1cg2}$	$99.2^{96.3}_{1cg5}$
0.125	$13.3^{12.4}_{0.1cg26o}$	$16.5^{15.5}_{0.1cg39o}$	$20.6^{18.6}_{0.1cg36o}$	$27.1^{24.8}_{0.1cg47o}$	$49.9^{47.3}_{0.1c}$	$79.0^{76.2}_{1cg1}$	$106.2^{103.4}_{1cg4}$
0.2	$17.0^{14.8}_{0.1o}$	$21.2^{19.2}_{0.1o}$	$24.4^{22.0}_{0.1}$	$32.1^{29.7}_{0.1}$	$56.3^{53.8}_{0.1cg1}$	$89.3^{86.8}_{1cg3}$	$121.1^{118.6}_{1}$
0.25	$18.4^{16.3}_{0.1}$	$22.3^{20.4}_{0.1}$	$26.9^{24.6}_{0.1}$	$32.9^{30.7}_{0.1}$	$59.5^{56.9}_{0.1}$	$94.8^{92.4}_{1cg2}$	$127.6^{125.0}_{1cg3}$

entry of form $a^{b}_{c\cdots}$: $a = \log_2$ complexity, $b = \log_2$ max-complexity, $c =$ precision
subscript c means that a *code-reduce* is used
subscript o means that a only 1 bit of the secret is found by WHT
subscript gb means that a *guess-secret*$(b, \cdot)$ is used

Table 5.3: Logarithmic time complexity on solving LPN with *guess-secret*

Sequence of chains. If we analyse in more detail one of the chains that we obtained, e.g. the chain for $\mathsf{LPN}_{512,0.125}$, we can see that it first uses a *sparse-secret*. Afterwards, the secret is reduced by applying five times the *xor-reduce* and one *code-reduce* at the end of the chain. With a total complexity of $2^{79.46}$ and $\theta < 33\%$ it recovers 64 bits of the secret. [6]

[6]The detailed results for the full recovery of the secret of this LPN instance are given in Appendix A.7.

$$(512, 63.3) \xrightarrow{\textit{sparse-secret}} (512, 63.3) \xrightarrow{\textit{xor-reduce}(59)} (453, 66.6) \xrightarrow{\textit{xor-reduce}(65)}$$
$$(388, 67.2) \xrightarrow{\textit{xor-reduce}(66)} (322, 67.4) \xrightarrow{\textit{xor-reduce}(66)} (256, 67.8) \xrightarrow{\textit{xor-reduce}(67)}$$
$$(189, 67.6) \xrightarrow{\textit{code-reduce}} (64, 67.6) \xrightarrow{\mathsf{WHT}}$$

The code used is a $[189, 64]$ concatenation made of ten random codes: one instance of a $[18, 6]$ code, five instances of a $[19, 6]$ code, and four instances of a $[19, 7]$ code. By manually tuning the number of equations without rounding, we can obtain with $n = 2^{63.299}$ a complexity of $2^{78.85}$. This is the value from Table 5.4.

On the *guess-secret* reduction. Our results show that the *guess-secret* step does not bring any significant improvement. If we compare Table 5.2 with Table 5.3, we can see that in few cases the guess step improves the total complexity. For $k \geq 512$, some results are not better than Table 5.2. This is most likely due to the lower precision used in Table 5.3.

We can see several cases where, at the end of a chain with *guess-secret*, only one bit of the secret is recovered by WHT. If only 1 bit of the secret is recovered by non-bruteforce methods, the next chain for $\mathsf{LPN}_{k-1,\tau}$ will have to be run several times, given the *guess-secret* step used in the chain for $\mathsf{LPN}_{k,\tau}$. Thus, it might happen that the first chain does not dominate the total complexity. So, our strategy to use sequences of chains has to be revised, but most likely, the final result will not be better than sequences of chains without *guess-secret*. So, we should rather avoid these chains ending with 1 bit recovery.

There is no case where a *guess-secret* without a chain ending with 1 bit brings any improvement.

On using random codes. With our results, we expand the pool of codes that can be used in the reduction *code-reduce*: besides perfect codes, we add quasi-perfect codes and some small random codes that we generated. One could wonder if these results could be improved further by using a big code, so that we do not split the reduction among several small codes. The challenge in big codes is to find a random code that introduces a large bias by having a small covering radius while still decreasing the size of the secret considerably (i.e. a code $[k, k-1, D]$ might not be interesting for our algorithm). By selecting codes used in steganography, we can improve further our results: for $\mathsf{LPN}_{512,0.125}$ from a complexity of $2^{78.85}$ we get a complexity of $2^{78.11}$ [7]. The other results have a similar improvement.

Comparing the results. For practical values we compare our results with the previous work [GJL14, LF06, ZJW16, BTV16].

[7]Personal communication with Simona Samardjiska.

(k, τ)	ASIACRYPT'14 [GJL14]	EUROCRYPT'16 [ZJW16]	our results
$(512, 0.125)$	$2^{86.96}(2^{79.9})$ (proceedings) $2^{81.90}(2^{79.7})$ (presentation) [10]	$2^{80.09}(2^{74.73})$	$2^{78.85}$
$(532, 0.125)$	$2^{88.62}(2^{81.82})$	$2^{82.17}(2^{76.90})$	$2^{81.02}$
$(592, 0.125)$	$2^{97.71}(2^{88.07})$	$2^{89.32}(2^{83.84})$	$2^{87.57}$

Table 5.4: Time complexity to solve LPN (in bit operations). These complexities are based on the formulas from Chapter 5.2 with the most favourable covering codes we constructed from our pool, with adjusted data complexity to reach a failure probability bounded by 33%. Originally claimed complexities by [GJL14] and [ZJW16] are under parentheses.

From the work of ASIACRYPT'14 [GJL14] and EUROCRYPT'16 [ZJW16], we have $\mathsf{LPN}_{512,0.125}$ which can be solved in time complexity of $2^{79.9}$ (with more precise complexity estimates). The comparison is shown in Table 5.4.[8] [9]

Recall that the previous work on LPN analysed *code-reduce* only with perfect codes. In Table 5.4, our computed complexities are based on the real codes that we built with our bigger pool to have a fair comparison.

We do better, provide concrete codes and we even remove the *guess-secret* step using a code in an optimized way. Thus, the results of Algorithm 5.4 improve all the existing results on solving LPN.

We put in Appendices A.5- A.8 details of our results: the complete list of the chains we obtain (for Table 5.2 and Table 5.3), an example of complete solving algorithm, and an analysis of the results from [GJL14] and [ZJW16] to obtain Table 5.4.

[8]As for [ZJW16], we only reported the results based on LF2 which are better than with LF1, as the LF(4) operation is incorrect. Details about this were presented in and [BV16a] and Section 3.7.

[9]Detailed results are provided in Appendix A.8.

[10]`http://des.cse.nsysu.edu.tw/asiacrypt2014/doc/1-1_SolvingLPNUsingCoveringCodes.pdf`

Chapter 6

How to Sequentialize Independent Parallel Attacks

In this chapter, we present a new framework that allows us to improve the complexity of solving LPN when given only a polynomial number of queries. The same framework can be adapted to several other problems, e.g. password guessing. The personal contribution in this chapter is a joint work with Serge Vaudenay that was published in [BV15].

Structure of the Chapter. Section 6.1 introduces the scenario we are working in and connects it with existing results. Section 6.3 formalizes the problem we solve, i.e. what is the optimal strategy for an adversary that wants to succeed in guessing a key from a sequence of independent keys, and presents a few useful results. In Section 6.4, we characterize the optimal strategies and show that they can be given a special regular structure. We then apply our results in Section 6.5 with LPN and password recovery. Finally, in Section 6.6, we study how the strategies are shaped by the distribution of the keys.

6.1 General Framework and Related Work

We have a framework where we assume that there are an infinite number of independent keys $K_1, K_2, \ldots$ and that we want to find at least one of these keys by trials with minimal complexity. Each key search can be stopped and resumed. The problem is to find the optimal strategy to run several partial key searches in a sequence. One restriction to our framework is that we assume a scenario where one can mount several independent attacks on a single CPU. I.e. one cannot run key searches in parallel.

In this optimization problem, we assume that the distributions D_i for each K_i are known. We denote $D = (D_1, D_2, \ldots)$. We consider the problem of guessing a key K_i, drawn following distribution D_i, which is not necessarily uniform. We assume that we try all key values exhaustively from the first to the last following a fixed ordering (we could assume that this ordering follows a decreasing likelihood but this is not necessary). If we stop the key search on K_i after m trials, the sequence of trials is denoted by $ii \cdots i = i^m$. It has a worst-case complexity m and a probability of success which we denote by $\Pr_D(i^m)$.

Instead of running parallel key searches in sequence, we could consider any other attack which decomposes into *steps* of the same complexity and in which each step has a specific probability to be the succeeding one. We assume that the ith attack has a probability $\Pr_D(i^m)$ to succeed within m steps and that each step has complexity 1. The fundamental problem is to wonder how to run steps of these attacks in a sequence so that we minimize the complexity until one attack succeeds. For instance, we could run attack 1 for up to m steps and decide to give up and try again with attack 2 if it fails for attack 1, and so on. We denote by $s = 1^m 2^m 3^m \cdots$ this strategy. Unsurprisingly, when the D_i's are the same, the average complexity of s is the ratio $\frac{C_D(1^m)}{\Pr_D(1^m)}$ where $C_D(1^m)$ is the expected complexity of the strategy 1^m which only runs attack 1 for m steps[1] and $\Pr_D(1^m)$ is its probability of success.

Traditionally, when we want to compare single-target attacks with different complexity C and probability of success p, we use as a rule of the thumb to compare the ratio $\frac{C}{p}$. Quite often, we have a continuum of attacks $C(m)$ with a number of steps limited to a variable m and we tune m so that $p(m)$ is a constant such as $\frac{1}{2}$. Indeed, the curve of $m \mapsto \frac{C(m)}{p(m)}$ is often decreasing (so has an L shape) or decreasing then increasing (with a U shape) and it is optimal to target $p(m) = \frac{1}{2}$. But sometimes, the curve can be increasing with a Γ shape. In this case, it is better to run an attack with very low probability of success and to try again until this succeeds. In some papers, e.g. [HVLN15], we consider $\min \frac{C(m)}{p(m)}$ as a complexity metric to compare attacks. Our framework justifies this choice.

LPN falls in the aforementioned scenario of guessing a k-bit biased noise vector by a simple transformation. Work on breaking cryptosystems with biased keys was also done in [MS91].

The guessing game that we describe in our framework also matches well the password guessing scenario where an attacker tries to gain access to a system by hacking an account of an employee. There exists an extensive work on the cryptanalytic time-memory tradeoffs for password guessing [ABC15, Hel80, Oec03, AC13, NS05, AJO05], but the game we analyse here requires no pre-computation done by the attacker.

6.2 Our Contribution

We develop a formalism to compare strategies and derive some useful lemmas. *We show that when we can run an infinite number of independent attacks of the same distribution, an optimal strategy is of the form* $1^m 2^m 3^m \cdots$ *and it has complexity*

$$\min_m \frac{C_D(1^m)}{\Pr_D(1^m)}$$

[1]$C_D(1^m)$ can be lower than m since there is a probability to succeed before reaching the mth step.

for some "magic" value m. This justifies the rule of thumb to compare attacks with different probabilities of success.

When the probability that an attack succeeds at each new step decreases (e.g., because we try possible key values in decreasing order of likelihood), there are two remarkable extreme cases: $m = n$ (where n is the maximal number of steps) corresponds to the normal single-target exhaustive search with a complexity equal to the *guesswork entropy* [Mas94] of the distribution; $m = 1$ corresponds to trying attacks for a single step until it works, with complexity 2^{-H_∞}, where H_∞ is the *min-entropy* of the distribution.

When looking at the "magic" value m in terms of the distribution D, we observe that in many cases there is a phase transition: when D is very close to uniform, we have $m = n$. As soon as it becomes slightly biased, we have $m = 1$. There is no graceful decrease from $m = n$ to $m = 1$.

We also treat the case where we have a finite number $|D|$ of independent attacks to run. We show that there is an optimal "magic" sequence $m_1, m_2, \ldots$ such that an optimal strategy has form

$$1^{m_1}2^{m_1}\cdots|D|^{m_1}1^{m_2}2^{m_2}\cdots|D|^{m_2}\cdots$$

The best strategy is first to run all attacks for m_1 steps in a sequence then to continue to run them for m_2 steps in a sequence, and so on.

Although our results look pretty natural, we show that there are distributions making the analysis counter-intuitive. Proving these results is actually non trivial.

We apply this formalism to LPN *by guessing the noise vector then performing a Gaussian elimination to extract the secret.* The optimal m decreases as the probability τ to have an error in a parity bit decreases from $\frac{1}{2}$. For $\tau = \frac{1}{2}$, the optimal m corresponds to a normal exhaustive search. For $\tau < \frac{1}{2} - \frac{\ln 2}{2k}$, where k is the length of the secret, the optimal m is 1: this corresponds to guessing that we have no noise at all. So, there is a phase transition.

Furthermore, for LPN with $\tau = k^{-\frac{1}{2}}$, which is what is used in several cryptographic constructions, the obtained complexity is $\mathsf{poly} \cdot e^{\sqrt{k}}$ which is much better than the usual $\mathsf{poly} \cdot 2^{\frac{k}{\log_2 k}}$ that we obtain for variants of the BKW algorithm [BTV16]. More generally, we obtain a complexity of $\mathsf{poly} \cdot e^{-k\ln(1-\tau)}$. It is not better than the BKW variants for constant τ but becomes interesting when $\tau < \frac{\ln 2}{\log_2 k}$.

When the number of samples is limited in the LPN problem with $\tau = k^{-\frac{1}{2}}$, we can still solve it with complexity $e^{\mathcal{O}\left(\sqrt{k}(\ln k)^2\right)}$ which is better than $e^{\mathcal{O}\left(\frac{k}{\ln\ln k}\right)}$ with the BKW variants [Lyu05].

For password search, we tried several empirical distributions of passwords and again obtained that the optimal m is $m = 1$. So, the complexity is 2^{-H_∞}.

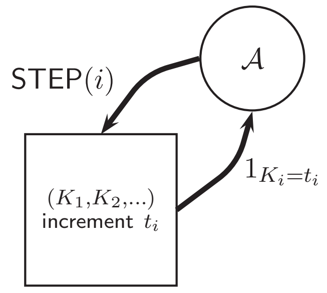

Figure 6.1: The STEP game

6.3 The STEP Game

In this section, we formalize our framework by addressing the fundamental question of what is the best strategy to succeed in at least one attack when we can step several independent attacks. Let $D = (D_1, D_2, \ldots)$ be a tuple of independent distributions. If it is finite, $|D|$ denotes the number of distributions.

We formalize our framework as a game where we have a ppt adversary $\mathcal{A}$ and an oracle that has a sequence of keys $(K_1, K_2, \ldots)$ where $K_i \leftarrow D_i$. At the beginning, the oracle assigns the keys according to their distribution. These distributions are known to the adversary $\mathcal{A}$. The adversary will test each key K_i by exhaustive search following a given ordering of possible values. We can assume that values are sorted by decreasing order of likelihood to obtain a minimal complexity but this is not necessary in our analysis. We only assume a fixed order. So, our framework generalizes to other types of attacks in which we cannot choose the order of the steps. Each test on K_i corresponds to a step in the exhaustive search for K_i. In general, we write "i" in a sequence to denote that we run one new step of the ith attack. The sequence of "i"s defines a strategy s. It can be finite or not. The sequence of steps we follow is thus a sequence of indices. For instance, i^m means "run the K_i search for m steps".

The oracle is an algorithm that has a special command: STEP(i). When queried with the command STEP(i), the oracle runs one more step of the ith attack (so, it increments a counter t_i and tests if $K_i = t_i$, assuming that possible key values are numbered from 1). If this happens, then the adversary wins. The adversary wins as soon as one attack succeeds (i.e., he guesses one of the keys from the sequence $K_1, K_2, \ldots$).

Definition 6.1 (Distributions). *A distribution D_i over a set of size n is a sequence of probabilities $D_i = (p_1, \ldots, p_n)$ of sum 1 such that $p_j \geq 0$ for $j = 1, \ldots, n$. We assume without loss of generality that $p_n \neq 0$ (Otherwise, we decrease n). We can equivalently specify the distribution D_i in an* incremental *way by a sequence $D_i = [p'_1, \ldots, p'_n]$ (denoted with square brackets) such that*

$$p'_j = \frac{p_j}{p_j + \cdots + p_n} \qquad p_j = p'_j(1 - p'_1) \cdots (1 - p'_{j-1})$$

for $j = 1, \ldots, n$.

We have $\Pr_D(i^j) = p_1 + \cdots + p_j = 1 - (1 - p'_1) \cdots (1 - p'_j)$, the probability of the j first values under D_i.

When considering the key search, it may be useful to assume that distributions are sorted by decreasing likelihood. We note that the equivalent condition to $p_j \geq p_{j+1}$ with the incremental description is $\frac{1}{p'_j} + j \leq \frac{1}{p'_{j+1}} + j + 1$, for $j = 1, \ldots, n-1$.

Definition 6.2 (Strategies). *Let D be a sequence of distributions $D = (D_1, \ldots, D_{|D|})$ (where $|D|$ can be infinite or not). A strategy for D is a sequence s of indices between 1 and $|D|$. It corresponds to Algorithm 6.1. We let $\Pr_D(s)$ be the probability that the*

Algorithm 6.1 Strategy s in the STEP game

```
1: initialize attacks $1, \ldots, |D|$
2: **for** $j = 1$ to $|s|$ **do**
3:     STEP($s_j$): run one more step of the attack $s_j$ and stop if succeeded
4: **end for**
5: stop (the algorithm fails)
```

strategy succeeds and $C_D(s)$ be the expected number of STEP *when running the algorithm until it stops. We say that the strategy is full if $\Pr_D(s) = 1$ and that it is partial otherwise.*

For example for $s = 11223344\cdots$, Algorithm 6.1 tests the first two values for each key.

We define the distribution that the keys are not among the already tested ones.

Definition 6.3 (Residual distribution). *Let $D = (D_1, \ldots, D_{|D|})$ be a sequence of distributions and let s be a strictly partial strategy for D (i.e., $\Pr_D(s) < 1$). We denote by "$|\neg s$" the residual distribution in the case where the strategy s does not succeed, i.e., the event $\neg s$ occurs.*

We let $\#\mathsf{occ}_s(i)$ denote the number of occurrences of i in s. We have

$$D|\neg s = \left(D_1|\neg 1^{\#\mathsf{occ}_s(1)}, \ldots, D_{|D|}|\neg |D|^{\#\mathsf{occ}_s(|D|)}\right)$$

where $D_i|\neg i^{t_i} = [p'_{i,t_i+1}, \ldots, p'_{i,n_i}]$ if $D_i = [p'_{i,1}, \ldots, p'_{i,n_i}]$. Hence, defining distributions in the incremental way makes the residual distribution being just a shift of the original one.

We write $\Pr_D(s'|\neg s) = \Pr_{D|\neg s}(s')$ and $C_D(s'|\neg s) = C_{D|\neg s}(s')$.

Next, we prove a list of useful lemmas in order to compute complexities, compare strategies, etc.

Lemma 6.4 (Success probability). *Let s be a strategy for D. The success probability is given by*

$$\Pr_D(s) = 1 - \prod_{i=1}^{|D|} \Pr_{D_i}(\neg i^{\#\mathsf{occ}_s(i)})$$

Proof. The failure corresponds to the case where for all i, K_i is not in $\{1, \ldots, \#\mathsf{occ}_s(i)\}$. The independence of the K_i implies the result. $\square$

Lemma 6.5 (Complexity of concatenated strategies)**.** *Let ss' be a strategy for D obtained by concatenating the sequences s and s'. If $\Pr_D(s) = 1$, we have $\Pr_D(ss') = \Pr_D(s)$ and $C_D(ss') = C_D(s)$. Otherwise, we have*

$$\begin{aligned}\Pr_D(ss') &= \Pr_D(s) + \left(1 - \Pr_D(s)\right) \Pr_D(s'|\neg s)\\ C_D(ss') &= C_D(s) + \left(1 - \Pr_D(s)\right) C_D(s'|\neg s)\ .\end{aligned}$$

Proof. The first equation is trivial from the definition of residual distributions and conditional probabilities.

The prefix strategy s succeeds with probability $\Pr_D(s)$. Let c be the complexity of s conditioned to the event that s succeeds. Clearly, the complexity of ss' conditioned to this event is equal to c. The complexity of ss' conditioned to the opposite event is equal to $|s| + C_D(s'|\neg s)$. So, $C_D(ss') = \Pr_D(s)c + (1 - \Pr_D(s))(|s| + C_D(s'|\neg s))$. The complexity of s conditioned to the event that s fails is equal to $|s|$. So, $C_D(s) = \Pr_D(s)c + (1 - \Pr_D(s))|s|$. From these two equations, we obtain the result. $\square$

Lemma 6.6 (Complexity with incremental distributions)**.** *Let $D_i = [p'_{i,1}, \ldots, p'_{i,n_i}]$ and let s be a strategy for $D = (D_1, D_2, \ldots)$. We have*

$$\begin{aligned}\Pr_D(s) &= 1 - \prod_{t'=1}^{|s|} (1 - p'_{s_{t'},\#\mathsf{occ}_{s_1\cdots s_{t'}}(s_{t'})})\\ C_D(s) &= \sum_{t=1}^{|s|} \prod_{t'=1}^{t-1} (1 - p'_{s_{t'},\#\mathsf{occ}_{s_1\cdots s_{t'}}(s_{t'})})\ .\end{aligned}$$

Proof. By induction, the probability that the strategy fails on the first $t-1$ steps is $q_t = \prod_{t'=1}^{t-1}(1 - p'_{s_{t'},\#\mathsf{occ}_{s_1\cdots s_{t'}}(s_{t'})})$. We can express $C_D(s) = \sum_{t=1}^{|s|} q_t$. So, we can deduce $\Pr_D(s)$ and $C_D(s)$. $\square$

Example 6.3.1. *For $D_1 = (p_1, \ldots, p_n) = [p'_1, \ldots, p'_n]$ and $m \le n$, due to Lemma 6.6 we have*

$$\Pr_D(1^m) = p_1 + \cdots + p_m = 1 - (1 - p'_1) \cdots (1 - p'_m)$$

and

$$\begin{aligned}C_D(1^m) &= \sum_{t=1}^{m} \prod_{j=1}^{t-1} (1 - p'_j)\\ &= \sum_{t=1}^{m} (p_t + \cdots + p_n) = p_1 + 2p_2 + \cdots + mp_m + mp_{m+1} + \cdots + mp_n\ .\end{aligned}$$

The second equality uses the relations from Definition 6.1.

For the next results, we analyse strategies where the same number of guesses are done for several keys (e.g. $1^m 2^m \cdots$). We want to concatenate an isomorphic copy w of a strategy v to another strategy u. For this, we make sure that w and u have no index in common.

Definition 6.7 (Disjoint copy of a strategy)**.** *Two strategies v and w are isomorphic if there exists an injective mapping φ such that $w_t = \varphi(v_t)$ for all t and $D_{\varphi(i)} = D_i$ for all i. So, $C_D(v) = C_D(w)$. Let u and v be two strategies for D. Whenever possible, we define a new strategy $w = \mathsf{new}_u(v)$ such that v and w are isomorphic and w has no index in common with u.*

We can define it by recursion: if $w_1 = \varphi(v_1), \ldots, w_{t-1} = \varphi(v_{t-1})$ are already defined and $\varphi(v_t)$ is not, we set it to the smallest index i (if exists) which does not appear in u nor in $w_1, \ldots, w_{t-1}$ and such that $D_i = D_{v_t}$.

For instance, if $v = 1^m$, all D_i are equal, and i is the minimal index which does not appear in u, we have $\mathsf{new}_u(v) = i^m$.

Lemma 6.8 (Complexity of a repetition of disjoint copies)**.** *Let s be a non-empty strategy for D. We define new strategies $s_{+1}, s_{+2}, \ldots$, disjoint copies of s, by recursion as follows: $s_{+r} = \mathsf{new}_{ss_{+1}\cdots s_{+(r-1)}}(s)$. We assume that $s_{+1}, s_{+2}, \ldots, s_{+(r-1)}$ can be constructed. If $\Pr_D(s) = 0$, then*

$$C_D(ss_{+1}s_{+2}\cdots s_{+(r-1)}) = r \cdot C_D(s).$$

Otherwise, we have

$$C_D(ss_{+1}s_{+2}\cdots s_{+(r-1)}) = \frac{1-(1-\Pr_D(s))^r}{\Pr_D(s)} C_D(s)\ .$$

For r going to ∞, we respectively obtain $C_D(ss_{+1}s_{+2}\cdots) = +\infty$ and

$$C_D(ss_{+1}s_{+2}\cdots) = \frac{C_D(s)}{\Pr_D(s)}\ .$$

For instance, for $s = 1^m$ and D_i all equal, the disjoint isomorphic copies of s are $s_{+r} = (1+r)^m$. I.e., we run m steps the $(1+r)$th attack. So, $ss_{+1}s_{+2}\cdots s_{+(r-1)} = 1^m 2^m \cdots r^m$.

Proof. We prove it by induction on r. This is trivial for $r = 1$. Let $\bar{s}_r = ss_{+1}s_{+2}\cdots s_{+r}$. If it is true for $r-2$, then

$$\begin{aligned} C_D(\bar{s}_{r-1}) &= C_D(\bar{s}_{r-2}) + (1 - \Pr_D(\bar{s}_{r-2}))C_D(s_{+(r-1)}|\neg\bar{s}_{r-2}) \\ &= \begin{cases} \frac{1-(1-\Pr_D(s))^{r-1}}{\Pr_D(s)} C_D(s) + (1-\Pr_D(\bar{s}_{r-2}))C_D(s_{+(r-1)}|\neg\bar{s}_{r-2}) & \text{if } \Pr_D(s) > 0 \\ (r-1)\cdot C_D(s) + (1-\Pr_D(\bar{s}_{r-2}))C_D(s_{+(r-1)}|\neg\bar{s}_{r-2}) & \text{if } \Pr_D(s) = 0 \end{cases} \end{aligned}$$

Clearly, we have $1 - \Pr_D(\bar{s}_{r-2}) = (1 - \Pr_D(s))^{r-1}$ and $C_D(s_{+(r-1)}|\neg\bar{s}_{r-2}) = C_D(s)$. So, we obtain the result. □

Example 6.3.2. *For all D_i equal, if we let $s = 1^m$, we can compute*

$$\begin{aligned} C_D(1^m 2^m \cdots r^m) &= \frac{1 - (1 - \Pr_D(1^m))^r}{\Pr_D(1^m)} C_D(1^m) \\ &= \frac{1 - (p_{m+1} + \cdots + p_n)^r}{p_1 + \cdots + p_m}(p_1 + 2p_2 + \cdots + mp_m + mp_{m+1} + \cdots + mp_n) \end{aligned}$$

We now consider $r = \infty$. For an infinite number of i.i.d distributions we have

$$\begin{aligned} C_D(1^m 2^m \cdots) &= \frac{C_D(1^m)}{\Pr_D(1^m)} \\ &= \frac{p_1 + 2p_2 + \cdots + mp_m + mp_{m+1} + \cdots + mp_n}{p_1 + \cdots + p_m} \\ &= \frac{\sum_{i=1}^m ip_i + m(1 - p_1 + \cdots + p_m)}{p_1 + \cdots + p_m} \\ &= G_m + m\left(\frac{1}{\Pr_{D_i}(1^m)} - 1\right) \end{aligned}$$

where $G_m = C_{D_1|1^m}(1^m)$ and $D_1|1^m = (\frac{p_1}{\Pr_{D_1}(1^m)}, \ldots, \frac{p_m}{\Pr_{D_1}(1^m)})$. If D_1 is ordered, G_m corresponds to the guesswork entropy of the key with distribution $D_1|1^m$.

We can see two extreme cases for $s = 1^m 2^m \cdots$. On one end, we have a strategy of exhaustively searching the key until it is found, i.e. take $m = n$. On the other extreme, we have a strategy where the adversary tests just one key before switching to another key, i.e. $m = 1$. For the sequences $s = 12\cdots$ and $s = 1^n 2^n \cdots$, i.e. $m = 1$ and $m = n$, when D_1 is ordered by decreasing likelihood, we obtain the following expected complexity:

$$\begin{aligned} m = 1 \Rightarrow \qquad & C_D(12\cdots) = \frac{1}{p_1} = 2^{-H_\infty(D_1)} \\ m = n \Rightarrow \qquad & C_D(1^n 2^n \cdots) = C_D(1^n) = G_n, \end{aligned}$$

where $H_\infty(D_1)$ and G_n denote the min-entropy and the guesswork entropy of the distribution D_1, respectively.

We now define a way to compare partial strategies.

Definition 6.9 (Strategy comparison)**.** *We define*

$$\mathsf{minC}_D(s) = \inf_{s';\Pr_D(ss')=1} C_D(ss')$$

the infimum of $C_D(ss')$, i.e. the greatest of its lower bounds. We write $s \leq_D s'$ if and only if $\mathsf{minC}_D(s) \leq \mathsf{minC}_D(s')$. A strategy s is optimal if $\mathsf{minC}_D(s) = \mathsf{minC}_D(\emptyset)$, where $\emptyset$ is the empty strategy (i.e. the strategy running no step at all).

So, s is better than s' if we can reach lower complexities by starting with s instead of s'. The partial strategy s is optimal if we can still reach the optimal complexity when we start by s.

Lemma 6.10 (Best prefixes are best strategies)**.** *If u and v are permutations of each other, we have $u \leq_D v$ if and only if $C_D(u) \leq C_D(v)$.*

Proof. Note that $\Pr_D(u) = 1$ is equivalent to $\Pr_D(v) = 1$. If $\Pr_D(u) = 1$, it holds that $\mathsf{minC}_D(u) = C_D(u)$ and $\mathsf{minC}_D(v) = C_D(v)$. So, the result is trivial in this case. Let us now assume that $\Pr_D(u) < 1$ and $\Pr_D(v) < 1$. For any s', by using Lemma 6.5 we have

$$C_D(us') = C_D(u) + \left(1 - \Pr_D(u)\right) C_D(s'|\neg u)$$

So,

$$\inf_{s';\Pr_D(us')=1} C_D(us') = C_D(u) + \left(1 - \Pr_D(u)\right) \inf_{s';\Pr_D(us')=1} C_D(s'|\neg u)$$

The same holds for v. Since u and v are permutations of each other, we have $D|\neg u = D|\neg v$. So, $\Pr_D(us') = \Pr_D(vs')$ and $C_D(s'|\neg u) = C_D(s'|\neg v)$. Hence, $\inf C_D(s'|\neg u) = \inf C_D(s'|\neg v)$. Furthermore, we have $\Pr_D(u) = \Pr_D(v)$. So, $\mathsf{minC}_D(u) \leq \mathsf{minC}_D(v)$ is equivalent to $C_D(u) \leq C_D(v)$. □

6.4 Optimal Strategy

The question we address in this chapter is: what is the optimal strategy for the adversary so that he obtains the best complexity in our STEP formalism? That is, we try to find the optimal sequence s for Algorithm 6.1. At a first glance, we may think that a *greedy* strategy, where we always make a step which is the most likely to succeed, is an optimal strategy. We show below that this is wrong. Sometimes, it is better to run a series of unlikely steps in one given attack because we can then run a much more likely one of the same attack after these steps are completed. However, criteria to find this strategy are not trivial at all.

The greedy algorithm is based on looking at the i for which the next applicable p'_j in D_i is the largest. With our formalism, this defines as follows.

Definition 6.11 (Greedy strategy)**.** *Let s be a strategy for D. We say that s is greedy if*

$$\Pr_D(s_t|\neg s_1 \cdots s_{t-1}) = \max_i \Pr_D(i|\neg s_1 \cdots s_{t-1})$$

for $t = 1, \ldots, |s|$.

The following example shows that the greedy strategy is not always optimal.

Example 6.4.1. *We take $|D| = \infty$ and all D_i equal to $D_i = (\frac{2}{3}, \frac{7}{36}, \frac{5}{36}) = [\frac{2}{3}, \frac{7}{12}, 1]$. After testing the first key, we have $D|\neg 1 = (D', D_2, D_3, \ldots)$ with $D' = (\frac{7}{12}, \frac{5}{12}) = [\frac{7}{12}, 1]$. Since $\frac{2}{3} > \frac{7}{12}$, the greedy algorithm would then test a new key and continue testing new keys. I.e., we would have $s = 1234\cdots$ as a greedy strategy. By applying Lemma 6.5, the complexity is solution to $c = 1 + \frac{1}{3}c$, i.e., $c = \frac{3}{2}$. However, the one-key strategy $s = 111$ has complexity*

$$\frac{2}{3} + 2\frac{7}{36} + 3\frac{5}{36} = \frac{53}{36} < \frac{3}{2}$$

so the greedy strategy is not the best one.

Remark: The above counterexample works even when $|D|$ is finite. If we take $D = (D_1, D_2)$ with $D_i = (\frac{2}{3}, \frac{7}{36}, \frac{5}{36}) = [\frac{2}{3}, \frac{7}{12}, 1]$, the greedy approach would test the strategy $s = 1211$ that has a complexity of

$$1 + \frac{1}{3}\left(1 + \frac{1}{3}\left(1 + \frac{5}{12}\cdot 1\right)\right) = \frac{161}{108}.$$

This is greater than $\frac{53}{36}$, which is the complexity of the strategy 111.

Next, we note that we may have no optimal strategy as the following example shows.

Example 6.4.2 (Distribution with no optimal strategy)**.** *Let q_i be an increasing sequence of probabilities which tends towards 1 without reaching it. Let $D_i = [q_i, q_i, \ldots, q_i, 1]$ of support n. We have $C(i^n) = \frac{1}{q_i}(1 - (1 - q_i)^n)$ which tends towards 1 as i grows. So, 1 is the best lower bound of the complexity of full strategies. But there is no full strategy of complexity 1.*

When the number of different distributions is finite, optimal strategies exist.

Lemma 6.12 (Existence of an optimal full strategy)**.** *Let $D = (D_1, D_2, \ldots)$ be a sequence of distributions. We assume that we have in D a finite number of different distributions. There exists a full strategy s such that $C_D(s)$ is minimal.*

Proof. Let $c = \inf C_D(s)$ over all full strategies s. c is well defined. Essentially, we want to prove that c is reached by one strategy, i.e. that the infimum is a minimum. First, if $c = \infty$, all full strategies have infinite complexity, and the result is trivial. So, we now assume that $c < +\infty$ and we prove the result by a diagonal argument.

We now construct $s = s_1 s_2 \cdots$ by recursion. We assume that $s_1 s_2 \cdots s_r$ is constructed such that $\mathsf{minC}(s_1 s_2 \cdots s_r) = c$. We concatenate $s_1, \ldots, s_r$ to i^m where m is such that $\Pr_D[i^{m-1}|\neg s_1 \cdots s_r] = 0$ and $\Pr_D[i^m|\neg s_1 \cdots s_r] > 0$. The values of i to try are the ones such that i appears in $s_1, \ldots, s_r$ (we have a finite number of them), and the ones which do not appear, but we can try only one for each different D_i. We take the choice minimizing $\mathsf{minC}(s_1 s_2 \cdots s_r i^m)$ and set $s_{r+1} = i^m$. So, we construct a strategy s.

If one key K_i is tested until exhaustion, we have $\Pr_D(s) = 1$. If no key is tested until exhaustion, there is an infinite number of keys with same distribution D_i which are tested. If $p = \Pr_D[i^m]$ is the nonzero probability with the smallest m of this distribution, there is an infinite number of tests which succeed with probability p. So, $\Pr_D(s) \geq 1-(1-p)^\infty = 1$. In all cases, as s has a probability to succeed of 1, s is a full strategy.

What remains to be proven is that $C_D(s) = c$. We now denote by s_i the ith step of s.

Let q_t be the probability that s fails on the first $t-1$ steps. We have $C_D(s) = \sum_{t=1}^{|s|} q_t$. Let $\varepsilon > 0$. For each r, by construction, there exists a tail strategy v such that $C_D(s_1 \cdots s_{r-1} v) \leq c + \varepsilon$. Since q_t is also the probability that $s_1 \cdots s_{r-1} v$ fails on the first $t-1$ steps for $t \leq r$, we have $\sum_{t=1}^{r} q_t \leq C_D(s_1 \cdots s_{r-1} v) \leq c + \varepsilon$. This holds for all r. So, we have $C_D(s) \leq c + \varepsilon$. Since this holds for all $\varepsilon > 0$, we have $C_D(s) \leq c$. Consequently, $C_D(s) = c$: s is an optimal and full strategy. □

The following two subsections show what is the structure of an optimal strategy.

6.4.1 Optimal Strategy for an Infinite Number of Distributions

Theorem 6.13. *Let $D = (D_1, D_2, \ldots)$ be a sequence of distributions. We assume that we have in D a finite number of pairwise different distributions but an infinite number of copies of each of them in D. Then, there exists a sequence of indices $i_1 < i_2 < \cdots$ and an integer m such that $D_{i_1} = D_{i_2} = \cdots$ and $s = i_1^m i_2^m \cdots$ is an optimal strategy of complexity $\frac{C_D(i_1^m)}{\Pr_D(i_1^m)}$.*

This result can be translated in: if we can run independent copies of several algorithms and that the probability that the ith one succeeds after m steps is $\Pr_D(i^m)$ with complexity $C_D(i^m)$, the best strategy to make at least one algorithm succeed is to run iteratively some copies of a single algorithm for m steps, for some magic value m. The complexity is $\frac{C_D(i^m)}{\Pr_D(i^m)}$.

To prove the result, we first state a useful lemma.

Lemma 6.14 (Is it better to do s or s' first?)**.** *If s and s' are non-empty and have no index in common (i.e., if $s_t \neq s'_{t'}$ for all t and t'), then $ss' \leq_D s's$ if and only if $\frac{C_D(s)}{\Pr_D(s)} \leq \frac{C_D(s')}{\Pr_D(s')}$ in $[0, +\infty]$, with the convension that $\frac{c}{p} = +\infty$ for $c > 0$ and $p = 0$.*

Proof. Due to Lemma 6.5, when $\Pr_D(s) < 1$ we have

$$C_D(ss') = C_D(s) + \left(1 - \Pr_D(s)\right) C_D(s'|\neg s) \ .$$

Since s' does not make use of the distributions which are dropped in $D|\neg s$, we have $C_D(s'|\neg s) = C_D(s')$. So,

$$C_D(ss') = C_D(s) + \left(1 - \Pr_D(s)\right) C_D(s') \ .$$

This is also clearly the case when $\Pr_D(s) = 1$. Similarly,

$$C_D(s's) = C_D(s') + \left(1 - \Pr_D(s')\right) C_D(s) \ .$$

So, $C_D(ss') \leq C_D(s's)$ is equivalent to

$$C_D(s) + \left(1 - \Pr_D(s)\right) C_D(s') \leq C_D(s') + \left(1 - \Pr_D(s')\right) C_D(s) .$$

Hence, this inequality is equivalent to $\frac{C_D(s)}{\Pr_D(s)} \leq \frac{C_D(s')}{\Pr_D(s')}$. □

We can now prove Th. 6.13.

Proof of Th. 6.13. Due to Lemma 6.12, we know that optimal full strategies exist. Let s be one of these. We let i be the index of an arbitrary key which is tested in s. We can write $s = u_0 i^{m_1} u_1 i^{m_2} \cdots i^{m_r} u_r$ where i appears in no u_j and $m_j > 0$ for all j, and $u_1, \ldots, u_{r-1}$ are non-empty.

Since s is optimal, by permuting i^{m_j} and either u_{j-1} or u_j, we obtain larger complexities. So, by applying Lemma 6.14, we obtain

$$\frac{C_D(i^{m_1})}{\Pr_D(i^{m_1})} \leq \frac{C_D(u_1|\neg u_0)}{\Pr_D(u_1|\neg u_0)} \leq \frac{C_D(i^{m_2}|\neg i^{m_1})}{\Pr_D(i^{m_1}|\neg i^{m_1})} \leq \cdots \leq C_D(u_r|\neg u_0 \cdots u_{r-1})$$

We now want to replace u_r in s by some isomorphic copy of s which is not overlapping with $u_0 i^{m_1} u_1 i^{m_2} \cdots i^{m_r}$. Due to the optimality of s, we would deduce

$$C_D(u_r|\neg u_0 \cdots u_{r-1}) \leq C_D(s|\neg u_0 \cdots u_{r-1}) = C_D(s)$$

so $\frac{C_D(i^{m_1})}{\Pr_D(i^{m_1})} \leq C_D(s)$ which would imply that the repetition of isomorphic copies of i^{m_1} are at least as good as s, so $\frac{C_D(i^{m_1})}{\Pr_D(i^{m_1})} = C_D(s)$ due to the optimality of s. But to replace u_r in s by the isomorphic copy of s, we need to rewrite the original s containing u_r by some isomorphic copy in which indices are left free to implement another isomorphic copy of s.

For that, we split the sequence $(1, 2, 3, \ldots)$ into two subsequences v and v' which are non-overlapping (i.e. $v_t \neq v'_{t'}$ for all t and t'), complete (i.e. for every integer j, v contains j or v' contains j), and representing each distribution with infinite number of occurrences (i.e. for all j, there exist infinite sequences $t_1 < t_2 < \cdots$ and $t'_1 < t'_2 < \cdots$ such that $D_j = D_{v_{t_\ell}} = D_{v'_{t'_\ell}}$ for all ℓ). For that, we can just construct v and v' iteratively: for each j, if the number of $j' < j$ such that $D_{j'} = D_j$ in v or v' is the same, we put j in v, otherwise (we may have only one more instance in v), we put j in v' (to balance again). For instance, if all D_i are equal, this construction puts all odd j in v and all even j in v'. Hence, we can define $s' = \mathsf{new}_v(s)$ and $s'' = \mathsf{new}_{v'}(s)$. s' will thus only use indices in v' while s'' will only use indices in v. Therefore, s' and s'' will be isomorphic, with no index in common. So, $C_D(s) = C_D(s') = C_D(s'')$.

Following the split of s, the strategy s' can be written $s' = u'_0 i'^{m_1} u'_1 i'^{m_2} \cdots i'^{m_r} u'_r$ with

$$\frac{C_D(i^{m_1})}{\Pr_D(i^{m_1})} = \frac{C_D(i'^{m_1})}{\Pr_D(i'^{m_1})} \leq C_D(u'_r|\neg u'_0 \cdots u'_{r-1}) = C_D(u'_r|\neg u'_0 i'^{m_1} u'_1 i'^{m_2} \cdots i'^{m_r})$$

If we replace u'_r in s' by s'', since s' is optimal, we obtain a larger complexity. So,

$$C_D(u'_0 i'^{m_1} u'_1 i'^{m_2} \cdots i'^{m_r} u'_r) \leq C_D(u'_0 i'^{m_1} u'_1 i'^{m_2} \cdots i'^{m_r} s'')$$

These two strategies have the prefix $u'_0 i'^{m_1} u'_1 i'^{m_2} \cdots i'^{m_r}$ in common. We can write their complexities by splitting this common prefix using Lemma 6.5. By eliminating the common terms, we deduce

$$C_D(u'_r | \neg u'_0 i'^{m_1} u'_1 i'^{m_2} \cdots i'^{m_r}) \leq C_D(s'' | \neg u'_0 i'^{m_1} u'_1 i'^{m_2} \cdots i'^{m_r}) = C_D(s'') = C_D(s)$$

We deduce

$$\frac{C_D(i^{m_1})}{\Pr_D(i^{m_1})} \leq C_D(s)$$

Let $i_1 < i_2 < \cdots$ be a sequence of keys using the distribution D_i. By Lemma 6.8, the strategy $i_1^m i_2^m \cdots$ has complexity $\frac{C_D(i^{m_1})}{\Pr_D(i^{m_1})}$. As s is optimal, we have $\frac{C_D(i^{m_1})}{\Pr_D(i^{m_1})} \geq C_D(s)$. Therefore, $\frac{C_D(i^{m_1})}{\Pr_D(i^{m_1})} = C_D(s)$. □

Here are examples of optimal m for different distributions.

Example 6.4.3 (Uniform distribution)**.** *For the uniform distribution $p_i = \frac{1}{n}$, with $1 \leq i \leq n$. We get $\Pr_D(1^m) = \frac{m}{n}$ and $G_m = \frac{m+1}{2}$. With this, we obtain $C_D(1^m 2^m \cdots) = n - \frac{m-1}{2}$. Thus, the value of m that minimizes the complexity is $m = n$ and $C_D(1^m 2^m \cdots) = \frac{n-1}{2}$. The best strategy is to exhaustively search the key until it is found.*

Example 6.4.4 (Geometric distribution)**.** *For the geometric distribution with parameter p, we have $p_i = (1-p)^{i-1} p$, with $i = 1, 2, \ldots$ or $D_i = [p, p, \ldots]$. Due to Lemma 6.5, we can see that for every infinite strategy s, $C_D(s) = \frac{1}{p}$.*

We give a formula to compute the optimal strategies for distributions obtained by composing several distributions. The formula is useful when we want to regroup equal consecutive p_j's in a distribution D_1 so that D_1 appears as a composition of uniform distributions.

Lemma 6.15. *Let $U_1, \ldots, U_k$ be independent distributions of support $n_1, \ldots, n_k$, respectively. Let $U_i = (p_{i,1}, \ldots, p_{i,n_i})$. Given a distribution $(\alpha_1, \ldots, \alpha_k)$ of support k, we define $D_1 = \alpha_1 U_1 + \alpha_2 U_2 + \ldots + \alpha_k U_k$ by $D_1 = (\alpha_1 p_{1,1}, \ldots, \alpha_1 p_{1,n_1}, \alpha_2 p_{2,1}, \ldots, \alpha_k p_{k,n_k})$.*

Let $m = \sum_{j=1}^{i} n_j$. We have

$$\begin{aligned} \Pr_{D_1}(1^{n_1} 1^{n_2} \cdots 1^{n_i}) &= \alpha_1 + \cdots + \alpha_i \\ C_{D_1}(1^{n_1} 1^{n_2} \cdots 1^{n_i}) &= \sum_{j=1}^{i} \alpha_j C_{U_j}(1^{n_j}) + \sum_{j=1}^{i} n_j \left(1 - \sum_{k=1}^{j} \alpha_k\right) \end{aligned}$$

We note that if all U_i are ordered and if $\alpha_i p_{i,n_i} \geq \alpha_{i+1} p_{i+1,1}$ for all $1 \leq i < k$, then D_1 is ordered as well.

We let $D = (D_1, D_1, \ldots)$. If we assume that the U_i are uniform distributions, we can use the observation following Lemma 6.18 to deduce from Th. 6.13 that the optimal strategy is $1^m 2^m \cdots$ for $m = \sum_{j=1}^{i} n_j$ and i minimizing

$$\mathsf{minC}_D(\emptyset) = \min_i \left(\frac{\sum_{j=1}^{i} \alpha_j C_{U_j}(1^{n_j}) + \sum_{j=1}^{i} n_j \left(1 - \sum_{k=1}^{j} \alpha_k\right)}{\sum_{j=1}^{i} \alpha_j} \right) .$$

Proof. We prove it by induction on i. It is trivial for $i = 0$. We assume the result holds for $i - 1$. By induction, we have

$$\begin{aligned} C_{D_1}(1^{n_1} \cdots 1^{n_i}) &= C_{D_1}(1^{n_1} \cdots 1^{n_{i-1}}) + (1 - \Pr_{D_1}(1^{n_1} \cdots 1^{n_{i-1}}))C_{D_1}(1^{n_i} | \neg(1^{n_1} \cdots 1^{n_{i-1}})) \\ &= \sum_{j=1}^{i-1} \alpha_j C_{U_j}(1^{n_j}) + \sum_{j=1}^{i-1} n_j \left(1 - \sum_{k=1}^{j} \alpha_k\right) + \alpha_i C_{U_i}(1^{n_i}) + n_i \left(1 - \sum_{k=1}^{i} \alpha_k\right) \\ &= \sum_{j=1}^{i} \alpha_j C_{U_j}(1^{n_j}) + \sum_{j=1}^{i} n_j \left(1 - \sum_{k=1}^{j} \alpha_k\right) \end{aligned}$$

The second equality is obtained from the fact that

$$\begin{aligned} C_{D_1}(1^{n_i} | \neg(1^{n_1} \cdots 1^{n_{i-1}})) &= \frac{\alpha_i}{\alpha_i + \cdots + \alpha_k} (p_{i,1} + 2p_{i,2} + \ldots + n_i p_{i,n_i}) + n_i \left(\frac{\alpha_{i+1} + \cdots + \alpha_k}{\alpha_i + \cdots + \alpha_k} \right) \\ &= \frac{\alpha_i}{1 - \Pr_{D_1}(1^{n_1} \cdots 1^{n_{i-1}})} C_{U_i}(1^{n_i}) + n_i \left(\frac{1 - \Pr_{D_1}(1^{n_1} \cdots 1^{n_{i-1}}) - \alpha_i}{1 - \Pr_{D_1}(1^{n_1} \cdots 1^{n_{i-1}})} \right) \end{aligned}$$

□

We note that Th. 6.13 does not extend if some distribution has a finite number of copies as the following example shows.

Example 6.4.5 (Distribution with no optimal strategy of the form $i_1^m i_2^m \cdots$). *Let $D_1 = [1 - \varepsilon, \varepsilon, \varepsilon, \ldots, \varepsilon, 1]$ of support n and $D_2 = D_3 = \cdots = [p, \ldots, p, 1]$ for $\varepsilon < p \le \frac{1}{2}$ and n large enough. Given a full strategy s, the formula in Lemma 6.6 defines a sequence $q_t(s) = p'_{s_t, \#\mathsf{occ}_{s_1 \cdots s_t}(s_t)}$. We can see that for all full strategies s and s', if $|s| \le |s'|$ and $q_t(s) \ge q_t(s')$ for $t = 1, \ldots, |s|$, then $C_D(s) \le C_D(s')$. With this, we can see that $s = 12^n$ is better than all full strategies with length at least $n+1$. There are only two full strategies with smaller length: 1^n and 2^n. We have $C_D(2^n) = \frac{1-(1-p)^n}{p} \approx \frac{1}{p} \ge 2$ as n grows. We have $C_D(12^n) = 1 + \varepsilon \frac{1-(1-p)^n}{p} \approx 1 + \frac{\varepsilon}{p}$ as n grows, so $C_D(12^n) < C_D(2^n)$ for n large enough. We have $C_D(1^n) = 1 + \varepsilon \frac{1-(1-\varepsilon)^{n-1}}{\varepsilon} = 2 - (1-\varepsilon)^{n-1} \approx 2$ so $C_D(12^n) < C_D(1^n)$ for n large enough. For all strategies of length at least $n+1$, $s = 12^n$ collected the largest possible p' values. So, the best strategy is $s = 12^n$. It is better than any strategy of form $i_1^m i_2^m \cdots$.*

6.4.2 Optimal Strategy for a Finite Number of Distributions

When we have a finite number of distributions, we may have no optimal strategy of the form given in Th. 6.13. We may have multiple layers of repetition of i^m as the following result shows.

Theorem 6.16. *Let D_1 be a distribution of finite support n. Let $D = (D_1, D_2, \ldots, D_{|D|})$ be a finite sequence of length $|D|$ in which $D_1 = D_2 = \cdots = D_{|D|}$. There exists a sequence $m_1, \ldots, m_r$ such that the strategy*

$$s = 1^{m_1} 2^{m_1} \cdots |D|^{m_1} 1^{m_2} 2^{m_2} \cdots |D|^{m_2} \cdots 1^{m_r}$$

is optimal.

For the proof of Theorem 6.16, we need the result of the following lemma.

Lemma 6.17. *Let $s = ui^a vj^b w$ be an optimal strategy with n occurrences of each key. We assume that $i \neq j$, $a < b$, u does not end with i, v has no occurrence of either i or j, and w has equal number of occurrences for i and j. Furthermore, we assume that either $a \neq 0$, or v is nonempty and starts with some k such that u does not end with k. Then, $C_D(s) = C_D(uj^{b-a} i^a vj^a w)$.*

Proof. We will show below that there exists $d > 0$ such that $a \leq b - d$ and $C_D(s) = C_D(uj^d i^a vj^{b-d} w)$. Hence, we can rewrite s by replacing u by uj^d and b by $b - d$. Since $d > 0$ and $a \leq b - d$, we can just apply this rewriting rule enough times until b is lowered down to a. Hence, we obtain the result.

To find d, we first write $s = u_0 i^{m_1} u_1 i^{m_2} \cdots i^{m_r} u_r i^a vj^b w$ where i appears in no u_t, the m_t are nonzero, and $u_1, \ldots, u_r$ are non-empty. (Note that since $a < b$, we must have $m_1 + \cdots + m_r > 0$ so $r \geq 1$.) Let n' be the equal number of occurrences of i and j in $ui^a vj^b$. Let t be the smallest index such that $m_1 + \cdots + m_t > n' - b$ (for $t = 0$, the left-hand term is 0 but $n' \geq b$; for $t = r$, the left-hand term is $n' - a$ and we know that $a < b$; so, t exists and $t > 0$). We write $m_t = m' + d$ such that $m_1 + \cdots + m_{t-1} + m' = n' - b$. So, $d > 0$. Note that $b - d = b - m_t + m' = n' - m_1 - \cdots - m_t = m_{t+1} + \cdots + m_r + a$. So, $b - d \geq a$. Clearly, $d \leq b$. We write $s = Hi^d Bi^a vj^d T$ with head $H = u_0 i^{m_1} u_1 i^{m_2} \cdots u_{t-1} i^{m'}$, body $B = u_t i^{m_{t+1}} \cdots i^{m_r} u_r$, and tail $T = j^{b-d} w$. Clearly, H has $n' - b$ occurrences of i and $Hi^d Bi^a v$ has $n' - b$ occurrences of j. Since s is optimal for D, $i^d Bi^a vj^d$ is optimal for $D|\neg H$. We note that B does not start with i (t is between 1 and r and u_t is nonempty and with no i) and that $i^a v$ is non-empty and with no j (either $a \neq 0$ or v is nonempty and with no j). We split $i^d Bi^a vj^d = i^d x_1 \cdots x_\ell i^a y_1 \cdots y_{\ell'} j^d$ where two consecutive blocks in the list $i^d, x_1, \ldots, x_\ell, i^a, y_1, \ldots, y_{\ell'}, j^d$ have no key in common. (For $a = 0$, we can always split so that x_ℓ and y_1 have no key in common by using the first term k of v which is not the last of u: we just take y_1 as a block of k's and x_ℓ as a block with no k.) We can apply Lemma 6.14 and obtain

$$\frac{C_D(i^d|\neg i^{n'-b})}{\Pr_D(i^d|\neg i^{n'-b})} \leq \frac{C_D(i^a|\neg i^{n'-a})}{\Pr_D(i^a|\neg i^{n'-a})} \leq \frac{C_D(y_1|\neg \cdots)}{\Pr_D(y_1|\neg \cdots)} \leq \frac{C_D(y_{\ell'}|\neg \cdots)}{\Pr_D(y_{\ell'}|\neg \cdots)} \leq \frac{C_D(j^d|\neg j^{n'-b})}{\Pr_D(j^d|\neg j^{n'-b})}$$

Since the first and the last terms are equal, all of them are equal. So, we can permute two consecutive blocks which have no index in common. Hence, we can propagate j^d earlier until it is stepped before i^a, since we know there is no other occurrence of j in the exchanged blocks. We obtain that

$$C_D(Hi^dBi^avj^dT) = C_D(Hi^dBj^di^avT)$$

as announced. □

Lemma 6.17 will be used in two ways.

1. For $s = u'j^cvj^bw$ with $c > 0$, $b > 0$, v with no i or j, and balanced occurrences of i and j in w, which has the same complexity as $s' = u'j^{b+c}vw$ (so, to apply the lemma we define $a = 0$, $u = u'j^c$, $k = j$, and $s = u'j^ci^0vj^bw$; all hypotheses are verified except v being non-empty, but the result is trivial for an empty v). This means that we can regroup j^c and j^b when they are separated by a v with no i and followed by a balanced tail w.

2. For $s = ui^avj^bw$ with $0 < a < b$, v with no i or j, and balanced occurrences of i and j in w, which has the same complexity as $s' = uj^{b-a}i^avj^aw$. This means that we can balance i^a and j^b when there are separated by a v with no i or j and followed by a balanced tail w.

In what follows, we say that a strategy is in a *normal form* if for all t, $i \mapsto \#\mathsf{occ}_{s_1\cdots s_t}(i)$ is a non-increasing function, i.e. $\#\mathsf{occ}_{s_1\cdots s_t}(i) \geq \#\mathsf{occ}_{s_1\cdots s_t}(i+1)$ for all i. For instance, 1112322133 is normal as the number of $\mathsf{STEP}(1)$ is at no time lower than the number of $\mathsf{STEP}(2)$ and the same for the number of $\mathsf{STEP}(2)$ and $\mathsf{STEP}(3)$.

Since all distributions are the same, all strategies can be rewritten into an equivalent one in a normal form: for this, for the smallest t such that there exists i such that $\#\mathsf{occ}_{s_1\cdots s_t}(i) < \#\mathsf{occ}_{s_1\cdots s_t}(i+1)$, it must be that $s_t = i+1$ and $\#\mathsf{occ}_{s_1\cdots s_{t-1}}(i) = \#\mathsf{occ}_{s_1\cdots s_{t-1}}(i+1)$. We can permute all values i and $i+1$ in the tail $s_ts_{t+1}\cdots$ and obtain an equivalent strategy on which the function becomes non-increasing at step t and is unchanged before. By performing enough such rewriting, we obtain an equivalent strategy in normal form. For instance, 12231332 is not normal. The smallest t is $t = 3$ when we make a second $\mathsf{STEP}(2)$ while we only did a single $\mathsf{STEP}(1)$. So, we permute 1 and 2 at this time and obtain 12132331. Then, we have $t = 7$ and permute 2 and 3 to obtain 12132321. Then, again $t = 7$ to permute 1 and 2 to obtain 12132312 which is normal.

We now prove Th. 6.16.

Proof of Th. 6.16. Let s be an optimal strategy. Due to the assumptions, it must be finite. We assume w.l.o.g. that s is in normal form. We note that we can always complete s in a form $s2^{a_2}3^{a_3}\cdots$ so that the final strategy has exactly n occurrences of each i. So, we assume w.l.o.g. that s has equal number of occurrences. We write $s = 1^{m_1}x_11^{m_2}x_2\cdots 1^{m_r}x_r$ where the x_t's are non-empty and with no 1 inside.

As detailed below, we rewrite x_r (and push some steps earlier in x_{r-1}) so that we obtain a permutation of the blocks $2^{m_r}, \ldots, |D|^{m_r}$. The rewriting is done by preserving the probability of success (which is 1) and the complexity (which is the optimal complexity). Then, we do the same operation in x_{r-1} and continue until x_1. When we are done, each x_t becomes a permutation of the blocks $2^{m_t}, \ldots, |D|^{m_t}$. Finally, we normalize the obtained rewriting of s and obtain the result.

We assume that s has already been rewritten so that for each $t' = t+1, \ldots, r$, the $x_{t'}$ sub-strategy is a permutation of the blocks $2^{m_{t'}}, \ldots, |D|^{m_{t'}}$. We explain now how to rewrite x_t. We make a loop for $j = 2$ to $|D|$. In the loop, we first regroup all blocks of j's by using Lemma 6.17 with $i = 1$: while we can write $x_t = u'j^c v j^b w'$ where $c > 0$, $b > 0$, v is non-empty with no j, and w' has no j, we write $u = 1^{m_1} x_1 1^{m_2} x_2 \cdots 1^{m_t} u'$ and $w = w' 1^{m_{t+1}} x_{t+1} \cdots 1^{m_r} x_r$, and set $a = 0$ and $i = 1$. This rewrites $x_t = u' j^{b+c} v w'$ by preserving the complexity and making a permutation. When this while loop is complete, we can only find a single block of j's in x_t and write $x_t = v j^b w'$, where v and w' have no j. So, we apply again Lemma 6.17 to balance 1^{m_t} and j^b: we write $u = 1^{m_1} x_1 1^{m_2} x_2 \cdots x_{t-1}$ and $w = w' 1^{m_{t+1}} x_{t+1} \cdots 1^{m_r} x_r$, and set $a = m_t$ and $i = 1$. This rewrites $1^{m_t} x_t$ to $j^{b-m_t} 1^{m_t} v j^{m_t} w'$ by preserving the complexity and making a permutation. So, this rewrites x_t to $v j^{m_t} w'$ and x_{t-1} to $x_{t-1} j^{b-m_t}$. When the loop of j is complete, x_t is a permutation of the blocks $2^{m_t}, \ldots, |D|^{m_t}$.

Interestingly, the sequence $m_1, \ldots, m_r$ is unchanged from our starting optimal normal full strategy s. If we rather start from an optimal full strategy s which is not in normal form, we can still see how to obtain this sequence: for each t, $m_1 + \cdots + m_t$ is the next record number of steps for an attack i after the $m_1 + \cdots + m_{t-1}$ record. That is the number of steps for the attack i when s decides to move to another attack. □

We provide toy examples below.

Example 6.4.6. *We take $D = (D_1, D_2)$ with $D_1 = D_2 = (\frac{3}{5}, \frac{9}{25}, \frac{1}{50}, \frac{1}{50}) = [\frac{3}{5}, \frac{18}{20}, \frac{1}{2}, 1]$. Here are the complexities of some full strategies.*

$$\begin{aligned}
C_D(1111) &= \frac{146}{100} = 1.46 \\
C_D(12111) &= \frac{792}{500} = 1.584 \\
C_D(11211) &= \frac{732}{500} = 1.464 \\
C_D(121211) &= \frac{7892}{5000} = 1.5784 \\
C_D(112211) &= \frac{7292}{5000} = 1.4584
\end{aligned}$$

so the last strategy is the best one. Notice that this is also a greedy strategy.

Example 6.4.7. *We take $D = (D_1, D_2)$ with $D_1 = D_2 = (\frac{70}{100}, \frac{20}{100}, \frac{5}{100}, \frac{3}{100}, \frac{1}{100}, \frac{1}{100}) =$*

$[\frac{70}{100}, \frac{2}{3}, \frac{1}{2}, \frac{3}{5}, \frac{1}{2}, 1]$. *Here are the complexities of some full strategies.*

$$\begin{aligned}
C_D(111111) &= 1.48\\
C_D(1211111) &= 1.44\\
C_D(12121111) &= 1.438\\
C_D(121212111) &= 1.439\\
C_D(121122111) &= 1.444
\end{aligned}$$

so $s = 12121111$ *is the best one. For this example, we have that the optimal strategy requires* $m_1 = 1$*,* $m_2 = 1$ *and* $m_3 = 4$*. It is also greedy.*

6.4.3 Finding the Optimal m

We provide here a simple criterion for the optimal m of Th. 6.13.

Lemma 6.18. *We let* $D_1 = (p_1, \ldots, p_n) = [p'_1, \ldots, p'_n]$ *be a distribution and define* $D = (D_1, D_1, \ldots)$*. Let* m *be such that* $s = 1^m 2^m \cdots$ *is an optimal strategy based on Th. 6.13. We have* $\frac{1}{p'_m} \leq C_D(1^m 2^m \cdots) \leq \frac{1}{p'_{m+1}}$.

Proof. We let $s = 2^m 3^m \cdots$ We know that $C_D(1^{m+1}s) \geq C_D(1^m s)$ since $1^m s$ is optimal. So,

$$\begin{aligned}
0 &\leq C_D(1^{m+1}s) - C_D(1^m s)\\
&= (1 - \Pr_D(1^m))(C_D(1s|\neg 1^m) - C_D(s))\\
&= (1 - \Pr_D(1^m))(1 - p'_{m+1} \cdot C_D(s))
\end{aligned}$$

from which we deduce $\frac{1}{p'_{m+1}} \geq C_D(s)$. Similarly, we have

$$\begin{aligned}
0 &\geq C_D(1^m s) - C_D(1^{m-1}s)\\
&= (1 - \Pr_D(1^{m-1}))(C_D(1s|\neg 1^{m-1}) - C_D(s))\\
&= (1 - \Pr_D(1^{m-1}))(1 - p'_m \cdot C_D(s))
\end{aligned}$$

from which we deduce $\frac{1}{p'_m} \leq C_D(s)$. □

We note that if $p_m = p_{m+1}$, then

$$p'_{m+1} = \frac{p_{m+1}}{p_{m+1} + \cdots + p_n} = \frac{p_m}{p_{m+1} + \cdots + p_n} > \frac{p_m}{p_m + p_{m+1} + \cdots + p_n} = p'_m$$

which is impossible (given the result from Lemma 6.18). Consequently, we must have $p_m \neq p_{m+1}$. So, in distributions when we have sequences of equal probabilities p_t, we can just look at the largest index t in the sequence as a possible candidate for being the value m.

Lemma 6.18 has an equivalent for Th. 6.16.

Lemma 6.19. *We let $D_1 = (p_1, \ldots, p_n) = [p'_1, \ldots, p'_n]$ be a distribution of support n and $D = (D_1, D_1, \ldots, D_1)$ of size $|D|$. Let $m_1, \ldots, m_r$ be such that $s = 1^{m_1} \cdots |D|^{m_1} 1^{m_2} \cdots |D|^{m_2} \cdots$ is an optimal strategy based on Th. 6.16. For all t, we have $\frac{1}{p'_{m_t}} \leq \frac{C_D(1^{m_t})}{\Pr_D(1^{m_t})} \leq \frac{1}{p'_{m_t+1}}$.*

The proof is essentially the same as for Lemma 6.18. We just have to compare the optimal strategy with

$$\cdots 1^{m_{t-1}} \cdots |D|^{m_{t-1}} 1^{m_t+1} 2^{m_t} \cdots |D|^{m_t} 1^{m_{t+1}-1} 2^{m_{t+1}} \cdots |D|^{m_{t+1}} 1^{m_{t+2}} \cdots |D|^{m_{t+2}} \cdots$$

and

$$\cdots 1^{m_{t-1}} \cdots |D|^{m_{t-1}} 1^{m_t-1} 2^{m_t} \cdots |D|^{m_t} 1^{m_{t+1}+1} 2^{m_{t+1}} \cdots |D|^{m_{t+1}} 1^{m_{t+2}} \cdots |D|^{m_{t+2}} \cdots$$

6.5 Applications

We apply our results on two applications: LPN and password guessing.

6.5.1 Solving Sparse LPN

We model the LPN problem in our STEP game. We use the noise bits as the keys the adversary $\mathcal{A}$ is trying to guess.

Recall that in the list of LPN solving algorithms, we have the one that guesses that the noise is 0 and runs a Gaussian elimination until it finds the correct solution. This algorithm works with complexity $\mathsf{poly} \cdot (1-\tau)^{-k}$ (See Section 3.8.3). So this algorithm is better than the BKW variants that work with a complexity of $\mathsf{poly} \cdot 2^{\frac{k}{\log_2 k}}$ as soon as $\tau < \frac{\ln 2}{\log_2 k}$, and in particular for $\tau = k^{-\frac{1}{2}}$ which is the case for some applications [Ale03, DP12].

The Gaussian elimination algorithm is reduced to finding a k-bit noise vector. It guesses that this vector is 0. If this does not work, the algorithm tries again with new LPN queries. We can see this as guessing at least one k-bit biased vector K_i which follows the distribution $D_i = \mathsf{Ber}_\tau^k$ defined by $\Pr[K_i = v] = \tau^{\mathsf{HW}(v)}(1-\tau)^{k-\mathsf{HW}(v)}$ in our framework. The most probable vector is $v = 0$ which has probability $\Pr[K_i = 0] = (1-\tau)^k$. The above algorithm corresponds to trying $K_1 = 0$ then $K_2 = 0$, ... i.e., the strategy $123 \cdots$ in our framework. We can wonder if there is a better $1^m 2^m 3^m \cdots$. This is the problem we study below. We will see that the answer is no: using $m = 1$ is the best option as soon as τ is less than $\frac{1}{2} - \varepsilon$ for $\varepsilon = \frac{\ln 2}{2k}$ which is pretty small.

For instance, for $\mathsf{LPN}_{768, \frac{1}{\sqrt{768}}}$ we obtain $C_D(12\cdots) = 2^{41}$. I.e., 2^{41} calls to the STEP command which corresponds to collecting k LPN queries and making a Gaussian elimination to recover the secret based on the assumption that the error bits are all 0. If we add up the cost of running Gaussian elimination in order to recover the secret, we obtain a complexity of 2^{70}. This outperforms all the BKW variants and proves that $\mathsf{LPN}_{768, \frac{1}{\sqrt{768}}}$ is not a secure instance for a 80-bit security. Furthermore, this algorithm outperforms even the covering code algorithm [GJL14].

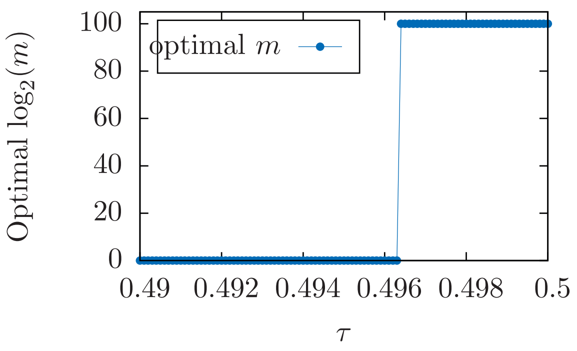

Figure 6.2: The change of optimal m for solving $\mathsf{LPN}_{100,\tau}$

D_i is a composite distribution of uniform ones in the sense defined in Lemma 6.15. Namely, $D_i = \sum_{w=0}^{k} \tau^k(1-\tau)^{k-w}U_w$ where U_w is uniform of support $\binom{k}{w}$. By Theorem 6.13, we know that there exists a magic m for which the strategy $s = 1^m 2^m \cdots$ is optimal. The analysis of composite distributions further says that m must be of form $m = B_w = \sum_{i=0}^{w} \binom{k}{i}$ for some magic w. Let c_m be the complexity of $1^m 2^m \cdots$. A value $w = k$, i.e. $m = n$ corresponds to the exhaustive search of the noise bits. For $w = 0$, i.e. $m = 1$, the adversary assumes that the noise is 0 every time he receives k queries from the LPN oracle.

We first computed experimentally the optimal m for the $\mathsf{LPN}_{100,\tau}$ instance where we take $0 < \tau < \frac{1}{2}$. The magic m takes the value 1 for a τ which is not close to $\frac{1}{2}$. As shown on Fig. 6.2, it changes to $n = 2^{100}$ around the value $\tau = 0.4965$. This boundary between two different strategies corresponds to the value $\tau = \frac{1}{2} - \frac{\ln 2}{2k}$ computed in our analysis below. Interestingly, there is no intermediate optimal m between 1 and n.

For cryptographic parameters, c_1 is optimal. The optimal w depends on τ. The case when τ is lower than $\frac{1}{k}$ is not interesting as it is likely that no error occurs so all w lead to a complexity which is very close to 1. Conversely, for $\tau = \frac{1}{2}$, the exhaustive search has a complexity of $c_n = \frac{1}{2}(2^k + 1)$ and $w = 0$ has a complexity of $c_1 = 2^k$. Actually, D_i is uniform in this case and we know that the optimal m completes batches of equal consecutive probabilities. So, the optimal strategy is the exhaustive search.

We now show that for $\tau < 0.16$, the best strategy is obtained for $w = 0$.

Below, we use $p_{B_w} = \tau^w(1-\tau)^{k-w}$ and $c_1 = (1-\tau)^{-k}$.

Let w_c be a threshold weight and let $\alpha = \Pr(1^{B_{w_c}})$. For $0 < w \le w_c$, due to Lemma 6.18, if c_{B_w} is optimal we have

$$c_{B_w} \ge \frac{1}{p'_{B_w}} = \frac{\Pr_D(\neg 1^{B_w - 1})}{p_{B_w}} \ge \frac{\Pr_D(\neg 1^{B_{w_c}})}{p_{B_w}} = \frac{1-\alpha}{p_{B_w}} = \frac{1-\alpha}{\left(\frac{\tau}{1-\tau}\right)^w} c_1 \ge \frac{1-\alpha}{\frac{\tau}{1-\tau}} c_1 \, .$$

For $\tau < 0.16$, we have $\frac{\tau}{1-\tau} < 0.20$. So, if $\alpha \le \frac{4}{5}$ we obtain $c_{B_w} > c_1$. This contradicts that w is optimal. For $w_c = \tau k$, the Central Limit Theorem gives us that $\alpha \approx \frac{1}{2}$ which is less than $\frac{4}{5}$. So, no w such that $0 < w \leq \tau k$ is optimal.

Now, for $w \geq w_c$, we have

$$c_w = \frac{C_D(1^{B_w})}{\Pr_D(1^{B_w})} \geq C_D(1^{B_w}) = \sum_{i=1}^{B_w} ip_i + B_w \Pr_D(\neg 1^{B_w}) \geq B_{w_c} \Pr_D(\neg 1^{B_{w_c}}) = (1-\alpha)B_{w_c}$$

By using the bound $B_{w_c} \geq \left(\frac{k}{w_c}\right)^{w_c}$, for $w_c = \tau k$ we have $\alpha \approx \frac{1}{2}$ and we obtain $c_w \geq \frac{1}{2}\tau^{-\tau k}$. We want to compare this to $c_1 = (1-\tau)^{-k}$. We look at the variations of the function $\tau \mapsto -k\tau \ln \tau - \ln 2 + k \ln(1-\tau)$. We can see by derivating twice that for $\tau \in [0, \frac{1}{2}]$, this function increases then decreases. For $\tau = 0.16$, it is positive. For $\tau = \frac{1}{k}$, it is also positive. So, for $\tau \in [\frac{1}{k}, 0.16]$, we have $c_{B_w} \geq c_1$.

Therefore, for all $\tau < 0.16$, c_1 is the best complexity so $m = 0$ is the magic value. Experiment shows that this remains true for all $\tau < \frac{1}{2} - \frac{\ln 2}{2k}$. Actually, we can easily see that c_1 becomes lower than $\frac{2^k+1}{2}$ for $\tau \approx \frac{1}{2} - \frac{\ln 2}{2k}$. We will discuss this in Section 6.6.

Solving LPN with $\mathcal{O}(k)$ queries. We now concentrate on the $m = n$ case to limit the query complexity to $\mathcal{O}(k)$. (In our framework, we need only k queries but we would practically need more to check that we did find the correct value.) So, we estimate the complexity of the full exhaustive search on one error vector x of k bits for LPN, i.e., $C_D(1^n)$. If p_t is the probability that x is the t-th enumerated vector, we have $C_D(1^n) = \sum_{t=1}^n tp_t$. For t between $B_{w-1}+1$ and B_w, the sum of the p_t's is the probability that we have exactly w errors. So, $C_D(1^n) \leq \sum_{w=0}^k B_w \Pr[w \text{ errors}]$. We approximate $\Pr[w \text{ errors}]$ to the continuous distribution. So, the Hamming weight has a normal distribution, with mean $k\tau$ and standard deviation $\sigma = \sqrt{k\tau(1-\tau)}$. We do the same for $B_w \approx \frac{2^k}{\sqrt{2\pi}} \int_{-\infty}^{\frac{2w-k}{\sqrt{k}}} e^{-\frac{v^2}{2}}\, dv$. With the change of variables $w = k\tau + t\sigma$, we have

$$\begin{aligned} C_D(1^n) &\leq \sum_{w=0}^{k} B_w \Pr[w \text{ errors}] \\ &\approx \frac{2^k}{2\pi} \int_{-\infty}^{+\infty} \left(\int_{-\infty}^{\frac{2w-k}{\sqrt{k}}} e^{-\frac{v^2}{2}}\, \mathrm{d}v \right) \frac{1}{\sigma} e^{-\frac{(w-k\tau)^2}{2\sigma^2}}\, \mathrm{d}w \\ &= \frac{2^k}{2\pi} \iint_{v \leq \frac{2k\tau - k + 2t\sigma}{\sqrt{k}}} e^{-\frac{t^2+v^2}{2}}\, \mathrm{d}v\, \mathrm{d}t \end{aligned}$$

The distance between the origin $(t, v) = (0, 0)$ and the line $v = \frac{2k\tau - k + 2t\sigma}{\sqrt{k}}$ is

$$d = \sqrt{k} \frac{1 - 2\tau}{\sqrt{1 + 4\tau(1-\tau)}}$$

τ	$\log_2(C_D(1^n))$	$\log_2\left(\frac{2^k}{d\sqrt{2\pi}}e^{-\frac{d^2}{2}}\right)$
0.1	1350.04	1314.81
0.125	1458.86	1429.33
0.25	1794.57	1788.49
0.4	1966.67	1966.55

Table 6.1: $\log_2(C_D(1^n))$ vs. $\log_2\left(\frac{2^k}{d\sqrt{2\pi}}e^{-\frac{d^2}{2}}\right)$ for $k = 2000$

By rotating the region on which we sum, we obtain

$$C_D(1^n) \approx \frac{2^k}{2\pi}\iint_{x\geq d} e^{-\frac{x^2+y^2}{2}}\,\mathrm{d}x\,\mathrm{d}y = \frac{2^k}{\sqrt{2\pi}}\int_d^{+\infty} e^{-\frac{x^2}{2}}\,\mathrm{d}x \sim \frac{2^k}{d\sqrt{2\pi}}e^{-\frac{d^2}{2}}$$

On Fig. 6.3 we can see that this approximation of $C_D(1^n)$ is very good for $\tau = k^{-\frac{1}{2}}$.

So, the complexity $C_D(1^n)$ is asymptotically $2^{k\left(1-\frac{1}{2\ln 2}\right)+\mathcal{O}(\sqrt{k})}$. Interestingly, the dominant part of $\log_2 C_D(1^n)$ is $0.2788 \times k$ and does not depend on τ as long as $\frac{1}{k} \ll \tau \ll \frac{1}{2}$. Although very good for the low k that we consider, this approximation of $C_D(1^n)$ deviates, probably because of the imprecise approximation of the B_w's. Next, we derive a bound which is much higher but asymptotically better (the curves crossing for $k \approx 50\,000$). We now use the bound $B_w \leq k^w$ and do the same computation as before. We have

$$\begin{aligned}
C_D(1^n) &\leq \sum_{w=0}^{k} k^w \Pr[w \text{ errors}] \\
&\approx \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} k^{k\tau+t\sigma}e^{-\frac{t^2}{2}}\,\mathrm{d}t \\
&= \frac{e^{\frac{1}{2}(\sigma\ln k)^2+k\tau\ln k}}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} e^{-\frac{(t-\sigma\ln k)^2}{2}}\,\mathrm{d}t \\
&= e^{\frac{1}{2}(\sigma\ln k)^2+k\tau\ln k}
\end{aligned}$$

So, $C_D(1^n) = e^{\frac{1}{2}\sqrt{k}(\ln k)^2+\mathcal{O}(\sqrt{k}\ln k)}$ for $\tau = k^{-\frac{1}{2}}$. Recall that Lyubashevsky [Lyu05] has a complexity of $e^{\mathcal{O}\left(\frac{k}{\ln\ln k}\right)}$ (See Section 3.9). We obtain a better result that is asymptotically better and that requires $\mathcal{O}(k)$ queries instead of $k^{1+\varepsilon}$. However, this new bound for $C_D(1^n)$ is very loose.

Outside the scenario of a sparse LPN, we display in Figure 6.4 the logarithmic complexity to solve LPN in our STEP game when the noise parameter is constant.

Comparing $\log_2(C_D(1^n))$ with our approximation, i.e. $\log_2\left(\frac{2^k}{d\sqrt{2\pi}}e^{-\frac{d^2}{2}}\right)$, we obtain the following results which validate our approximations (See Table 6.1).

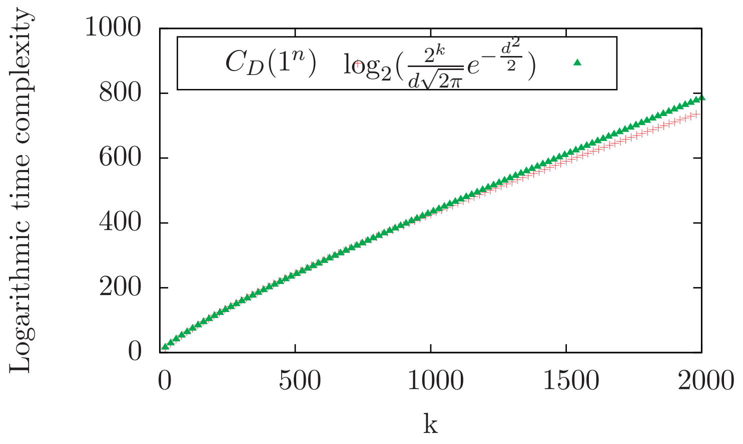

Figure 6.3: $\log_2(C_D(1^n))$ vs. $\log_2\left(\frac{2^k}{d\sqrt{2\pi}}e^{-\frac{d^2}{2}}\right)$ for $\tau = k^{-\frac{1}{2}}$

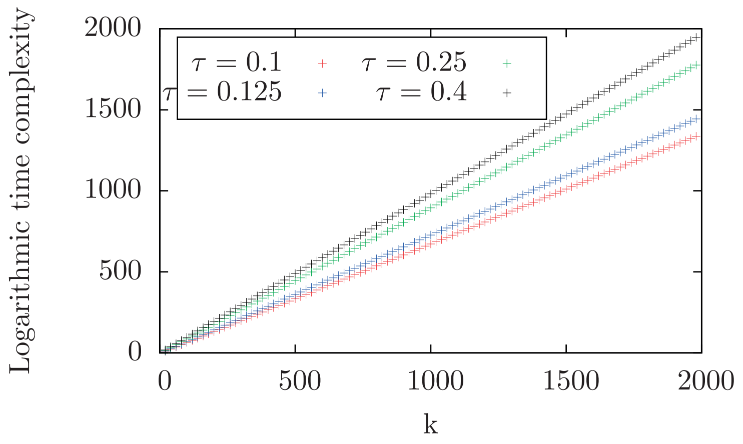

Figure 6.4: $\log_2(C_D(1^n))$ for constant τ

6.5.2 Password Recovery

There is numerous information nowadays accounting attacks and leaks of passwords from different famous companies. From these leaks, the community has studied what are the worst passwords used by the users. Having in mind these statistics, we are interested to see what is the best strategy of an outsider that tries to get access to a system having access to a list of users. The goal of the attacker is to hack one account. He can try to hack several accounts. Within our framework, we compute to see what is the optimal m for the strategy $1^m 2^m \cdots$. In this given scenario, the strategy corresponds to making m guesses for each user until it reaches the end of the list and starting again with new guesses.

We consider the statistics that we have found for the 10 000 Top Passwords[2] and the one done for the database with passwords in clear from the RockYou hack[3]. Studies on the distribution of user's passwords were also done in [DMR10, WACS10, Bon12, Sch06]. The first case-study analyses what are the top 10 000 passwords from a total 6.5 million username-passwords that leaked. The most frequent passwords are the following:

password	$p_1 = 0.00493$
123456	$p_2 = 0.00400$
12345678	$p_3 = 0.00133$
1234	$p_4 = 0.00089$

In the case of the RockYou hack, where 32 million of passwords were leaked, we find that the most frequent passwords and their probability of usage is:

123456	$p_1 = 0.009085$
12345	$p_2 = 0.002471$
123456789	$p_3 = 0.002400$
Password	$p_4 = 0.000194$

Moreover, approximately 20% of the users used the most frequent 5 000 passwords. What these statistics show is that users frequently choose poor and predictable passwords. While dictionary attacks are very efficient, we study here the case where the attacker wants to minimize the number of trials until he gets access to the system, with no pre-computation done. By using our formulas of computing $C_D(1^m 2^m \cdots)$, we obtain in both of the above distributions that $m = 1$ is the optimal one. This means that the attacker tries for each username the most probable password and in average, after couple of hundred of users (for the two studies we obtain C_D to be about 203 and about 110), he will manage to access the system. We note that having $m = 1$ is very nice as for the typical password guessing scenario, we need to have a small m to avoid complications of blocking accounts and triggering an alarm that the system is under an attack.

[2] `https://xato.net/passwords/more-top-worst-passwords/#.VNiORvnF-xW`
[3] `http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf`

6.6 On the Phase Transition

Given the experience of the previous applications, we can see that for "regular" distributions, the optimal m falls from $m = n$ to the minimal m as the bias of the distribution increases. We let n_1 be such that $p_1 = p_2 = \cdots = p_{n_1} \neq p_{n_1+1}$ and n_2 be such that $p_{n_1+1} = \cdots = p_{n_1+n_2} \neq p_{n_1+n_2+1}$. Due to Lemma 6.18, the magic value m can only be n_1, $n_1 + n_2$, or more. We study here when the curves of $C_D(1^{n_1}2^{n_1}\cdots)$, $C_D(1^{n_1+n_2}2^{n_1+n_2}\cdots)$, and $C_U(1^n) = \frac{n+1}{2}$ cross each other.

Lemma 6.20. *We consider a composite distribution $D_1 = \alpha U_1 + \beta U_2 + (1-\alpha-\beta)D'$, where U_1 and U_2 are uniform of support n_1 and n_2. For U uniform, we have*

$$C_D(1^{n_1}2^{n_1}\cdots) \leq C_D(1^{n_1+n_2}2^{n_1+n_2}\cdots) \iff \alpha - \beta\frac{n_1}{n_2} \geq \alpha\left(\alpha + \beta\frac{1-n_1/n_2}{2}\right)$$
$$C_D(1^{n_1}2^{n_1}\cdots) \leq C_U(1^n) \iff \frac{n/n_1+1}{2} \geq \frac{1}{\alpha}$$

Note that for $2^{-H_\infty} \geq \frac{2}{n}$, we have $\frac{\alpha}{n_1} \geq \frac{2}{n}$ so the second property is satisfied.

As an example, for $n_1 = n_2 = 1$, the first condition becomes $\alpha - \beta \geq \alpha^2$ which is the case of all the distribution we tried for password recovery. The second condition becomes $2^{-H_\infty} \geq \frac{2}{n+1}$, which is also always satisfied.

For LPN, we have $n_1 = 1$, $n_2 = k$, $\alpha = (1-\tau)^k$, and $\beta = n_2\tau(1-\tau)^{k-1}$. The first and second conditions become

$$(1-\tau)^k \leq \frac{1-2\tau}{1+\frac{k-3}{2}\tau} \qquad \text{and} \qquad (1-\tau)^k \geq \frac{2}{2^k+1}$$

respectively. They are always satisfied unless τ is very close to $\frac{1}{2}$: by letting $\tau = \frac{1}{2} - \varepsilon$ with $\varepsilon \to 0$, the right-hand term of the first condition is asymptotically equivalent to $\frac{8\varepsilon}{k+1}$ and the left-hand term tends towards 2^{-k}. The balance is thus for $\tau \approx \frac{1}{2} - \frac{k+1}{8}2^{-k}$. The second condition gives

$$\tau \leq 1 - \left(\frac{2^k+1}{2}\right)^{-\frac{1}{k}} = \frac{1}{2} - \frac{\ln 2}{2k} - o\left(\frac{1}{k}\right)$$

So, we can explain the phase transition in $\mathsf{LPN}_{k,\tau}$ as follows: if we make τ decrease from $\frac{1}{2}$, for each fixed m, the complexity of all possible $C_D(1^m)$ smoothly decrease. The function for $m = n_1$ crosses the one of $m = n_1 + n_2$ before it crosses $\frac{n+1}{2}$ which is close to the value of the one for $m = n$. So, the curve for $m = n_1$ becomes interesting *after* having beaten the curve for $m = n_1 + n_2$. This proves that we never have a magic m equal to $n_1 + n_2$. Presumably, it is the case for all other curves as well. This explains the abrupt fall from $m = n$ to $m = 1$ which we observed on Fig. 6.2.

Proof. We have

$$C_D(1^{n_1}2^{n_1}\cdots) = \frac{C_D(1^{n_1})}{\Pr_D(1^{n_1})} = \frac{\alpha\frac{n_1+1}{2} + (1-\alpha)n_1}{\alpha}$$

and

$$C_D(1^{n_1+n_2}2^{n_1+n_2}\cdots) = \frac{C_D(1^{n_1+n_2})}{\Pr_D(1^{n_1+n_2})} = \frac{\alpha\frac{n_1+1}{2} + \beta\left(n_1 + \frac{n_2+1}{2}\right) + (1-\alpha-\beta)(n_1+n_2)}{\alpha+\beta}$$

so

$$\begin{aligned} \frac{C_D(1^{n_1})}{\Pr_D(1^{n_1})} &\le \frac{C_D(1^{n_1+n_2})}{\Pr_D(1^{n_1+n_2})} \iff \\ \frac{\alpha\frac{n_1+1}{2} + (1-\alpha)n_1}{\alpha} &\le \frac{\alpha\frac{n_1+1}{2} + \beta\left(n_1 + \frac{n_2+1}{2}\right) + (1-\alpha-\beta)(n_1+n_2)}{\alpha+\beta} \iff \\ \alpha - \beta\frac{n_1}{n_2} &\ge \alpha\left(\alpha + \beta\frac{1-n_1/n_2}{2}\right) \end{aligned}$$

For the second property, we have

$$\begin{aligned} C_D(1^{n_1}2^{n_1}\cdots) \le C_U(1^n) &\iff \frac{C_D(1^{n_1})}{\Pr_D(1^{n_1})} \le C_U(1^n) \\ &\iff \frac{\alpha\frac{n_1+1}{2} + (1-\alpha)n_1}{\alpha} \le \frac{n+1}{2} \\ &\iff \frac{n/n_1+1}{2} \ge \frac{1}{\alpha} \end{aligned}$$

□

Our framework enables the analysis of different strategies to sequentialize algorithms where the objective is to make one succeed as soon as possible.

When the algorithms have the same distribution and are unlimited in number, the optimal strategy is of form $1^m2^m\cdots$ for some magic m. As the distribution becomes biased, we observe a phase transition from the regular single-algorithm run 1^n (i.e., $m = n$) to the single-step multiple algorithms $123\cdots$ (i.e., $m=1$) which is very abrupt in the application we considered: LPN and password recovery. Besides the 2 problems we have studied here, we believe that our results can prove to be useful in other cryptographic applications.

Chapter 7

Conclusion and Further Work

7.1 Conclusion

This thesis presents an algorithmic study of LPN. In Chapter 3, we gave an overview of the existing LPN solving algorithms. By formally analysing them we introduced several improvements. An LPN solving algorithm works in two phases: a reduction and a solving phase. During the reduction phase, we reduce the size of the secret through several reduction techniques. The solving phase recovers part of the secret through a solving technique. We validated our heuristic results with practice in Chapter 4. This allowed us to asses the security offered by each algorithm and to propose secure parameters to be used in practice by the LPN based cryptographic primitives.

In Chapter 5, we further improved the complexity of an LPN solving algorithm by saying how the reduction steps should be organized in order to maximize their use. We designed an algorithm that receives at input an LPN instance and provides at output the steps of the algorithm that solves the LPN instance and minimizes the time complexity. This algorithm allows us to automatize an LPN solving algorithm. It also provides flexibility as our algorithm could be further adapted and automatized if new reduction techniques are introduced. The results we obtained bring improvements to the existing work and to the best of our knowledge we have the best algorithm for solving LPN. A similar analysis in optimizing the reduction steps is done for LWE [KF15].

In Chapter 6, we provided the STEP framework that allows us to improve the existing results in the area of solving LPN for the case where the initial number of queries are limited.

7.2 Future Work

Recall that for the BKW* algorithm, we made the assumption that the noise bits are independent. This assumption is supported by the practical results that show that the performance of the algorithm is not affected by it. An interesting theoretical result in this direction would be to use martingales to prove we do not need the independence in order to apply the Chernoff bound or the Central Limit Theorem.

The performance of an LPN solving algorithm can be improved in several ways. First, one could study whether *code-reduce* reductions applied in cascade can bring improvement compared to *code-reduce* applied once. We recall that the distribution of the secret changes in this case and we did not cover this scenario. Also, one could extend the pool of codes used by this reduction. We already have indications that this can improve the existing results. Using a larger code, instead of concatenating several small codes, can give us a bigger, better bias. Thus, the solving method would require less number of queries. This can decrease the number of initial queries and the overall performance of a solving algorithm can be improved.

A new reduction technique can bring another improvement. The introduction of the *code-reduce* brought some fresh air in the research area of LPN solving algorithms and there is no result to indicate that we exhausted all our methods. The algorithm presented in Chapter 5 can easily incorporate a new technique and show how this new tool can be optimally used.

For the STEP game, there are several interesting questions that could be analysed. Our game considers an adversary that runs his attack sequentially and wants to succeed in just one attack. A more realistic scenario to consider is the one where the adversary can run several attacks in parallel, e.g. i attacks. Given this, one could check what is the best strategy and if the complexity of a parallel optimal strategy achieves to be i times better than a sequential optimal strategy. Also, the goal of the adversary can be to succeed in j out of n attacks. In this case should the attacker focus on only one attack at once until he succeeds in j of them or should he apply a different strategy?

Our main results in the STEP game considers identical distributions. In the case where we have several different distributions, how to compare them and say which one is to be preferred? Is a uniform distribution with a small support better than a biased distribution with a larger support where the first candidates have a high probability to occur?

Final Words. To conclude, we want to emphasize on the importance of assessing the hardness of problems on which we rely in cryptography. This proves to be crucial in this moment when NIST is calling for candidates for post-quantum cryptography. While problems like factorization and discrete logarithm have a long history of cryptanalysis, some of the post-quantum hard problems lack this. A collaborative effort of the cryptography community is needed in order to have thorough study of the problems that will ensure the security of our data in the future.

List of Algorithms

List of Definitions

Bibliography

[ABC15] Gildas Avoine, Adrien Bourgeois, and Xavier Carpent. Analysis of rainbow tables with fingerprints. In Ernest Foo and Douglas Stebila, editors, *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*, volume 9144 of *Lecture Notes in Computer Science*, pages 356–374. Springer, 2015. *Cited on page:* 80.

[AC13] Gildas Avoine and Xavier Carpent. Optimal storage for rainbow tables. In Hyang-Sook Lee and Dong-Guk Han, editors, *Information Security and Cryptology - ICISC 2013 - 16th International Conference, Seoul, Korea, November 27-29, 2013, Revised Selected Papers*, volume 8565 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 2013. *Cited on page:* 80.

[ACF$^+$15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Des. Codes Cryptography*, 74(2):325–354, 2015. *Cited on page:* 22.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009. *Cited on pages:* 8 and 24.

[ADKF70] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of a directed graph. 1970. *Cited on page:* 57.

[AFFP14] Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy Modulus Switching for the BKW Algorithm on LWE. In Krawczyk [Kra14], pages 429–445. *Cited on page:* 35.

[AG11] Sanjeev Arora and Rong Ge. New Algorithms for Learning in Presence of Errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2011. *Cited on page:* 14.

[AJO05] Gildas Avoine, Pascal Junod, and Philippe Oechslin. Time-memory trade-offs: False alarm detection using checkpoints. In Subhamoy Maitra, C. E. Veni Madhavan, and Ramarathnam Venkatesan, editors, *Progress in Cryptology - INDOCRYPT 2005, 6th International Conference on Cryptology in India, Bangalore, India, December 10-12, 2005, Proceedings*, volume 3797 of *Lecture Notes in Computer Science*, pages 183–196. Springer, 2005. *Cited on page:* 80.

[Ale03] Michael Alekhnovich. More on Average Case vs Approximation Complexity. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 298–307. IEEE Computer Society, 2003. *Cited on pages:* 8, 14, 32, 47, and 97.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015. *Cited on page:* 35.

[BBDF08] Tsonka Stefanova Baicheva, Iliya Bouyukliev, Stefan M. Dodunekov, and Veerle Fack. Binary and ternary linear quasi-perfect codes with small dimensions. *IEEE Transactions on Information Theory*, 54(9):4335–4339, 2008. *Cited on pages:* 65 and 66.

[BCD06] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. HB^{++}: a Lightweight Authentication Protocol Secure against Some Attacks. In *Second International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SecPerU 2006), 29 June 2006, Lyon, France*, pages 28–33. IEEE Computer Society, 2006. *Cited on page:* 8.

[Ber] Daniel J. Bernstein. Optimizing linear maps modulo 2. `http://binary.cr.yp.to/linearmod2-20090830.pdf`. *Cited on page:* 57.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic Primitives Based on Hard Learning Problems. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993. *Cited on pages:* 8 and 10.

[BG14] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *Information Security and Privacy - 19th Australasian Conference, ACISP 2014, Wollongong, NSW, Australia, July 7-9, 2014. Proceedings*, volume 8544 of *Lecture Notes in Computer Science*, pages 322–337. Springer, 2014. *Cited on page:* 35.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 435–440. ACM, 2000. *Cited on pages:* 13, 14, 15, 16, and 17.

[BL12] Daniel J. Bernstein and Tanja Lange. Never Trust a Bunny. In Jaap-Henk Hoepman and Ingrid Verbauwhede, editors, *Radio Frequency Identification. Security and Privacy Issues - 8th International Workshop, RFIDSec 2012, Nijmegen, The Netherlands, July 2-3, 2012, Revised Selected Papers*, volume 7739 of *Lecture Notes in Computer Science*, pages 137–148. Springer, 2012. *Cited on pages:* 24 and 74.

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller Decoding Exponents: Ball-Collision Decoding. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011. *Cited on page:* 33.

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584. ACM, 2013. *Cited on pages:* 2 and 8.

[Bon12] Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 538–552. IEEE Computer Society, 2012. *Cited on page:* 102.

[BR10] Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2010. *Cited on page:* 35.

[BTV16] Sonia Bogos, Florian Tramèr, and Serge Vaudenay. On solving LPN using BKW and variants - Implementation and analysis. *Cryptography and Communications*, 8(3):331–369, 2016. *Cited on pages:* 13, 18, 21, 22, 24, 39, 56, 60, 64, 65, 76, and 81.

[BV15] Sonia Bogos and Serge Vaudenay. How to sequentialize independent parallel attacks? - biased distributions have a phase transition. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 704–731. Springer, 2015. *Cited on pages:* 14 and 79.

[BV16a] Sonia Bogos and Serge Vaudenay. Observations on the LPN Solving Algorithm from Eurocrypt'16. Cryptology ePrint Archive, Report 2016/451, 2016. `https://eprint.iacr.org/2016/451`. *Cited on pages:* 13, 32, and 77.

[BV16b] Sonia Bogos and Serge Vaudenay. Optimization of LPN Solving Algorithms. In Cheon and Takagi [CT16], pages 703–728. *Cited on pages:* 13, 14, 18, and 55.

[CHLL97] G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein. *Covering Codes*. North-Holland Mathematical Library. Elsevier Science, 1997. *Cited on page:* 65.

[CKJS85] Gérard D. Cohen, Mark G. Karpovsky, H. F. Mattson Jr., and James R. Schatz. Covering radius - survey and recent results. *IEEE Transactions on Information Theory*, 31(3):328–343, 1985. *Cited on pages:* 65 and 66.

[CNV13] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-middle: Improved MITM attacks. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 222–240. Springer, 2013. *Cited on page:* 35.

[CT65] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):pp. 297–301, 1965. *Cited on pages:* 11 and 20.

[CT16] Jung Hee Cheon and Tsuyoshi Takagi, editors. *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, 2016.

[CTIN09] José Carrijo, Rafael Tonicelli, Hideki Imai, and Anderson C. A. Nascimento. A Novel Probabilistic Passive Attack on the Protocols HB and HB$^+$. *IEICE Transactions*, 92-A(2):658–662, 2009. *Cited on page:* 36.

[DB14] Nikesh S. Dattani and Nathaniel Bryans. Quantum factorization of 56153 with only 4 qubits. *CoRR*, abs/1411.6758, 2014. *Cited on page:* 2.

[Des77] Des. Data encryption standard. In *In FIPS PUB 46, Federal Information Processing Standards Publication*, pages 46–2, 1977. *Cited on page:* 1.

[DMN12] Nico Döttling, Jörn Müller-Quade, and Anderson C. A. Nascimento. IND-CCA Secure Cryptography Based on a Variant of the LPN Problem. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 485–503. Springer, 2012. *Cited on page:* 8.

[DMR10] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password strength: An empirical analysis. In *INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA*, pages 983–991. IEEE, 2010. *Cited on page:* 102.

[DP12] Ivan Damgård and Sunoo Park. Is Public-Key Encryption Based on LPN Practical? *IACR Cryptology ePrint Archive*, 2012:699, 2012. *Cited on pages:* 8, 14, 32, 47, and 97.

[DSP07] Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved meet-in-the-middle attacks on reduced-round DES. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007. *Cited on page:* 35.

[DTV15] Alexandre Duc, Florian Tramèr, and Serge Vaudenay. Better algorithms for LWE and LWR. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 173–202. Springer, 2015. *Cited on pages:* 15 and 22.

[DV13] Alexandre Duc and Serge Vaudenay. HELEN: A Public-Key Cryptosystem Based on the LPN and the Decisional Minimal Distance Problems. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on*

Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings, volume 7918 of *Lecture Notes in Computer Science*, pages 107–126. Springer, 2013. *Cited on page:* 8.

[EM05] Tuvi Etzion and Beniamin Mounits. Quasi-perfect codes with small distance. *IEEE Transactions on Information Theory*, 51(11):3938–3946, 2005. *Cited on page:* 65.

[FC16] Marc Fischlin and Jean-Sébastien Coron, editors. *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*. Springer, 2016.

[FIP01] Federal information processing standards publication (FIPS 197). Advanced Encryption Standard (AES), 2001. *Cited on page:* 1.

[Fit14] Robert Fitzpatrick. *Some Algorithms for Learning with Errors*. PhD thesis, Royal Holloway, University of London, 2014. *Cited on page:* 17.

[FMI$^+$06] Marc P. C. Fossorier, Miodrag J. Mihaljevic, Hideki Imai, Yang Cui, and Kanta Matsuura. An Algorithm for Solving the LPN Problem and Its Application to Security Evaluation of the HB Protocols for RFID Authentication. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2006. *Cited on pages:* 13 and 14.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985. *Cited on page:* 1.

[GJL14] Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN Using Covering Codes. In Sarkar and Iwata [SI14], pages 1–20. *Cited on pages:* xix, 13, 14, 15, 22, 24, 25, 26, 27, 28, 46, 47, 50, 55, 56, 57, 58, 62, 64, 65, 74, 76, 77, 97, 140, 141, 146, and 150.

[GJNS14] Jian Guo, Jérémy Jean, Ivica Nikolic, and Yu Sasaki. Meet-in-the-middle attacks on generic feistel constructions. In Sarkar and Iwata [SI14], pages 458–477. *Cited on page:* 35.

[GLRW10] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full tiger, and improved results on MD4 and SHA-2. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 56–75. Springer, 2010. *Cited on page:* 35.

[GR15] Rosario Gennaro and Matthew Robshaw, editors. *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*. Springer, 2015.

[GRS08] Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. HB$^{\#}$: Increasing the Security and Efficiency of HB^{+}. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 361–378. Springer, 2008. *Cited on page:* 8.

[GS85] Ronald L. Graham and Neil J. A. Sloane. On the covering radius of codes. *IEEE Transactions on Information Theory*, 31(3):385–401, 1985. *Cited on pages:* 65 and 66.

[HB01] Nicholas J. Hopper and Manuel Blum. Secure Human Identification Protocols. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001. *Cited on page:* 8.

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980. *Cited on page:* 80.

[Hoe63] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963. *Cited on pages:* 10, 18, and 32.

[HVLN15] Jialin Huang, Serge Vaudenay, Xuejia Lai, and Kaisa Nyberg. Capacity and data complexity in multidimensional linear attack. In Gennaro and Robshaw [GR15], pages 141–160. *Cited on page:* 80.

[IS12] Takanori Isobe and Kyoji Shibutani. All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 202–221. Springer, 2012. *Cited on page:* 35.

[Iso11] Takanori Isobe. A single-key attack on the full GOST block cipher. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop,* *FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011. *Cited on page:* 35.

[JW05] Ari Juels and Stephen A. Weis. Authenticating Pervasive Devices with Human Protocols. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005. *Cited on page:* 8.

[Kea93] Michael J. Kearns. Efficient noise-tolerant learning from statistical queries. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 392–401. ACM, 1993. *Cited on page:* 8.

[KF15] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Gennaro and Robshaw [GR15], pages 43–62. *Cited on page:* 105.

[Kir11] Paul Kirchner. Improved Generalized Birthday Attack. *IACR Cryptology ePrint Archive*, 2011:377, 2011. *Cited on pages:* 24 and 32.

[KMP14] Eike Kiltz, Daniel Masny, and Krzysztof Pietrzak. Simple Chosen-Ciphertext Security from Low-Noise LPN. In Krawczyk [Kra14], pages 1–18. *Cited on page:* 8.

[KPC$^+$11] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient Authentication from Hard Learning Problems. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 7–26. Springer, 2011. *Cited on page:* 8.

[Kra14] Hugo Krawczyk, editor. *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*. Springer, 2014.

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith. Parallel and Concurrent Security of the HB and HB$^+$ Protocols. *J. Cryptology*, 23(3):402–421, 2010. *Cited on page:* 10.

[LBC$^+$12] E. Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariantoni, A. Megrant, P. O'Malley, D. Sank, A. Vainsencher, J. Wenner, T. White, Y. Yin, A. N. Cleland, and J. M. Martinis. Computing prime factors with a Josephson phase qubit quantum processor. *Nature Physics*, 8:719–723, October 2012. *Cited on page:* 2.

[LBYP07] C.-Y. Lu, D. E. Browne, T. Yang, and J.-W. Pan. Demonstration of a Compiled Version of Shor's Quantum Factoring Algorithm Using Photonic Qubits. *Physical Review Letters*, 99(25):250504, December 2007. *Cited on page:* 2.

[LF06] Éric Levieil and Pierre-Alain Fouque. An Improved LPN Algorithm. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2006. *Cited on pages:* 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 40, 43, 47, 56, 76, and 107.

[LM13] Vadim Lyubashevsky and Daniel Masny. Man-in-the-Middle Secure Authentication Schemes from LPN and Weak PRFs. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 308–325. Springer, 2013. *Cited on page:* 8.

[LWL$^+$07] B. P. Lanyon, T. J. Weinhold, N. K. Langford, M. Barbieri, D. F. V. James, A. Gilchrist, and A. G. White. Experimental demonstration of a compiled version of shor's algorithm with quantum entanglement. *Phys. Rev. Lett.*, 99:250505, Dec 2007. *Cited on page:* 2.

[Lyu05] Vadim Lyubashevsky. The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2005. *Cited on pages:* 14, 38, 81, and 100.

[Mas94] J.L. Massey. Guessing and entropy. In *Information Theory, 1994. Proceedings., 1994 IEEE International Symposium on*, pages 204–, Jun 1994. *Cited on page:* 81.

[MLL$^+$12] E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'Brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6:773–776, November 2012. *Cited on page:* 2.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding Random Linear Codes in O(2^{0.054n}). In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011. *Cited on page:* 14.

[MS78] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-holland Publishing Company, 2nd edition, 1978. *Cited on pages:* 11 and 12.

[MS91] Willi Meier and Othmar Staffelbach. Analysis of pseudo random sequence generated by cellular automata. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 1991. *Cited on page:* 80.

[NS05] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 364–372. ACM, 2005. *Cited on page:* 80.

[NS16] Ivica Nikolic and Yu Sasaki. A new algorithm for the unbalanced meet-in-the-middle problem. In Cheon and Takagi [CT16], pages 627–647. *Cited on page:* 35.

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003. *Cited on page:* 80.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342. ACM, 2009. *Cited on pages:* 2 and 8.

[PW72] W.W. Peterson and E.J. Weldon. *Error-correcting Codes.* MIT Press, 1972. *Cited on pages:* 65 and 66.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005. *Cited on pages:* 1, 2, and 8.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. *Cited on page:* 1.

[Sas13] Yu Sasaki. Meet-in-the-middle preimage attacks on AES hashing modes and an application to whirlpool. *IEICE Transactions*, 96-A(1):121–130, 2013. *Cited on page:* 35.

[Sch06] Bruce Schneier. Real-world passwords. `https://www.schneier.com/blog/archives/2006/12/realworld_passw.html`, December 2006. [Online]. *Cited on page:* 102.

[Sel08] Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *J. Cryptology*, 21(1):131–147, 2008. *Cited on page:* 22.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. *Cited on page:* 2.

[SI14] Palash Sarkar and Tetsu Iwata, editors. *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*. Springer, 2014.

[SKS+13] Yu Sasaki, Wataru Komatsubara, Yasuhide Sakai, Lei Wang, Mitsugu Iwamoto, Kazuo Sakiyama, and Kazuo Ohta. Meet-in-the-middle preimage attacks revisited - new results on MD5 and HAVAL. In Pierangela Samarati, editor, *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*, pages 111–122. SciTePress, 2013. *Cited on page:* 35.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988. *Cited on page:* 14.

[SWS$^+$12] Yu Sasaki, Lei Wang, Yasuhide Sakai, Kazuo Sakiyama, and Kazuo Ohta. Three-subset meet-in-the-middle attack on reduced XTEA. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings*, volume 7374 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2012. *Cited on page:* 35.

[VSB$^+$01] Lieven M. K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood, and Isaac L. Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, pages 883–887, 2001. *Cited on page:* 2.

[WACS10] Matt Weir, Sudhir Aggarwal, Michael P. Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 162–175. ACM, 2010. *Cited on page:* 102.

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002. *Cited on pages:* 30 and 31.

[WSK$^+$12] Lei Wang, Yu Sasaki, Wataru Komatsubara, Kazuo Sakiyama, and Kazuo Ohta. Meet-in-the-middle (second) preimage attacks on two double-branch hash functions RIPEMD and RIPEMD-128. *IEICE Transactions*, 95-A(1):100–110, 2012. *Cited on page:* 35.

[XZL$^+$12] N. Xu, J. Zhu, D. Lu, X. Zhou, X. Peng, and J. Du. Quantum Factorization of 143 on a Dipolar-Coupling Nuclear Magnetic Resonance System. *Physical Review Letters*, 108(13):130501, March 2012. *Cited on page:* 2.

[ZJW16] Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster algorithms for solving LPN. In Fischlin and Coron [FC16], pages 168–195. *Cited on pages:* xix, 13, 14, 29, 30, 31, 56, 61, 62, 64, 67, 76, 77, 140, 142, 143, 144, 147, 148, 150, 151, and 152.

Appendix A

Appendix

A.1 LF1 - Full Recovery of the Secret

We provide here an example of the LF1 algorithm, for the $\mathsf{LPN}_{512,0.125}$ instance, where we recover the full secret. We provide the values of a, b, n and time complexity to show that indeed the number of queries for the first iteration, dominates the number of queries needed later on. Also, this shows that the time complexity of recovering the first block dominates the total time complexity. For $\mathsf{LPN}_{512,0.125}$, we obtain the following values (See Table A.1).

The way one can interpret this table is the following: LF1 recovers first 74 bits by taking $a = 7$ and requiring $2^{76.59}$ queries. The total complexity of this step, i.e. the reduction, solving and updating operation, is of $2^{88.57}$ bit operations. Next, LF1 solves $\mathsf{LPN}_{438,0.125}$ and continues this process until it recovers the whole secret.

We can easily see that indeed the number of queries and the time complexity of the first block dominate the other values.

A.2 Heuristic Approximation

In our work, we make the following assumption on the failure probability:

$$E\left(1-(1-\varphi(Z(s)))^{2^{k'}-1}\right) \approx 1-(1-\varphi(E(Z(s))))^{2^{k'}-1},$$

where s is the secret, of k' bits, we recover with the Walsh Hadamard Transform (WHT), $\varphi(x) = \frac{1}{2} + \frac{1}{2}\mathsf{erf}(\frac{x}{\sqrt{2}})$ and erf is the Gauss error function, $Z(s) = -\frac{\delta'(s)}{\sqrt{2-\delta'(s)^2}}\sqrt{n'}$. So, we consider that the average probability of error is very close to the probability of error, i.e. $E(p(s)) \approx p$, where $p = 1-(1-\varphi(Z))^{2^{k'}-1}$. and $Z = -\frac{\delta'}{\sqrt{2-\delta'^2}}\sqrt{n'}$. The values n' and δ' are computed as from our formulae for the biases.

Table A.1: Full secret recovery for the instance $\mathsf{LPN}_{512,0.125}$

i	a	b	$\log_2 n$	$\log_2 t$
1	7	74	76.59	88.57
2	7	64	66.73	78.50
3	7	55	63.21	74.66
4	6	55	57.32	68.45
5	6	46	48.32	59.19
6	6	39	41.36	51.96
7	6	33	36.51	46.83
8	5	33	35.00	44.90
9	5	27	29.01	38.59
10	5	22	24.26	33.53
11	5	18	21.87	30.79
12	4	18	19.63	28.09
13	4	14	16.06	24.08
14	4	11	14.46	22.07
15	3	11	12.42	19.51
16	3	8	10.68	17.12
17	2	8	9.08	15.02
18	2	4	8.00	12.63
19	1	4	6.73	10.57

In order to validate this assumption, we compare our theory with what we observe in practice. For this, we take several chains that contain a code reduction (as the bias introduced by this operation depends on the secret s). The following chains were analysed:

```
k=48, tau=0.005, theta=0.33:
(48,5.8)-sparse-(48,2.9)-xor(1)-(47,3.6)-xor(1)-(46,5.1)-xor(2)-(44,7.2)-code
    -(1,7.2)-WHT
with the codes [44,1,r]

k=48, tau=0.05, theta=0.33:
(48,6.2100)-sparse-(48,4.70)-xor( 1)-(47,7.35)-xor( 1)-(46,12.69)-code
    -(24,12.69)-xor(14)-(10,10.37)-WHT
with the codes [23,12,G][23,12,G]
```

```
k=64, tau=0.05, theta=0.28:
(64,11.30)-sparse-(64,11.3)-code-(10,11.3)-WHT
with the codes [5,1,r][5,1,r][5,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r
    ][7,1,r]

k=64, tau=0.1, theta=0.32:
(64,8.88)-sparse-(64,8.68)- xor( 4)-(60,12.35)-xor(11)-(49,12.70)- xor(11)
    -(38,13.40)-code-(23,13.40)-WHT
with the codes [15,11,H][23,12,G]

k=64, tau=0.1, theta = 0.33:
(64,9.3)-sparse-(64,9.1)-xor(4)-(60,13.2)-xor(12)-(48,13.4)-xor(12)-(36,13.8)-
    code-(1,13.8)-WHT
with the codes [36,1,r]

k=256, tau=0.2, theta=0.12:
(256,41.94)-sparse-(256,41.94)-xor(37)-(219,45.88)-xor(45)-(174,45.76)-xor(45)
    -(129,45.52)-xor(44)-(85,46.04)-code-(45,46.04)-WHT
with the codes [2,1,r][7,4,H][7,4,H][23,12,G][23,12,G][23,12,G]
```

We obtain the following results:
$\mathsf{LPN}_{48,0.05}$ with the codes $[44,1,r]$

- $E(p(s)) = 0.4231$
- $1-(1-\varphi(E(Z(s))))^{2^{k'}-1} = 0.3220$
- $1-(1-\varphi(Z))^{2^{k'}-1} = 0.310893249450781$
- $E(\delta(s)) = 0.0514$
- $\delta = 0.0573009521884174$
- $E(Z(s)) = -0.461981714958069$
- $Z = -0.493326542156718$

$\mathsf{LPN}_{48,0.05}$ with the codes $[23,12,G][23,12,G]$

- $E(p(s)) = 0.4780$
- $1-(1-\varphi(E(Z(s))))^{2^{k'}-1} = 0.0318$
- $1-(1-\varphi(Z))^{2^{k'}-1} = 0.334275183642941$
- $E(\delta(s)) = 0.1298$
- $\delta = 0.129822931628186$
- $E(Z(s)) = -4.00049008907774$

- $Z = -3.35442479599458$

$\mathsf{LPN}_{64,0.05}$ with the codes $[5,1,r][5,1,r][5,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r]$

- $E(p(s)) = 0.8670$
- $1 - (1 - \varphi(E(Z(s))))^{2^{k'}-1} = 0.074$
- $1 - (1 - \varphi(Z))^{2^{k'}-1} = 0.282948298443234$
- $E(\delta(s)) = 0.10654$
- $\delta = 0.09580$
- $E(Z(s)) = -3.79016746911067$
- $Z = -3.40978448268245$

$\mathsf{LPN}_{64,0.1}$ with the codes $[15,11,H][23,12,G]$

- $E(p(s)) = 0.5353$
- $1 - (1 - \varphi(E(Z(s))))^{2^{k'}-1} = 0.3274$
- $1 - (1 - \varphi(Z))^{2^{k'}-1} = 0.335578150280515$
- $E(\delta(s)) = 0.0724$
- $\delta = 0.0724220723200000$
- $E(Z(s)) = -5.33685553225594$
- $Z = -5.33137227108397$

$\mathsf{LPN}_{64,0.1}$ with the codes $[36,1,r]$

- $E(p(s)) = 0.4461$
- $1 - (1 - \varphi(E(Z(s))))^{2^{k'}-1} = 0.3159$
- $1 - (1 - \varphi(Z))^{2^{k'}-1} = 0.316793578967859$
- $E(\delta(s)) = 0.0056$
- $\delta = 0.00564445020331737$
- $E(Z(s)) = -0.478971239492193$
- $Z = -0.476684216638090$

$\mathsf{LPN}_{256,0.2}$ with the codes $[2,1,r][7,4,H][7,4,H][23,12,G][23,12,G][23,12,G]$

- $E(p(s)) = 0.7609$
- $1-(1-\varphi(E(Z(s))))^{2^{k'}-1} = 0.1209$
- $1-(1-\varphi(Z))^{2^{k'}-1} = 0.120943626782345$
- $E(\delta(s)) = 1.2933e-6$
- $\delta = 1.2933e-6$
- $E(Z(s)) = -7.77878914531831$
- $Z = -7.77878914490038$

From this, we conclude that the approximations $E(\delta(s)) \approx \delta$ and $E(Z(s)) \approx Z$ are correct but that there is a small gap between $E(p(s))$ and p. The function $x \mapsto 1-(1-\varphi(x))^{2^{k'}-1}$ is convex in the region $x \leq 0$ so we rather have $E(p(s)) \geq 1-(1-\varphi(E(Z(s))))^{2^{k'}-1}$.

A.3 Covering Code with a Bias of 0

For a *code-reduce* we have that $\mathsf{bc} = E((-1)^{\langle v_i - g_i, s\rangle})$.

For a repetition code $[k,1,D]$ we have $D=k$ and $R = \lfloor \frac{k-1}{2} \rfloor$. Thus, $\mathsf{Hw}(v_i - g_i) \leq R$. When k is odd and the $\mathsf{Hw}(s)$ is even we have $(-1)^{\langle x,s\rangle} = (-1)^{\langle \bar{x},s\rangle}$ for any $x \in \{0,1\}^k$ where $\bar{x}$ is the complement of x.

Thus,

$$\begin{aligned}
E_{v_i}\left((-1)^{\langle v_i-g_i,s\rangle}\right) &= E_x\left((-1)^{\langle x,s\rangle} \mid \mathsf{HW}(x) \leq R\right) \\
&= E_y\left((-1)^{\langle y,s\rangle} \mid \mathsf{HW}(x) > R\right)
\end{aligned}$$

The second equality holds because $R < \frac{k}{2}$ and because of the aforementioned property.

We also know that $E_x((-1)^{\langle x,s\rangle}) = 0$ for $s \neq 0$ as the bits of x are independent. We deduce that for $s \neq 0$

$$\begin{aligned}
E_x((-1)^{\langle x,s\rangle}) &= E_x\left((-1)^{\langle x,s\rangle} | \mathsf{HW}(x) \leq R\right) \cdot \Pr[\mathsf{HW}(x) \leq R] + \\
&+ E_x\left((-1)^{\langle x,s\rangle} | \mathsf{HW}(x) > R\right) \cdot \Pr[\mathsf{HW}(x) > R] \\
&= E_x\left((-1)^{\langle x,s\rangle} | \mathsf{HW}(x) \leq R\right) \cdot (\Pr[\mathsf{HW}(x) \leq R] + \Pr[\mathsf{HW}(x) > R]) \\
&= E_x\left((-1)^{\langle x,s\rangle} | \mathsf{HW}(x) \leq R\right) = \mathsf{bc} \\
&= 0
\end{aligned}$$

Thus, $\mathsf{bc} = 0$ for a repetition code $[k,1,D]$ where k is odd and the Hamming weight of s is even and $s \neq 0$.

A.4 Random Codes

We present below a list of random codes and their bc. The way we generate these codes is the following: for a $[k, k']$ code, we generate, step by step, random vectors of size k until we have k' independent ones. This will form the generator matrix G. Once we generate all the $2^{k'}$ codewords, we use our formula to compute the bc. After many trials, we keep the code with the largest value for bc.

Random Codes for $\tau = 0.05$:

$[4,2]$ bc $= 0.925000000000000$	$[18,3]$ bc $= 0.542640210536539$	$[13,6]$ bc $= 0.785820312500064$
$[7,2]$ bc $= 0.820437500000001$	$[19,3]$ bc $= 0.523425509870706$	$[15,6]$ bc $= 0.734159374999888$
$[9,2]$ bc $= 0.755670312499999$	$[10,4]$ bc $= 0.799187499999992$	$[16,6]$ bc $= 0.708464716796699$
$[10,2]$ bc $= 0.717886796875003$	$[12,4]$ bc $= 0.747566406249973$	$[17,6]$ bc $= 0.682572705077685$
$[11,2]$ bc $= 0.689080273437506$	$[13,4]$ bc $= 0.717848281250054$	$[18,6]$ bc $= 0.656482291502392$
$[12,2]$ bc $= 0.658278378906265$	$[14,4]$ bc $= 0.686005234375007$	$[19,6]$ bc $= 0.633382513669716$
$[13,2]$ bc $= 0.627343367480487$	$[15,4]$ bc $= 0.659313910156184$	$[9,7]$ bc $= 0.924999999999994$
$[14,2]$ bc $= 0.605104670898428$	$[16,4]$ bc $= 0.635095716308526$	$[10,7]$ bc $= 0.901249999999986$
$[15,2]$ bc $= 0.574849437353450$	$[17,4]$ bc $= 0.609055215502953$	$[14,7]$ bc $= 0.789750781249873$
$[16,2]$ bc $= 0.550932480072046$	$[18,4]$ bc $= 0.583034990081217$	$[16,7]$ bc $= 0.741090429687176$
$[17,2]$ bc $= 0.522348102187504$	$[19,4]$ bc $= 0.560423084937082$	$[17,7]$ bc $= 0.714251874999825$
$[18,2]$ bc $= 0.503960971625559$	$[8,5]$ bc $= 0.912500000000003$	$[18,7]$ bc $= 0.687121879880629$
$[19,2]$ bc $= 0.479500628284949$	$[11,5]$ bc $= 0.810718749999989$	$[19,7]$ bc $= 0.664058937986979$
$[5,3]$ bc $= 0.925000000000000$	$[12,5]$ bc $= 0.780814062499967$	$[10,8]$ bc $= 0.924999999999989$
$[8,3]$ bc $= 0.830843750000002$	$[13,5]$ bc $= 0.756492578125057$	$[11,8]$ bc $= 0.912500000000028$
$[9,3]$ bc $= 0.795250000000000$	$[14,5]$ bc $= 0.723323867187460$	$[15,8]$ bc $= 0.797351562500136$
$[10,3]$ bc $= 0.764220312499999$	$[15,5]$ bc $= 0.696788447265571$	$[17,8]$ bc $= 0.746043945312908$
$[11,3]$ bc $= 0.734533281250005$	$[16,5]$ bc $= 0.670566863281211$	$[18,8]$ bc $= 0.720697158201192$
$[12,3]$ bc $= 0.707727851562494$	$[17,5]$ bc $= 0.642065001220673$	$[19,8]$ bc $= 0.696926562499299$
$[13,3]$ bc $= 0.677121250000042$	$[18,5]$ bc $= 0.621900535155624$	$[11,9]$ bc $= 0.925000000000025$
$[14,3]$ bc $= 0.644854535644540$	$[19,5]$ bc $= 0.596215895090632$	$[12,9]$ bc $= 0.912500000000054$
$[15,3]$ bc $= 0.622457954101448$	$[8,6]$ bc $= 0.925000000000003$	$[16,9]$ bc $= 0.803749999999664$
$[16,3]$ bc $= 0.592374557714663$	$[9,6]$ bc $= 0.912499999999993$	$[18,9]$ bc $= 0.754477929686187$
$[17,3]$ bc $= 0.570423946075392$	$[12,6]$ bc $= 0.813390624999974$	

Random Codes for $\tau = 0.1$:

$[4,2]$ bc $= 0.850000000000000$	$[14,3]$ bc $= 0.406910531250004$	$[15,5]$ bc $= 0.477960312500255$
$[7,2]$ bc $= 0.650250000000000$	$[15,3]$ bc $= 0.374183078125198$	$[16,5]$ bc $= 0.432172781249713$
$[9,2]$ bc $= 0.550462500000000$	$[16,3]$ bc $= 0.338270570312456$	$[17,5]$ bc $= 0.404812453125008$
$[10,2]$ bc $= 0.501896250000001$	$[17,3]$ bc $= 0.305143173281109$	$[18,5]$ bc $= 0.370472807812691$
$[11,2]$ bc $= 0.451706624999987$	$[18,3]$ bc $= 0.282576971953005$	$[19,5]$ bc $= 0.342473907030773$
$[12,2]$ bc $= 0.414441562499999$	$[19,3]$ bc $= 0.257462926757663$	$[8,6]$ bc $= 0.850000000000003$
$[13,2]$ bc $= 0.384841406249986$	$[10,4]$ bc $= 0.620124999999991$	$[9,6]$ bc $= 0.785000000000002$
$[14,2]$ bc $= 0.347599265625008$	$[12,4]$ bc $= 0.533606249999987$	$[12,6]$ bc $= 0.657625000000007$
$[15,2]$ bc $= 0.315965939062601$	$[13,4]$ bc $= 0.498295625000009$	$[13,6]$ bc $= 0.604262500000019$
$[16,2]$ bc $= 0.292209785156346$	$[14,4]$ bc $= 0.456234062499976$	$[15,6]$ bc $= 0.523140625000133$
$[17,2]$ bc $= 0.255932410640333$	$[15,4]$ bc $= 0.423086156250238$	$[16,6]$ bc $= 0.483241562499645$
$[18,2]$ bc $= 0.235804919976357$	$[16,4]$ bc $= 0.388249140624795$	$[17,6]$ bc $= 0.449978906249989$
$[19,2]$ bc $= 0.217948973378862$	$[17,4]$ bc $= 0.357040026562527$	$[18,6]$ bc $= 0.415639015625284$
$[5,3]$ bc $= 0.850000000000000$	$[18,4]$ bc $= 0.326219003906170$	$[19,6]$ bc $= 0.385346414062386$
$[8,3]$ bc $= 0.662249999999999$	$[19,4]$ bc $= 0.298997873515392$	$[9,7]$ bc $= 0.850000000000006$
$[9,3]$ bc $= 0.620124999999997$	$[8,5]$ bc $= 0.805000000000002$	$[10,7]$ bc $= 0.804999999999990$
$[10,3]$ bc $= 0.570712499999995$	$[11,5]$ bc $= 0.647124999999997$	$[14,7]$ bc $= 0.610312500000011$
$[11,3]$ bc $= 0.523481249999980$	$[12,5]$ bc $= 0.602012500000006$	$[16,7]$ bc $= 0.535203124999684$
$[12,3]$ bc $= 0.483963124999979$	$[13,5]$ bc $= 0.557631250000012$	$[17,7]$ bc $= 0.492602812500183$
$[13,3]$ bc $= 0.439562812499995$	$[14,5]$ bc $= 0.514058124999978$	$[18,7]$ bc $= 0.459242031250249$

$[19,7]$ bc $= 0.427382828126207$
$[10,8]$ bc $= 0.849999999999990$
$[11,8]$ bc $= 0.824999999999974$
$[15,8]$ bc $= 0.624562499999774$
$[17,8]$ bc $= 0.543265625000525$
$[18,8]$ bc $= 0.505529062500634$
$[19,8]$ bc $= 0.466630156251362$
$[11,9]$ bc $= 0.849999999999978$
$[12,9]$ bc $= 0.825000000000046$
$[16,9]$ bc $= 0.625812499999804$

Random Codes for $\tau = 0.125$:

$[4,2]$ bc $= 0.812500000000000$
$[7,2]$ bc $= 0.590820312500000$
$[9,2]$ bc $= 0.469238281250000$
$[10,2]$ bc $= 0.410583496093750$
$[11,2]$ bc $= 0.363045692443848$
$[12,2]$ bc $= 0.331746578216553$
$[13,2]$ bc $= 0.295313835144043$
$[14,2]$ bc $= 0.262624084949493$
$[15,2]$ bc $= 0.229796074330807$
$[16,2]$ bc $= 0.199777818284929$
$[17,2]$ bc $= 0.178411910077557$
$[18,2]$ bc $= 0.161525560077280$
$[19,2]$ bc $= 0.142899490048876$
$[5,3]$ bc $= 0.812500000000000$
$[8,3]$ bc $= 0.596679687500000$
$[9,3]$ bc $= 0.538574218750000$
$[10,3]$ bc $= 0.484985351562500$
$[11,3]$ bc $= 0.438781738281250$
$[12,3]$ bc $= 0.395263671875000$
$[13,3]$ bc $= 0.344362258911133$
$[14,3]$ bc $= 0.318330764770508$
$[15,3]$ bc $= 0.280164837837219$
$[16,3]$ bc $= 0.250766009092331$
$[17,3]$ bc $= 0.222185984253883$
$[18,3]$ bc $= 0.201188247650862$
$[19,3]$ bc $= 0.173051953781396$
$[10,4]$ bc $= 0.556884765625000$
$[12,4]$ bc $= 0.459289550781250$
$[13,4]$ bc $= 0.416761398315430$
$[14,4]$ bc $= 0.364337921142578$
$[15,4]$ bc $= 0.324589252471924$
$[16,4]$ bc $= 0.295842528343201$
$[17,4]$ bc $= 0.266516566276550$
$[18,4]$ bc $= 0.238161936402321$
$[19,4]$ bc $= 0.215231126174331$
$[8,5]$ bc $= 0.734375000000000$
$[11,5]$ bc $= 0.568603515625000$
$[12,5]$ bc $= 0.515930175781250$
$[13,5]$ bc $= 0.467575073242188$
$[14,5]$ bc $= 0.427307128906250$
$[15,5]$ bc $= 0.384399414062500$
$[16,5]$ bc $= 0.345722198486328$
$[17,5]$ bc $= 0.308802604675293$
$[18,5]$ bc $= 0.278767108917236$
$[19,5]$ bc $= 0.249812759459019$
$[8,6]$ bc $= 0.812500000000000$
$[9,6]$ bc $= 0.757812500000000$
$[12,6]$ bc $= 0.584716796875000$
$[13,6]$ bc $= 0.533325195312500$
$[15,6]$ bc $= 0.436737060546875$
$[16,6]$ bc $= 0.391594886779785$
$[17,6]$ bc $= 0.354474067687988$
$[18,6]$ bc $= 0.323782920837402$
$[19,6]$ bc $= 0.291754990816116$
$[9,7]$ bc $= 0.812500000000000$
$[10,7]$ bc $= 0.757812500000000$
$[14,7]$ bc $= 0.539184570312500$
$[16,7]$ bc $= 0.451797485351562$
$[17,7]$ bc $= 0.404412269592285$
$[18,7]$ bc $= 0.368431568145752$
$[19,7]$ bc $= 0.336303114891052$
$[10,8]$ bc $= 0.812500000000000$
$[11,8]$ bc $= 0.781250000000000$
$[15,8]$ bc $= 0.542846679687500$
$[17,8]$ bc $= 0.452558517456055$
$[18,8]$ bc $= 0.415075302124023$
$[19,8]$ bc $= 0.374835968017578$
$[11,9]$ bc $= 0.812500000000000$
$[12,9]$ bc $= 0.757812500000000$
$[16,9]$ bc $= 0.547607421875000$
$[18,9]$ bc $= 0.462966918945312$

Random Codes for $\tau = 0.2$:

$[4,2]$ bc $= 0.700000000000000$
$[7,2]$ bc $= 0.405500000000000$
$[9,2]$ bc $= 0.272749999999999$
$[10,2]$ bc $= 0.221980000000003$
$[11,2]$ bc $= 0.177584000000003$
$[12,2]$ bc $= 0.154187500000002$
$[13,2]$ bc $= 0.126058437500001$
$[14,2]$ bc $= 0.102578124999976$
$[15,2]$ bc $= 0.0806773999999974$
$[16,2]$ bc $= 0.0672851875000390$
$[17,2]$ bc $= 0.0541784750000697$
$[18,2]$ bc $= 0.0430625200001377$
$[19,2]$ bc $= 0.0349497860001292$
$[5,3]$ bc $= 0.700000000000000$
$[8,3]$ bc $= 0.405499999999999$
$[9,3]$ bc $= 0.343000000000001$
$[10,3]$ bc $= 0.291275000000004$
$[11,3]$ bc $= 0.240512500000004$
$[12,3]$ bc $= 0.197742500000001$
$[13,3]$ bc $= 0.156951999999985$
$[14,3]$ bc $= 0.137524624999980$
$[15,3]$ bc $= 0.111005200000025$
$[16,3]$ bc $= 0.0928898000000887$
$[17,3]$ bc $= 0.0767771256251488$
$[18,3]$ bc $= 0.0609095375002112$
$[19,3]$ bc $= 0.0505532889999707$
$[10,4]$ bc $= 0.364750000000007$
$[12,4]$ bc $= 0.258325000000002$
$[13,4]$ bc $= 0.213679999999971$
$[14,4]$ bc $= 0.176458750000002$
$[15,4]$ bc $= 0.147204875000059$
$[16,4]$ bc $= 0.121787406250114$
$[17,4]$ bc $= 0.102391512500167$
$[18,4]$ bc $= 0.0842524562500867$
$[19,4]$ bc $= 0.0690329674999138$
$[8,5]$ bc $= 0.589999999999999$
$[11,5]$ bc $= 0.369250000000010$
$[12,5]$ bc $= 0.318649999999986$
$[13,5]$ bc $= 0.268149999999959$
$[14,5]$ bc $= 0.225544999999992$
$[15,5]$ bc $= 0.188910625000082$
$[16,5]$ bc $= 0.157532562500143$
$[17,5]$ bc $= 0.130860062500169$
$[18,5]$ bc $= 0.109656978125099$
$[19,5]$ bc $= 0.0912561109374092$
$[8,6]$ bc $= 0.699999999999998$
$[9,6]$ bc $= 0.650000000000002$
$[12,6]$ bc $= 0.379749999999974$
$[13,6]$ bc $= 0.328624999999967$
$[15,6]$ bc $= 0.235025000000079$
$[16,6]$ bc $= 0.197588125000182$
$[17,6]$ bc $= 0.169437500000210$
$[18,6]$ bc $= 0.139357843750180$
$[19,6]$ bc $= 0.119218062500138$
$[9,7]$ bc $= 0.700000000000003$
$[10,7]$ bc $= 0.620000000000010$
$[14,7]$ bc $= 0.343999999999986$
$[16,7]$ bc $= 0.245225000000221$
$[17,7]$ bc $= 0.207013750000245$
$[18,7]$ bc $= 0.175233125000239$
$[19,7]$ bc $= 0.151714656250229$
$[10,8]$ bc $= 0.700000000000010$
$[11,8]$ bc $= 0.619999999999996$
$[15,8]$ bc $= 0.356000000000120$
$[17,8]$ bc $= 0.255875000000282$
$[18,8]$ bc $= 0.214251250000277$
$[19,8]$ bc $= 0.185132187500286$
$[11,9]$ bc $= 0.699999999999990$
$[12,9]$ bc $= 0.649999999999956$
$[16,9]$ bc $= 0.366875000000244$

Random Codes for $\tau = 0.25$:

$[4, 2]$ bc $= 0.625000000000000$	$[18, 3]$ bc $= 0.0254254341125488$	$[13, 6]$ bc $= 0.230468750000000$
$[7, 2]$ bc $= 0.292968750000000$	$[19, 3]$ bc $= 0.0197929143905640$	$[15, 6]$ bc $= 0.147705078125000$
$[9, 2]$ bc $= 0.184570312500000$	$[10, 4]$ bc $= 0.263671875000000$	$[16, 6]$ bc $= 0.118072509765625$
$[10, 2]$ bc $= 0.143066406250000$	$[12, 4]$ bc $= 0.168457031250000$	$[17, 6]$ bc $= 0.0952911376953125$
$[11, 2]$ bc $= 0.107299804687500$	$[13, 4]$ bc $= 0.129882812500000$	$[18, 6]$ bc $= 0.0742111206054688$
$[12, 2]$ bc $= 0.0848999023437500$	$[14, 4]$ bc $= 0.101013183593750$	$[19, 6]$ bc $= 0.0585670471191406$
$[13, 2]$ bc $= 0.0620727539062500$	$[15, 4]$ bc $= 0.0771789550781250$	$[9, 7]$ bc $= 0.625000000000000$
$[14, 2]$ bc $= 0.0496444702148438$	$[16, 4]$ bc $= 0.0606994628906250$	$[10, 7]$ bc $= 0.531250000000000$
$[15, 2]$ bc $= 0.0366325378417969$	$[17, 4]$ bc $= 0.0467262268066406$	$[14, 7]$ bc $= 0.240234375000000$
$[16, 2]$ bc $= 0.0274744033813477$	$[18, 4]$ bc $= 0.0371999740600586$	$[16, 7]$ bc $= 0.154541015625000$
$[17, 2]$ bc $= 0.0209437608718872$	$[19, 4]$ bc $= 0.0295500755310059$	$[17, 7]$ bc $= 0.119049072265625$
$[18, 2]$ bc $= 0.0164231657981873$	$[8, 5]$ bc $= 0.531250000000000$	$[18, 7]$ bc $= 0.0965576171875000$
$[19, 2]$ bc $= 0.0120856314897537$	$[11, 5]$ bc $= 0.275390625000000$	$[19, 7]$ bc $= 0.0788497924804688$
$[5, 3]$ bc $= 0.625000000000000$	$[12, 5]$ bc $= 0.218750000000000$	$[10, 8]$ bc $= 0.625000000000000$
$[8, 3]$ bc $= 0.320312500000000$	$[13, 5]$ bc $= 0.170166015625000$	$[11, 8]$ bc $= 0.562500000000000$
$[9, 3]$ bc $= 0.250000000000000$	$[14, 5]$ bc $= 0.138671875000000$	$[15, 8]$ bc $= 0.251953125000000$
$[10, 3]$ bc $= 0.196289062500000$	$[15, 5]$ bc $= 0.111480712890625$	$[17, 8]$ bc $= 0.162719726562500$
$[11, 3]$ bc $= 0.150146484375000$	$[16, 5]$ bc $= 0.0867004394531250$	$[18, 8]$ bc $= 0.134338378906250$
$[12, 3]$ bc $= 0.121704101562500$	$[17, 5]$ bc $= 0.0682830810546875$	$[19, 8]$ bc $= 0.106536865234375$
$[13, 3]$ bc $= 0.0912780761718750$	$[18, 5]$ bc $= 0.0538368225097656$	$[11, 9]$ bc $= 0.625000000000000$
$[14, 3]$ bc $= 0.0709533691406250$	$[19, 5]$ bc $= 0.0424060821533203$	$[12, 9]$ bc $= 0.531250000000000$
$[15, 3]$ bc $= 0.0566749572753906$	$[8, 6]$ bc $= 0.625000000000000$	$[16, 9]$ bc $= 0.255859375000000$
$[16, 3]$ bc $= 0.0420742034912109$	$[9, 6]$ bc $= 0.500000000000000$	$[18, 9]$ bc $= 0.171264648437500$
$[17, 3]$ bc $= 0.0335798263549805$	$[12, 6]$ bc $= 0.292968750000000$	

The random codes used for $\mathsf{LPN}_{512,0.125}$ have the following generator matrices:

```
[18,6]
G is:
(0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1)
(0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0)
(0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0)
(1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0)
(0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1)
(0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1)



[19,7]
G is:
(0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0)
(1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0)
(0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0)
(0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1)
(0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1)
(0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1)

[19,6]
G is:
(0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)
(0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1)
(1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1)
```

```
(1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0)
(1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0)
(1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0)
```

A.5 Solving LPN without *guess-secret*

We present below the chains without *guess-secret*. The codes used in the *code-reduce* correspond to the perfect and quasi-perfect codes presented in the paper, to which we add random codes that we generated. The list of random codes can be found in this document as well.

```
% below, lsumc and lmaxc are the logarithmic total and max- complexities

% what follows uses precision .1, without guess

k=32 tau=0.050 theta=0.33 precision=0.10 lmaxc=11.26 lsumc=13.89:
  (32,11.2)-drop(1)-(31,10.2)-drop(4)-(27,6.2)-xor(5)-(22,6.4)-xor(5)-(17,6.8)-
      xor(6)-(11,6.6)-xor(5)-(6,7.2)-xor(5)-(1,8.4)-WHT
k=32 tau=0.100 theta=0.33 precision=0.10 lmaxc=12.70 lsumc=15.04:
  (32,12.7)-drop(1)-(31,11.7)-drop(4)-(27,7.7)-xor(7)-(20,7.4)-xor(6)-(14,7.8)-
      xor(6)-(8,8.6)-xor(7)-(1,9.2)-WHT
k=32 tau=0.125 theta=0.33 precision=0.10 lmaxc=13.52 lsumc=15.66:
  (32,13.3)-drop(1)-(31,12.3)-drop(4)-(27,8.3)-xor(7)-(20,8.6)-xor(7)-(13,9.2)-
      xor(8)-(5,9.4)-drop(4)-(1,5.4)-WHT
k=32 tau=0.200 theta=0.33 precision=0.10 lmaxc=14.80 lsumc=17.01:
  (32,14.8)-drop(1)-(31,13.8)-drop(4)-(27,9.8)-xor(9)-(18,9.6)-xor(8)-(10,10.2)
      -xor(8)-(2,11.4)-drop(1)-(1,10.4)-WHT
k=32 tau=0.250 theta=0.33 precision=0.10 lmaxc=16.30 lsumc=18.42:
  (32,16.3)-drop(1)-(31,15.3)-drop(4)-(27,11.3)-xor(11)-(16,10.6)-xor(8)
      -(8,12.2)-WHT

k=48 tau=0.050 theta=0.33 precision=0.10 lmaxc=12.94 lsumc=14.52:
  (48,5.8)-sparse-(48,2.9)-xor(1)-(47,3.6)-xor(1)-(46,5.1)-xor(2)-(44,7.2)-code
      ([44,1,r])-(1,7.2)-WHT
k=48 tau=0.100 theta=0.33 precision=0.10 lmaxc=16.43 lsumc=18.58:
  (48,16)-drop(1)-(47,15)-drop(4)-(43,11)-xor(10)-(33,11)-xor(10)-(23,11)-xor
      (10)-(13,11)-xor(9)-(4,12)-drop(3)-(1,9)-WHT
k=48 tau=0.125 theta=0.33 precision=0.10 lmaxc=17.00 lsumc=19.29:
  (48,16.5)-drop(1)-(47,15.5)-drop(4)-(43,11.5)-xor(11)-(32,11)-xor(9)-(23,12)-
      xor(11)-(12,12)-xor(10)-(2,13)-drop(1)-(1,12)-WHT
k=48 tau=0.200 theta=0.33 precision=0.10 lmaxc=19.23 lsumc=21.25:
  (48,18.8)-drop(1)-(47,17.8)-drop(4)-(43,13.8)-xor(13)-(30,13.6)-xor(12)
      -(18,14.2)-xor(13)-(5,14.4)-drop(4)-(1,10.4)-WHT
k=48 tau=0.250 theta=0.33 precision=0.10 lmaxc=20.43 lsumc=22.34:
  (48,20)-drop(1)-(47,19)-drop(4)-(43,15)-xor(14)-(29,15)-xor(14)-(15,15)-drop
      (2)-(13,13)-WHT
```

```
k=64 tau=0.050 theta=0.33 precision=0.10 lmaxc=14.43 lsumc=16.04:
  (64,6.3)-sparse-(64,3.9)-xor(1)-(63,5.7)-xor(2)-(61,8.4)-code([61,1,r])
      -(1,8.4)-WHT
k=64 tau=0.100 theta=0.33 precision=0.10 lmaxc=19.38 lsumc=21.58:
  (64,9.3)-sparse-(64,9.1)-xor(4)-(60,13.2)-xor(12)-(48,13.4)-xor(12)-(36,13.8)
      -code([36,1,r])-(1,13.8)-WHT
k=64 tau=0.125 theta=0.33 precision=0.10 lmaxc=20.50 lsumc=22.94:
  (64,20.5)-drop(1)-(63,19.5)-drop(5)-(58,14.5)-xor(14)-(44,14)-xor(12)-(32,15)
      -xor(14)-(18,15)-xor(13)-(5,16)-drop(4)-(1,12)-WHT
k=64 tau=0.200 theta=0.33 precision=0.10 lmaxc=22.00 lsumc=24.42:
  (64,22)-drop(1)-(63,21)-drop(5)-(58,16)-xor(15)-(43,16)-xor(15)-(28,16)-xor
      (14)-(14,17)-WHT
k=64 tau=0.250 theta=0.33 precision=0.10 lmaxc=24.58 lsumc=26.86:
  (64,24.5)-drop(1)-(63,23.5)-drop(5)-(58,18.5)-xor(18)-(40,18)-xor(16)-(24,19)
      -xor(17)-(7,20)-WHT

k=100 tau=0.050 theta=0.33 precision=0.10 lmaxc=18.46 lsumc=20.47:
  (100,7.9)-sparse-(100,7.1)-xor(2)-(98,11.2)-xor(10)-(88,11.4)-xor(10)
      -(78,11.8)-code([78,1,r])-(1,11.8)-WHT
k=100 tau=0.100 theta=0.33 precision=0.10 lmaxc=25.39 lsumc=27.61:
  (100,14.9)-sparse-(100,14.9)-xor(11)-(89,17.8)-xor(16)-(73,18.6)-xor(17)
      -(56,19.2)-code([18,5,rnd150927][19,6,rnd150927][19,6,rnd150927])
      -(17,19.2)-WHT
k=100 tau=0.125 theta=0.33 precision=0.10 lmaxc=26.30 lsumc=28.91:
  (100,26.3)-drop(1)-(99,25.3)-drop(6)-(93,19.3)-xor(18)-(75,19.6)-xor(19)
      -(56,19.2)-xor(17)-(39,20.4)-xor(19)-(20,20.8)-drop(2)-(18,18.8)-WHT
k=100 tau=0.200 theta=0.33 precision=0.10 lmaxc=29.75 lsumc=32.06:
  (100,29.1)-drop(1)-(99,28.1)-drop(5)-(94,23.1)-xor(22)-(72,23.2)-xor(22)
      -(50,23.4)-xor(22)-(28,23.8)-drop(6)-(22,17.8)-WHT
k=100 tau=0.250 theta=0.33 precision=0.10 lmaxc=30.75 lsumc=32.94:
  (100,30.1)-drop(1)-(99,29.1)-drop(5)-(94,24.1)-xor(23)-(71,24.2)-xor(23)
      -(48,24.4)-xor(23)-(25,24.8)-drop(3)-(22,21.8)-WHT

k=256 tau=0.050 theta=0.33 precision=0.10 lmaxc=34.45 lsumc=36.75:
  (256,21.8)-sparse-(256,21.8)-xor(17)-(239,25.6)-xor(24)-(215,26.2)-xor(25)
      -(190,26.4)-xor(25)-(165,26.8)-code([165,1,r])-(1,26.8)-WHT
k=256 tau=0.100 theta=0.33 precision=0.10 lmaxc=44.22 lsumc=46.75:
  (256,32.3)-sparse-(256,32.3)-xor(28)-(228,35.6)-xor(34)-(194,36.2)-xor(35)
      -(159,36.4)-xor(35)-(124,36.8)-code([11,4,S][18,5,rnd150927][19,5,
      rnd150927][19,5,rnd150927][19,5,rnd150927][19,5,rnd150927][19,5,rnd150927
      ])-(34,36.8)-WHT
k=256 tau=0.125 theta=0.33 precision=0.10 lmaxc=47.35 lsumc=49.90:
  (256,35.6)-sparse-(256,35.6)-xor(31)-(225,39.2)-xor(38)-(187,39.4)-xor(38)
      -(149,39.8)-xor(39)-(110,39.6)-code([15,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,7,rnd150926][19,7,rnd150926])-(37,39.6)-WHT
k=256 tau=0.200 theta=0.33 precision=0.10 lmaxc=53.82 lsumc=56.31:
  (256,42.4)-sparse-(256,42.4)-xor(38)-(218,45.8)-xor(45)-(173,45.6)-xor(44)
      -(129,46.2)-xor(45)-(84,46.4)-code([15,7,2BCH][25,15,W][25,15,W][19,7,
      rnd150927])-(44,46.4)-drop(1)-(43,45.4)-WHT
k=256 tau=0.250 theta=0.33 precision=0.10 lmaxc=56.88 lsumc=59.47:
  (256,56.7)-drop(1)-(255,55.7)-drop(7)-(248,48.7)-xor(48)-(200,48.4)-xor(47)
      -(153,48.8)-xor(47)-(106,49.6)-xor(59)-(47,39.2)-WHT

k=512 tau=0.050 theta=0.33 precision=0.10 lmaxc=55.09 lsumc=57.77:
  (512,41.3)-sparse-(512,41.3)-xor(36)-(476,45.6)-xor(44)-(432,46.2)-xor(46)
      -(386,45.4)-xor(44)-(342,45.8)-xor(44)-(298,46.6)-code([9,1,r][9,1,r
      ][11,1,r][11,1,r][11,1,r][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927
      ][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927][19,3,
      rnd150927][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927
      ][19,3,rnd150927])-(44,46.6)-WHT
k=512 tau=0.100 theta=0.33 precision=0.10 lmaxc=70.92 lsumc=73.68:
  (512,57.4)-sparse-(512,57.4)-xor(52)-(460,61.8)-xor(61)-(399,61.6)-xor(60)
      -(339,62.2)-xor(61)-(278,62.4)-xor(61)-(217,62.8)-code([8,2,iGop][19,5,
      rnd150927][19,5,rnd150927][19,5,rnd150927][19,6,rnd150927][19,6,rnd150927
      ][19,5,rnd150927][19,5,rnd150927][19,5,rnd150927][19,5,rnd150927][19,5,
      rnd150927][19,5,rnd150927])-(59,62.8)-WHT
k=512 tau=0.125 theta=0.33 precision=0.10 lmaxc=76.22 lsumc=78.85:
  (512,63.3)-sparse-(512,63.3)-xor(59)-(453,66.6)-xor(65)-(388,67.2)-xor(66)
      -(322,67.4)-xor(66)-(256,67.8)-xor(67)-(189,67.6)-code([18,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,7,rnd150926][19,7,rnd150926][19,7,rnd150926][19,7,rnd150926
      ])-(64,67.6)-WHT
k=512 tau=0.200 theta=0.33 precision=0.10 lmaxc=86.38 lsumc=89.04:
  (512,73.6)-sparse-(512,73.6)-xor(69)-(443,77.2)-xor(76)-(367,77.4)-xor(76)
      -(291,77.8)-xor(77)-(214,77.6)-xor(76)-(138,78.2)-code([23,12,G][25,15,W
      ][25,15,W][25,15,W][25,15,W][15,4,rnd150927])-(76,78.2)-drop(2)-(74,76.2)-
      WHT
k=512 tau=0.250 theta=0.33 precision=0.10 lmaxc=91.97 lsumc=94.66:
  (512,79.3)-sparse-(512,79.3)-xor(75)-(437,82.6)-xor(81)-(356,83.2)-xor(82)
      -(274,83.4)-xor(82)-(192,83.8)-xor(83)-(109,83.6)-code([31,26,H][16,15,
      Chop][31,20,W][31,20,W])-(81,83.6)-drop(1)-(80,82.6)-WHT

k=768 tau=0.050 theta=0.33 precision=0.10 lmaxc=74.03 lsumc=76.63:
  (768,60)-sparse-(768,60)-xor(55)-(713,64)-xor(63)-(650,64)-xor(63)-(587,64)-
      xor(63)-(524,64)-xor(62)-(462,65)-code([11,1,r][11,1,r][19,3,rnd150927
      ][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927][19,3,
      rnd150927][19,3,rnd150927][13,1,r][13,1,r][13,1,r][13,1,r][13,1,r][13,1,r
      ][13,1,r][13,1,r][13,1,r][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927
      ][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927][19,3,rnd150927][19,3,
      rnd150927][19,3,rnd150927][19,3,rnd150927])-(62,65)-WHT
k=768 tau=0.100 theta=0.33 precision=0.10 lmaxc=96.04 lsumc=98.97:
  (768,82.1)-sparse-(768,82.1)-xor(77)-(691,86.2)-xor(85)-(606,86.4)-xor(85)
      -(521,86.8)-xor(86)-(435,86.6)-xor(85)-(350,87.2)-xor(86)-(264,87.4)-code
      ([17,5,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,
      rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927
      ][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,
      rnd150927])-(83,87.4)-WHT
k=768 tau=0.125 theta=0.33 precision=0.10 lmaxc=103.01 lsumc=105.89:
  (768,89.1)-sparse-(768,89.1)-xor(84)-(684,93.2)-xor(92)-(592,93.4)-xor(92)
      -(500,93.8)-xor(93)-(407,93.6)-xor(92)-(315,94.2)-xor(93)-(222,94.4)-code
      ([25,15,W][25,15,W][25,15,W][14,3,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926])-(90,94.4)-WHT
k=768 tau=0.200 theta=0.33 precision=0.10 lmaxc=118.18 lsumc=121.04:
  (768,104.7)-sparse-(768,104.7)-xor(100)-(668,108.4)-xor(107)-(561,108.8)-xor
      (108)-(453,108.6)-xor(107)-(346,109.2)-xor(108)-(238,109.4)-xor(108)
      -(130,109.8)-code([63,57,H][16,15,Chop][20,13,W][31,20,W])-(105,109.8)-WHT
k=768 tau=0.250 theta=0.33 precision=0.10 lmaxc=124.63 lsumc=127.35:
  (768,111.3)-sparse-(768,111.3)-xor(107)-(661,114.6)-xor(113)-(548,115.2)-xor
      (114)-(434,115.4)-xor(114)-(320,115.8)-xor(115)-(205,115.6)-code([5,1,r
      ][25,15,W][25,15,W][25,15,W][25,15,W][25,15,W][25,15,W][25,15,W][25,15,W])
      -(121,115.6)-drop(9)-(112,106.6)-WHT
```

A.6 Solving LPN with *guess-secret*

We present below the chains that use *guess-secret*.

```
% below, lsumc and lmaxc are the logarithmic total and max- complexities
% lrep is the logarithmic number of repetitions applied to WHT

% what follows uses precision .1, with guess

k=32 tau=0.050 theta=0.33 precision=0.10 lmaxc=10.90 lsumc=11.85 lrep=8.57:
  (32,5.1)-sparse-(32,1.2)-guess(13,3)-(19,1.2)-code([19,1,r])-(1,1.2)-WHT
k=32 tau=0.100 theta=0.33 precision=0.10 lmaxc=11.65 lsumc=12.41 lrep=8.87:
  (32,5.1)-sparse-(32,1.2)-guess(23,2)-(9,1.2)-code([9,1,r])-(1,1.2)-WHT
k=32 tau=0.125 theta=0.33 precision=0.10 lmaxc=12.40 lsumc=13.30 lrep=9.96:
  (32,5.1)-sparse-(32,1.2)-guess(26,2)-(6,1.2)-code([6,1,r])-(1,1.2)-WHT
k=32 tau=0.200 theta=0.33 precision=0.10 lmaxc=14.80 lsumc=17.01 lrep=0.00:
  (32,14.8)-drop(1)-(31,13.8)-drop(4)-(27,9.8)-xor(9)-(18,9.6)-xor(8)-(10,10.2)
      -xor(8)-(2,11.4)-drop(1)-(1,10.4)-WHT
k=32 tau=0.250 theta=0.33 precision=0.10 lmaxc=16.30 lsumc=18.42 lrep=0.00:
  (32,16.3)-drop(1)-(31,15.3)-drop(4)-(27,11.3)-xor(11)-(16,10.6)-xor(8)
      -(8,12.2)-WHT

k=48 tau=0.050 theta=0.33 precision=0.10 lmaxc=12.52 lsumc=13.01 lrep=8.27:
  (48,5.7)-sparse-(48,2)-guess(23,2)-(25,2)-code([25,1,r])-(1,2)-WHT
k=48 tau=0.100 theta=0.33 precision=0.10 lmaxc=14.25 lsumc=15.23 lrep=11.35:
  (48,5.7)-sparse-(48,2)-guess(37,2)-(11,2)-code([11,1,r])-(1,2)-WHT
```

```
k=48 tau=0.125 theta=0.33 precision=0.10 lmaxc=15.49 lsumc=16.49 lrep=12.68:
  (48,5.7)-sparse-(48,2)-guess(39,2)-(9,2)-code([9,1,r])-(1,2)-WHT
k=48 tau=0.200 theta=0.33 precision=0.10 lmaxc=19.23 lsumc=21.25 lrep=0.00:
  (48,18.8)-drop(1)-(47,17.8)-drop(4)-(43,13.8)-xor(13)-(30,13.6)-xor(12)
      -(18,14.2)-xor(13)-(5,14.4)-drop(4)-(1,10.4)-WHT
k=48 tau=0.250 theta=0.33 precision=0.10 lmaxc=20.43 lsumc=22.34 lrep=0.00:
  (48,20)-drop(1)-(47,19)-drop(4)-(43,15)-xor(14)-(29,15)-xor(14)-(15,15)-drop
      (2)-(13,13)-WHT

k=64 tau=0.050 theta=0.33 precision=0.10 lmaxc=13.74 lsumc=14.44 lrep=10.04:
  (64,6.1)-sparse-(64,2.2)-guess(38,2)-(26,2.2)-code([26,1,r])-(1,2.2)-WHT
k=64 tau=0.100 theta=0.33 precision=0.10 lmaxc=16.76 lsumc=17.71 lrep=13.80:
  (64,6.1)-sparse-(64,2.2)-guess(52,2)-(12,2.2)-code([12,1,r])-(1,2.2)-WHT
k=64 tau=0.125 theta=0.33 precision=0.10 lmaxc=18.61 lsumc=20.57 lrep=12.07:
  (64,6.8)-sparse-(64,5.6)-xor(1)-(63,9.2)-xor(8)-(55,9.4)-guess(36,2)-(19,9.4)
      -code([19,4,rnd150926])-(4,9.4)-drop(3)-(1,6.4)-WHT
k=64 tau=0.200 theta=0.33 precision=0.10 lmaxc=22.00 lsumc=24.42 lrep=0.00:
  (64,22)-drop(1)-(63,21)-drop(5)-(58,16)-xor(15)-(43,16)-xor(15)-(28,16)-xor
      (14)-(14,17)-WHT
k=64 tau=0.250 theta=0.33 precision=0.10 lmaxc=24.58 lsumc=26.86 lrep=0.00:
  (64,24.5)-drop(1)-(63,23.5)-drop(5)-(58,18.5)-xor(18)-(40,18)-xor(16)-(24,19)
      -xor(17)-(7,20)-WHT

k=100 tau=0.050 theta=0.33 precision=0.10 lmaxc=16.19 lsumc=17.20 lrep=13.37:
  (100,6.7)-sparse-(100,2)-guess(75,2)-(25,2)-code([25,1,r])-(1,2)-WHT
k=100 tau=0.100 theta=0.33 precision=0.10 lmaxc=22.14 lsumc=24.02 lrep=17.74:
  (100,6.9)-sparse-(100,4.3)-xor(1)-(99,6.5)-xor(3)-(96,9)-guess(77,2)-(19,9)-
      code([19,6,rnd150927])-(6,9)-drop(5)-(1,4)-WHT
k=100 tau=0.125 theta=0.33 precision=0.10 lmaxc=24.80 lsumc=27.14 lrep=16.87:
  (100,11.2)-sparse-(100,11.1)-xor(6)-(94,15.2)-xor(14)-(80,15.4)-xor(15)
      -(65,14.8)-guess(47,3)-(18,14.8)-code([18,8,rnd150926])-(8,14.8)-drop(7)
      -(1,7.8)-WHT
k=100 tau=0.200 theta=0.33 precision=0.10 lmaxc=29.75 lsumc=32.06 lrep=0.00:
  (100,29.1)-drop(1)-(99,28.1)-drop(5)-(94,23.1)-xor(22)-(72,23.2)-xor(22)
      -(50,23.4)-xor(22)-(28,23.8)-drop(6)-(22,17.8)-WHT
k=100 tau=0.250 theta=0.33 precision=0.10 lmaxc=30.75 lsumc=32.94 lrep=0.00:
  (100,30.1)-drop(1)-(99,29.1)-drop(5)-(94,24.1)-xor(23)-(71,24.2)-xor(23)
      -(48,24.4)-xor(23)-(25,24.8)-drop(3)-(22,21.8)-WHT

k=256 tau=0.050 theta=0.33 precision=0.10 lmaxc=28.02 lsumc=30.13 lrep=17.28:
  (256,8.1)-sparse-(256,4.2)-xor(1)-(255,6.3)-xor(2)-(253,9.6)-xor(8)
      -(245,10.2)-guess(178,1)-(67,10.2)-code([67,1,r])-(1,10.2)-WHT
k=256 tau=0.100 theta=0.33 precision=0.10 lmaxc=43.49 lsumc=45.99 lrep=27.36:
  (256,26)-sparse-(256,26)-xor(21)-(235,30)-xor(29)-(206,30)-xor(29)-(177,30)-
      xor(29)-(148,30)-guess(100,4)-(48,30)-code([11,4,S][18,5,rnd150927][19,6,
      rnd150927])-(15,30)-drop(14)-(1,16)-WHT
k=256 tau=0.125 theta=0.33 precision=0.10 lmaxc=47.35 lsumc=49.90 lrep=0.00:
  (256,35.6)-sparse-(256,35.6)-xor(31)-(225,39.2)-xor(38)-(187,39.4)-xor(38)
      -(149,39.8)-xor(39)-(110,39.6)-code([15,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,7,rnd150926][19,7,rnd150926])-(37,39.6)-WHT
k=256 tau=0.200 theta=0.33 precision=0.10 lmaxc=53.82 lsumc=56.34 lrep=1.00:
  (256,42.4)-sparse-(256,42.4)-xor(38)-(218,45.8)-xor(45)-(173,45.6)-xor(44)
      -(129,46.2)-xor(45)-(84,46.4)-guess(1,1)-(83,46.4)-code([14,6,FP][25,15,W
      ][25,15,W][19,7,rnd150927])-(43,46.4)-drop(1)-(42,45.4)-WHT
k=256 tau=0.250 theta=0.33 precision=0.10 lmaxc=56.88 lsumc=59.47 lrep=0.00:
  (256,56.7)-drop(1)-(255,55.7)-drop(7)-(248,48.7)-xor(48)-(200,48.4)-xor(47)
      -(153,48.8)-xor(47)-(106,49.6)-xor(59)-(47,39.2)-WHT

% what follows uses precision 1

k=512 tau=0.050 theta=0.33 precision=1.00 lmaxc=47.29 lsumc=49.56 lrep=39.23:
  (512,10)-sparse-(512,9)-xor(2)-(510,15)-xor(14)-(496,15)-xor(14)-(482,15)-
      guess(417,2)-(65,15)-code([13,1,r][14,1,r][19,3,rnd150927][19,3,rnd150927
      ])-(8,15)-drop(7)-(1,8)-WHT
k=512 tau=0.100 theta=0.33 precision=1.00 lmaxc=71.09 lsumc=73.68 lrep=1.60:
  (512,58)-sparse-(512,58)-xor(53)-(459,62)-xor(61)-(398,62)-xor(61)-(337,62)-
      xor(61)-(276,62)-xor(61)-(215,62)-guess(2,1)-(213,62)-code([5,1,r][18,5,
      rnd150927][19,5,rnd150927][19,5,rnd150927][19,6,rnd150927][19,6,rnd150927
      ][19,5,rnd150927][19,5,rnd150927][19,5,rnd150927][19,5,rnd150927][19,5,
      rnd150927][19,5,rnd150927])-(58,62)-WHT
k=512 tau=0.125 theta=0.33 precision=1.00 lmaxc=76.24 lsumc=78.97 lrep=0.19:
  (512,63)-sparse-(512,63)-xor(58)-(454,67)-xor(66)-(388,67)-xor(66)-(322,67)-
      xor(66)-(256,67)-xor(65)-(191,68)-guess(1,0)-(190,68)-code([19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,7,rnd150926][19,7,rnd150926][19,7,rnd150926][19,7,rnd150926
      ])-(64,68)-WHT
k=512 tau=0.200 theta=0.33 precision=1.00 lmaxc=86.79 lsumc=89.28 lrep=2.16:
  (512,74)-sparse-(512,74)-xor(70)-(442,77)-xor(76)-(366,77)-xor(75)-(291,78)-
      xor(77)-(214,78)-xor(77)-(137,78)-guess(3,1)-(134,78)-code([25,15,W
      ][25,15,W][25,15,W][25,15,W][25,15,W][9,4,BBD])-(79,78)-drop(6)-(73,72)-
      WHT
k=512 tau=0.250 theta=0.33 precision=1.00 lmaxc=92.36 lsumc=94.85 lrep=1.68:
  (512,79)-sparse-(512,79)-xor(74)-(438,83)-xor(82)-(356,83)-xor(82)-(274,83)-
      xor(81)-(193,84)-xor(83)-(110,84)-guess(2,1)-(108,84)-code([15,11,H
      ][31,26,H][31,20,W][31,20,W])-(77,84)-WHT

k=768 tau=0.050 theta=0.33 precision=1.00 lmaxc=65.98 lsumc=68.15 lrep=58.89:
  (768,10)-sparse-(768,8)-xor(1)-(767,14)-xor(12)-(755,15)-xor(14)-(741,15)-
      guess(682,2)-(59,15)-code([7,1,r][7,1,r][19,3,rnd150927][7,1,r][19,3,
      rnd150927])-(9,15)-drop(8)-(1,7)-WHT
k=768 tau=0.100 theta=0.33 precision=1.00 lmaxc=96.34 lsumc=99.21 lrep=0.76:
  (768,82)-sparse-(768,82)-xor(77)-(691,86)-xor(85)-(606,86)-xor(85)-(521,86)-
      xor(85)-(436,86)-xor(85)-(351,86)-xor(84)-(267,87)-guess(5,0)-(262,87)-
      code([15,5,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927
      ][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,
      rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927][19,6,rnd150927
      ][19,6,rnd150927])-(83,87)-WHT
k=768 tau=0.125 theta=0.33 precision=1.00 lmaxc=103.42 lsumc=106.18 lrep=2.44:
  (768,89)-sparse-(768,89)-xor(84)-(684,93)-xor(92)-(592,93)-xor(91)-(501,94)-
      xor(93)-(408,94)-xor(93)-(315,94)-xor(92)-(223,95)-guess(4,1)-(219,95)-
      code([25,15,W][25,15,W][18,6,rnd150926][18,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926][19,7,rnd150926][19,7,rnd150926][19,7,
      rnd150926][19,7,rnd150926])-(88,95)-drop(1)-(87,94)-WHT
k=768 tau=0.200 theta=0.33 precision=1.00 lmaxc=118.57 lsumc=121.12 lrep=0.00:
  (768,116)-drop(7)-(761,109)-xor(108)-(653,109)-xor(108)-(545,109)-xor(108)
      -(437,109)-xor(108)-(329,109)-xor(107)-(222,110)-xor(109)-(113,110)-drop
      (7)-(106,103)-WHT
k=768 tau=0.250 theta=0.33 precision=1.00 lmaxc=125.01 lsumc=127.63 lrep=2.25:
  (768,111)-sparse-(768,111)-xor(106)-(662,115)-xor(114)-(548,115)-xor(114)
      -(434,115)-xor(113)-(321,116)-xor(115)-(206,116)-guess(3,1)-(203,116)-code
      ([3,1,r][25,15,W][25,15,W][25,15,W][25,15,W][25,15,W][25,15,W][25,15,W
      ][25,15,W])-(121,116)-drop(11)-(110,105)-WHT
```

A.7 Solving $\mathsf{LPN}_{512,0.125}$

We present here the chains for a complete recovery of the secret in the case of $\mathsf{LPN}_{512,0.125}$.

```
Step 1:
k=512 tau=0.125 theta=0.33 precision=0.10 lmaxc=76.22 lsumc=78.85:
  (512,63.3)-sparse-(512,63.3)-xor(59)-(453,66.6)-xor(65)-(388,67.2)-xor(66)
      -(322,67.4)-xor(66)-(256,67.8)-xor(67)-(189,67.6)-code([18,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,7,rnd150926][19,7,rnd150926][19,7,rnd150926][19,7,rnd150926
      ])-(64,67.6)-WHT

Step 2:
k=448 tau=0.125 theta=0.11 precision=0.10 lmaxc=69.20 lsumc=71.99:
  (448,56.6)-sparse-(448,56.6)-xor(52)-(396,60.2)-xor(59)-(337,60.4)-xor(59)
      -(278,60.8)-xor(60)-(218,60.6)-xor(59)-(159,61.2)-code([25,15,W][25,15,W
      ][14,3,rnd150926][19,4,rnd150926][19,4,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926])-(59,61.2)-drop(1)-(58,60.2)-WHT

Step 3:
k=390 tau=0.125 theta=0.04 precision=0.10 lmaxc=62.70 lsumc=65.39:
  (390,50.3)-sparse-(390,50.3)-xor(46)-(344,53.6)-xor(52)-(292,54.2)-xor(53)
      -(239,54.4)-xor(53)-(186,54.8)-xor(54)-(132,54.6)-code([25,15,W][13,4,
      rnd150926][18,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,7,rnd150926
      ][19,7,rnd150926])-(51,54.6)-WHT

Step 4:
```

```
k=339 tau=0.125 theta=0.01 precision=0.10 lmaxc=57.06 lsumc=59.78:
  (339,44.7)-sparse-(339,44.7)-xor(40)-(299,48.4)-xor(47)-(252,48.8)-xor(48)
      -(204,48.6)-xor(47)-(157,49.2)-xor(48)-(109,49.4)-code([3,1,r][25,15,W
      ][25,15,W][18,5,rnd150926][19,4,rnd150926][19,6,rnd150926])-(46,49.4)-WHT

Step 5:
k=293 tau=0.125 theta=0.00 precision=0.10 lmaxc=51.81 lsumc=54.52:
  (293,39.7)-sparse-(293,39.7)-xor(35)-(258,43.4)-xor(42)-(216,43.8)-xor(43)
      -(173,43.6)-xor(42)-(131,44.2)-xor(43)-(88,44.4)-code([25,15,W][25,15,W
      ][19,5,rnd150926][19,6,rnd150926])-(41,44.4)-WHT

Step 6:
k=252 tau=0.125 theta=0.00 precision=0.10 lmaxc=47.00 lsumc=49.61:
  (252,34.8)-sparse-(252,34.8)-xor(30)-(222,38.6)-xor(37)-(185,39.2)-xor(38)
      -(147,39.4)-xor(38)-(109,39.8)-code([14,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,7,rnd150926][19,7,rnd150926])-(37,39.8)-WHT

Step 7:
k=215 tau=0.125 theta=0.00 precision=0.10 lmaxc=42.36 lsumc=44.86:
  (215,30.7)-sparse-(215,30.7)-xor(26)-(189,34.4)-xor(33)-(156,34.8)-xor(34)
      -(122,34.6)-xor(33)-(89,35.2)-code([7,1,r][25,15,W][19,6,rnd150926][19,4,
      rnd150926][19,6,rnd150926])-(32,35.2)-WHT

Step 8:
k=183 tau=0.125 theta=0.00 precision=0.10 lmaxc=38.35 lsumc=40.76:
  (183,27)-sparse-(183,27)-xor(23)-(160,30)-xor(28)-(132,31)-xor(30)-(102,31)-
      xor(30)-(72,31)-code([25,15,W][13,4,rnd150926][16,5,rnd150926][18,5,
      rnd150926])-(29,31)-WHT

Step 9:
k=154 tau=0.125 theta=0.00 precision=0.10 lmaxc=34.38 lsumc=36.94:
  (154,23.5)-sparse-(154,23.5)-xor(19)-(135,27)-xor(26)-(109,27)-xor(26)
      -(83,27)-xor(25)-(58,28)-code([25,15,W][15,5,rnd150926][18,5,rnd150926])
      -(25,28)-WHT

Step 10:
k=129 tau=0.125 theta=0.00 precision=0.10 lmaxc=31.34 lsumc=33.82:
  (129,20.6)-sparse-(129,20.6)-xor(16)-(113,24.2)-xor(23)-(90,24.4)-xor(23)
      -(67,24.8)-xor(24)-(43,24.6)-code([25,15,W][18,7,rnd150926])-(22,24.6)-WHT

Step 11:
k=107 tau=0.125 theta=0.00 precision=0.10 lmaxc=27.85 lsumc=30.15:
  (107,27)-drop(1)-(106,26)-drop(5)-(101,21)-xor(20)-(81,21)-xor(20)-(61,21)-
      xor(20)-(41,21)-xor(19)-(22,22)-drop(2)-(20,20)-WHT

Step 12:
k=87 tau=0.125 theta=0.00 precision=0.10 lmaxc=24.54 lsumc=27.05:
  (87,24.1)-drop(1)-(86,23.1)-drop(5)-(81,18.1)-xor(17)-(64,18.2)-xor(17)
      -(47,18.4)-xor(17)-(30,18.8)-xor(17)-(13,19.6)-WHT

Step 13:
k=74 tau=0.125 theta=0.00 precision=0.10 lmaxc=23.30 lsumc=25.55:
  (74,22.9)-drop(1)-(73,21.9)-drop(5)-(68,16.9)-xor(16)-(52,16.8)-xor(15)
      -(37,17.6)-xor(21)-(16,13.2)-WHT

Step 14:
k=58 tau=0.125 theta=0.00 precision=0.10 lmaxc=19.80 lsumc=22.19:
  (58,19.8)-drop(1)-(57,18.8)-drop(5)-(52,13.8)-xor(13)-(39,13.6)-xor(12)
      -(27,14.2)-xor(13)-(14,14.4)-drop(1)-(13,13.4)-WHT

Step 15:
k=45 tau=0.125 theta=0.00 precision=0.10 lmaxc=17.29 lsumc=19.69:
  (45,17)-drop(1)-(44,16)-drop(5)-(39,11)-xor(9)-(30,12)-xor(11)-(19,12)-drop
      (1)-(18,11)-xor(8)-(10,13)-WHT

Step 16:
k=35 tau=0.125 theta=0.00 precision=0.10 lmaxc=15.97 lsumc=17.96:
  (35,15.7)-drop(1)-(34,14.7)-drop(4)-(30,10.7)-xor(10)-(20,10.4)-xor(9)
      -(11,10.8)-drop(1)-(10,9.8)-WHT

Step 17:
k=25 tau=0.125 theta=0.00 precision=0.10 lmaxc=13.30 lsumc=15.33:
  (25,12.9)-drop(1)-(24,11.9)-drop(3)-(21,8.9)-xor(8)-(13,8.8)-xor(7)-(6,9.6)-
      WHT

Step 18:
k=19 tau=0.125 theta=0.00 precision=0.10 lmaxc=12.21 lsumc=14.30:
  (19,11.9)-drop(1)-(18,10.9)-drop(3)-(15,7.9)-xor(9)-(6,5.8)-xor(1)-(5,9.6)-
      WHT

Step 19:
k=14 tau=0.125 theta=0.00 precision=0.10 lmaxc=10.93 lsumc=10.93:
  (14,4)-sparse-guess(14)
```

Overall, the complexity of going through all chains is

$$\begin{aligned}&2^{78.85} + 2^{71.99} + 2^{65.39} + 2^{59.78} + 2^{54.52} + 2^{49.61} + 2^{44.86} + 2^{40.76} + 2^{36.94}\\ &+ 2^{33.82} + 2^{30.15} + 2^{27.05} + 2^{25.55} + 2^{22.19} + 2^{19.69} + 2^{17.96} + 2^{15.33}\\ &+ 2^{14.30} + 2^{10.93} \approx 2^{78.86}\end{aligned}$$

with a probability of success at least $\frac{1}{2}$.

A.8 Chains from [GJL14] and [ZJW16]

In this section, we present the analysis on the results from ASIACRYPT'14 [GJL14] and EUROCRYPT'16 [ZJW16]. We compute complexities and biases following our formulas, and always use the largest pool of codes that we had to have the most favorable results. Sometimes, the number of initial queries is either too small or too large to have a failure probability close to 33% so we correct it to be able to compare chains. Together with these chains, we provide the best ones that we found.

In the data below, `lcomp` denotes the logarithmic complexity in each step of the chain, `lbc2` denotes the logarithmic value of δ_s^2, `lr_all` denotes the logarithmic number of necessary average of repetitions of the overall chain for guessing to succeed, `lr_wht` denotes the logarithmic number of repetitions which apply to WHT only, `lmaxcomp` denotes the logarithmic max-complexity, `ltotcomp` denotes the logarithmic total complexity of the chain. The name of the reduction steps differ a bit: `sparse` denotes *sparse-secret*, `part` denotes *partition-reduce*, `xor` denotes *xor-reduce*, `guess` denotes *guess-secret*, and `code` denotes *code-reduce*. For each chain, we put the computations for the exact chain instead of the rounded one. Each reduction step is followed by the indication of the vertex $(k, \log_2 n)$.

A.8.1 Chains for $\mathsf{LPN}_{512,1/8}$

Here is the chain proposed in [GJL14] (proceedings) with a claimed complexity of $2^{79.9}$:

```
chain for LPN_{512,0.125} with n=2^66.3000
       sparse-(512,66.30) lcomp=79.245 lbc2=-0.830
     part(63)-(449,66.15) lcomp=75.300 lbc2=-1.660
     part(63)-(386,65.97) lcomp=74.956 lbc2=-3.320
     part(63)-(323,65.78) lcomp=74.565 lbc2=-6.641
     part(63)-(260,65.55) lcomp=74.111 lbc2=-13.281
     part(63)-(197,65.28) lcomp=73.571 lbc2=-26.562
     part(63)-(134,64.94) lcomp=72.900 lbc2=-53.125
  guess(10u2)-(124,64.94) lcomp=0.000 lbc2=-53.125
         code-(64,64.94) lcomp=71.899 lbc2=-65.809
        WHT(theta=100.00) lcomp=75.125
           lr_all=0.184 lr_wht=5.807
                      lmaxcomp=81.12
                      ltotcomp=81.58
  for the used code, log_2 bc^2 =-12.684598566058570684917951628899665523
  used code family: old+QP+rnd
  params=[5,1,r][25,15,W][25,15,W][25,12,FP][25,15,W][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{512,0.125} with n=2^73.2200
       sparse-(512,73.22) lcomp=85.769 lbc2=-0.830
     part(63)-(449,73.22) lcomp=82.220 lbc2=-1.660
     part(63)-(386,73.22) lcomp=82.029 lbc2=-3.320
     part(63)-(323,73.22) lcomp=81.810 lbc2=-6.641
     part(63)-(260,73.22) lcomp=81.552 lbc2=-13.281
```

```
    part(63)-(197,73.21) lcomp=81.238 lbc2=-26.562
    part(63)-(134,73.21) lcomp=80.836 lbc2=-53.125
  guess(10u2)-(124,73.21) lcomp=0.000 lbc2=-53.125
          code-(64,73.21) lcomp=80.167 lbc2=-65.809
         WHT(theta=28.00) lcomp=79.300
          lr_all=0.184 lr_wht=5.807
                     lmaxcomp=85.95
                     ltotcomp=86.96
  for the used code, log_2 bc^2 =-12.684598566058570684917951628899665523
  used code family: old+QP+rnd
  params=[5,1,r][25,15,W][25,15,W][25,12,FP][25,15,W][19,6,rnd150926]
```

Here is the chain presented by [GJL14] (presented at the conference) with a claimed complexity of $2^{79.7}$:

```
chain for LPN_{512,0.125} with n=2^63.7000
       sparse-(512,63.70) lcomp=76.467 lbc2=-0.830
      xor(62)-(450,64.40) lcomp=73.400 lbc2=-1.660
      xor(62)-(388,65.80) lcomp=74.614 lbc2=-3.320
      xor(62)-(326,68.60) lcomp=77.200 lbc2=-6.641
      xor(62)-(264,74.20) lcomp=82.549 lbc2=-13.281
      xor(62)-(202,85.40) lcomp=93.444 lbc2=-26.562
  guess(22u2)-(180,85.40) lcomp=0.000 lbc2=-26.562
          code-(60,85.40) lcomp=92.892 lbc2=-58.986
          WHT(theta=0.00) lcomp=91.307
          lr_all=1.091 lr_wht=7.989
                     lmaxcomp=100.39
                     ltotcomp=100.43
  for the used code, log_2 bc^2 =-32.423925711598927470606260602594044481
  used code family: old+QP+rnd
  params=[3,1,r][25,15,W][19,4,rnd150926][19,4,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{512,0.125} with n=2^63.1030
       sparse-(512,63.10) lcomp=76.101 lbc2=-0.830
      xor(62)-(450,63.21) lcomp=72.206 lbc2=-1.660
      xor(62)-(388,63.41) lcomp=72.226 lbc2=-3.320
      xor(62)-(326,63.82) lcomp=72.424 lbc2=-6.641
      xor(62)-(264,64.65) lcomp=72.997 lbc2=-13.281
      xor(62)-(202,66.30) lcomp=74.340 lbc2=-26.562
  guess(22u2)-(180,66.30) lcomp=0.000 lbc2=-26.562
          code-(60,66.30) lcomp=73.788 lbc2=-58.986
         WHT(theta=26.00) lcomp=72.717
          lr_all=1.091 lr_wht=7.989
                     lmaxcomp=81.80
                     ltotcomp=81.90
  for the used code, log_2 bc^2 =-32.423925711598927470606260602594044481
  used code family: old+QP+rnd
  params=[3,1,r][25,15,W][19,4,rnd150926][19,4,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the chain from [ZJW16] based on LF1 with a claimed complexity of $2^{75.897}$:

```
chain for LPN_{512,0.125} with n=2^71.2910
     drop( 5)-(507,66.29) lcomp=72.245 lbc2=-0.830
       sparse-(507,66.29) lcomp=79.225 lbc2=-0.830
     part(63)-(444,66.14) lcomp=75.277 lbc2=-1.660
     part(63)-(381,65.96) lcomp=74.930 lbc2=-3.320
     part(63)-(318,65.76) lcomp=74.535 lbc2=-6.641
     part(63)-(255,65.53) lcomp=74.076 lbc2=-13.281
     part(63)-(192,65.26) lcomp=73.527 lbc2=-26.562
  guess(20u1)-(172,65.26) lcomp=0.000 lbc2=-26.562
          code-(62,65.26) lcomp=72.686 lbc2=-55.531
          WHT(theta=0.00) lcomp=73.371
           lr_all=1.905 lr_wht=4.392
                       lmaxcomp=81.13
                       ltotcomp=81.79
  for the used code, log_2 bc^2 =-28.968290632587130827717807602787613454
  used code family: old+QP+rnd
  params=[25,15,W][14,5,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{512,0.125} with n=2^70.5660
     drop( 5)-(507,65.57) lcomp=71.520 lbc2=-0.830
       sparse-(507,65.57) lcomp=78.383 lbc2=-0.830
     part(63)-(444,65.30) lcomp=74.552 lbc2=-1.660
     part(63)-(381,64.97) lcomp=74.094 lbc2=-3.320
     part(63)-(318,64.55) lcomp=73.545 lbc2=-6.641
     part(63)-(255,63.94) lcomp=72.860 lbc2=-13.281
     part(63)-(192,62.88) lcomp=71.937 lbc2=-26.562
  guess(20u1)-(172,62.88) lcomp=0.000 lbc2=-26.562
          code-(62,62.88) lcomp=70.309 lbc2=-55.531
         WHT(theta=31.00) lcomp=73.032
           lr_all=1.905 lr_wht=4.392
                       lmaxcomp=80.29
                       ltotcomp=81.07
  for the used code, log_2 bc^2 =-28.968290632587130827717807602787613454
  used code family: old+QP+rnd
  params=[25,15,W][14,5,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

For comparison, here is the same computation using only perfect codes:

```
chain for LPN_{512,0.125} with n=2^70.8630
     drop( 5)-(507,65.86) lcomp=71.817 lbc2=-0.830
       sparse-(507,65.86) lcomp=78.578 lbc2=-0.830
     part(63)-(444,65.65) lcomp=74.849 lbc2=-1.660
     part(63)-(381,65.40) lcomp=74.444 lbc2=-3.320
     part(63)-(318,65.10) lcomp=73.973 lbc2=-6.641
     part(63)-(255,64.71) lcomp=73.409 lbc2=-13.281
     part(63)-(192,64.19) lcomp=72.706 lbc2=-26.562
  guess(20u1)-(172,64.19) lcomp=0.000 lbc2=-26.562
          code-(62,64.19) lcomp=71.612 lbc2=-56.834
         WHT(theta=31.00) lcomp=73.169
           lr_all=1.905 lr_wht=4.392
                     lmaxcomp=80.48
                     ltotcomp=81.27
  for the used code, log_2 bc^2 =-30.272072056802635203237971232374080351
  used code family: old
  params=[5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][7,1,r
      ][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][23,12,G][23,12,G][23,12,G][23,12,G]
```

Here is the chain from [ZJW16] based on LF2 with a claimed complexity of $2^{74.732}$:

```
chain for LPN_{512,0.125} with n=2^69.9870
     drop( 5)-(507,64.99) lcomp=70.941 lbc2=-0.830
       sparse-(507,64.99) lcomp=78.045 lbc2=-0.830
      xor(64)-(443,64.97) lcomp=73.973 lbc2=-1.660
      xor(64)-(379,64.95) lcomp=73.765 lbc2=-3.320
      xor(64)-(315,64.90) lcomp=73.514 lbc2=-6.641
      xor(64)-(251,64.79) lcomp=73.195 lbc2=-13.281
      xor(64)-(187,64.58) lcomp=72.764 lbc2=-26.562
  guess(17u1)-(170,64.58) lcomp=0.000 lbc2=-26.562
          code-(62,64.58) lcomp=71.993 lbc2=-54.892
          WHT(theta=0.00) lcomp=73.232
           lr_all=1.497 lr_wht=4.170
                     lmaxcomp=79.54
                     ltotcomp=80.45
  for the used code, log_2 bc^2 =-28.329845374218742534238304113938514928
  used code family: old+QP+rnd
  params=[25,15,W][13,4,rnd150926][18,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,7,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{512,0.125} with n=2^69.9140
     drop( 5)-(507,64.91) lcomp=70.868 lbc2=-0.830
       sparse-(507,64.91) lcomp=77.605 lbc2=-0.830
      xor(64)-(443,64.83) lcomp=73.900 lbc2=-1.660
      xor(64)-(379,64.66) lcomp=73.619 lbc2=-3.320
      xor(64)-(315,64.31) lcomp=73.222 lbc2=-6.641
      xor(64)-(251,63.62) lcomp=72.611 lbc2=-13.281
      xor(64)-(187,62.25) lcomp=71.596 lbc2=-26.562
  guess(17u1)-(170,62.25) lcomp=0.000 lbc2=-26.562
          code-(62,62.25) lcomp=69.657 lbc2=-54.892
         WHT(theta=28.00) lcomp=72.990
           lr_all=1.497 lr_wht=4.170
```

```
                    lmaxcomp=79.10
                    ltotcomp=80.09
  for the used code, log_2 bc^2 =-28.329845374218742534238304113938514928
  used code family: old+QP+rnd
  params=[25,15,W][13,4,rnd150926][18,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,7,rnd150926]
```

For comparison, here is the same computation using only perfect codes:

```
chain for LPN_{512,0.125} with n=2^69.9540
     drop( 5)-(507,64.95) lcomp=70.908 lbc2=-0.830
       sparse-(507,64.95) lcomp=78.027 lbc2=-0.830
      xor(64)-(443,64.91) lcomp=73.940 lbc2=-1.660
      xor(64)-(379,64.82) lcomp=73.699 lbc2=-3.320
      xor(64)-(315,64.63) lcomp=73.382 lbc2=-6.641
      xor(64)-(251,64.26) lcomp=72.931 lbc2=-13.281
      xor(64)-(187,63.53) lcomp=72.236 lbc2=-26.562
  guess(17u1)-(170,63.53) lcomp=0.000 lbc2=-26.562
         code-(62,63.53) lcomp=70.937 lbc2=-56.158
        WHT(theta=20.00) lcomp=73.090
          lr_all=1.497 lr_wht=4.170
                    lmaxcomp=79.52
                    ltotcomp=80.37
  for the used code, log_2 bc^2 =-29.595714087454877581434997641085959931
  used code family: old
  params=[5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r
      ][7,1,r][7,1,r][7,1,r][7,1,r][23,12,G][23,12,G][23,12,G][23,12,G]
```

Here is the chain from [ZJW16] based on LF(4) with a claimed complexity of $2^{72.844}$:

```
chain for LPN_{512,0.125} with n=2^63.5260
     drop(10)-(502,53.53) lcomp=64.525 lbc2=-0.830
       sparse-(502,53.53) lcomp=66.734 lbc2=-0.830
     lf4(156)-(346,53.52) lcomp=115.024 lbc2=-3.320
     lf4(156)-(190,53.49) lcomp=114.473 lbc2=-13.281
  guess(16u1)-(174,53.49) lcomp=0.000 lbc2=-13.281
         code-(60,53.49) lcomp=60.934 lbc2=-43.739
         WHT(theta=0.00) lcomp=70.675
          lr_all=1.366 lr_wht=4.087
                    lmaxcomp=116.39
                    ltotcomp=117.14
  for the used code, log_2 bc^2 =-30.457352130329587529315901036020611109
  used code family: old+QP+rnd
  params=[25,15,W][19,4,rnd150926][16,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{512,0.125} with n=2^63.3730
     drop(10)-(502,53.37) lcomp=64.372 lbc2=-0.830
       sparse-(502,53.37) lcomp=66.329 lbc2=-0.830
```

```
    lf4(156)-(346,52.91) lcomp=114.718 lbc2=-3.320
    lf4(156)-(190,51.04) lcomp=113.249 lbc2=-13.281
  guess(16u1)-(174,51.04) lcomp=0.000 lbc2=-13.281
          code-(60,51.04) lcomp=58.486 lbc2=-43.739
         WHT(theta=30.00) lcomp=70.609
           lr_all=1.366 lr_wht=4.087
                      lmaxcomp=116.08
                      ltotcomp=116.53
  for the used code, log_2 bc^2 =-30.457352130329587529315901036020611109
  used code family: old+QP+rnd
  params=[25,15,W][19,4,rnd150926][16,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the chain we obtain by running our algorithm with precision 0.1, after removing the roundings and adjusting the number of queries:

```
chain for LPN_{512,0.125} with n=2^63.2990
       sparse-(512,63.30) lcomp=76.215 lbc2=-0.830
      xor(59)-(453,66.60) lcomp=75.598 lbc2=-1.660
      xor(65)-(388,67.20) lcomp=76.019 lbc2=-3.320
      xor(66)-(322,67.39) lcomp=75.992 lbc2=-6.641
      xor(66)-(256,67.78) lcomp=76.115 lbc2=-13.281
      xor(67)-(189,67.57) lcomp=75.784 lbc2=-26.562
          code-(64,67.57) lcomp=75.130 lbc2=-60.165
         WHT(theta=28.00) lcomp=75.528
                      lmaxcomp=76.22
                      ltotcomp=78.84
  for the used code, log_2 bc^2 =-33.602837640638774272597586977976944615
  used code family: old+QP+rnd
  params=[18,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,7,rnd150926][19,7,rnd150926][19,7,rnd150926
      ][19,7,rnd150926]
```

For comparison, here is the same computation using only perfect and quasi-perfect codes:

```
chain for LPN_{512,0.125} with n=2^63.3400
       sparse-(512,63.34) lcomp=76.240 lbc2=-0.830
      xor(59)-(453,66.68) lcomp=75.680 lbc2=-1.660
      xor(65)-(388,67.36) lcomp=76.183 lbc2=-3.320
      xor(66)-(322,67.72) lcomp=76.320 lbc2=-6.641
      xor(66)-(256,68.44) lcomp=76.771 lbc2=-13.281
      xor(67)-(189,68.88) lcomp=76.880 lbc2=-26.562
          code-(64,68.88) lcomp=76.442 lbc2=-61.459
        WHT(theta=18.00) lcomp=76.009
                      lmaxcomp=76.88
                      ltotcomp=79.36
  for the used code, log_2 bc^2 =-34.896841882956364951443303534972935944
  used code family: old+QP
  params=[7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r][7,1,r
      ][8,2,iGop][11,4,S][25,12,FP][25,12,FP][25,12,FP][25,12,FP]
```

For comparison, here is the same computation using only perfect codes:

```
chain for LPN_{512,0.125} with n=2^63.3490
      sparse-(512,63.35) lcomp=76.245 lbc2=-0.830
     xor(59)-(453,66.70) lcomp=75.698 lbc2=-1.660
     xor(65)-(388,67.40) lcomp=76.219 lbc2=-3.320
     xor(66)-(322,67.79) lcomp=76.392 lbc2=-6.641
     xor(66)-(256,68.58) lcomp=76.915 lbc2=-13.281
     xor(67)-(189,69.17) lcomp=77.168 lbc2=-26.562
         code-(64,69.17) lcomp=76.730 lbc2=-61.749
        WHT(theta=18.00) lcomp=76.150
                   lmaxcomp=77.17
                   ltotcomp=79.51
  for the used code, log_2 bc^2 =-35.18627839534300082832444302208873451 4
  used code family: old
  params=[5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][5,1,r][7,1,r][7,1,r][7,1,r
    ][7,1,r][7,1,r][6,1,r][7,1,r][7,1,r][7,1,r][23,12,G][23,12,G][23,12,G
    ][23,12,G]
```

A.8.2 Chains for $\mathsf{LPN}_{532,1/8}$

Here is the chain proposed in [GJL14] with a claimed complexity of $2^{81.82}$:

```
chain for LPN_{532,0.125} with n=2^68.0000
      sparse-(532,68.00) lcomp=81.153 lbc2=-0.830
    part(65)-(467,67.81) lcomp=77.055 lbc2=-1.660
    part(65)-(402,67.58) lcomp=76.675 lbc2=-3.320
    part(65)-(337,67.32) lcomp=76.236 lbc2=-6.641
    part(65)-(272,67.00) lcomp=75.719 lbc2=-13.281
    part(65)-(207,66.58) lcomp=75.087 lbc2=-26.562
    part(65)-(142,66.00) lcomp=74.278 lbc2=-53.125
  guess(12u2)-(130,66.00) lcomp=0.000 lbc2=-53.125
         code-(66,66.00) lcomp=73.022 lbc2=-66.904
       WHT(theta=100.00) lcomp=77.153
          lr_all=0.290 lr_wht=6.304
                   lmaxcomp=83.75
                   ltotcomp=84.06
  for the used code, log_2 bc^2 =-13.779193042112753687122226810335264979
  used code family: old+QP+rnd
  params=[25,15,W][25,15,W][25,15,W][23,12,G][13,4,rnd150926][19,5,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{532,0.125} with n=2^74.3600
      sparse-(532,74.36) lcomp=87.314 lbc2=-0.830
    part(65)-(467,74.36) lcomp=83.415 lbc2=-1.660
    part(65)-(402,74.36) lcomp=83.225 lbc2=-3.320
    part(65)-(337,74.35) lcomp=83.007 lbc2=-6.641
    part(65)-(272,74.35) lcomp=82.750 lbc2=-13.281
    part(65)-(207,74.35) lcomp=82.439 lbc2=-26.562
    part(65)-(142,74.35) lcomp=82.042 lbc2=-53.125
 guess(12u2)-(130,74.35) lcomp=0.000 lbc2=-53.125
        code-(66,74.35) lcomp=81.369 lbc2=-66.904
       WHT(theta=33.00) lcomp=80.549
         lr_all=0.290 lr_wht=6.304
                    lmaxcomp=87.60
                    ltotcomp=88.62
 for the used code, log_2 bc^2 =-13.779193042112753687122226810335264979
 used code family: old+QP+rnd
 params=[25,15,W][25,15,W][25,15,W][23,12,G][13,4,rnd150926][19,5,rnd150926]
```

Here is the chain from [ZJW16] based on LF1 with a claimed complexity of $2^{78.182}$:

```
chain for LPN_{532,0.125} with n=2^73.5840
    drop( 5)-(527,68.58) lcomp=74.538 lbc2=-0.830
      sparse-(527,68.58) lcomp=81.476 lbc2=-0.830
    part(65)-(462,68.46) lcomp=77.626 lbc2=-1.660
    part(65)-(397,68.32) lcomp=77.310 lbc2=-3.320
    part(65)-(332,68.17) lcomp=76.954 lbc2=-6.641
    part(65)-(267,68.00) lcomp=76.544 lbc2=-13.281
    part(65)-(202,67.81) lcomp=76.059 lbc2=-26.562
 guess(20u1)-(182,67.81) lcomp=0.000 lbc2=-26.562
        code-(64,67.81) lcomp=75.313 lbc2=-57.882
        WHT(theta=0.00) lcomp=75.597
         lr_all=1.905 lr_wht=4.392
                    lmaxcomp=83.38
                    ltotcomp=84.06
 for the used code, log_2 bc^2 =-31.319243871934051230338929039571812625
 used code family: old+QP+rnd
 params=[5,1,r][25,15,W][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
     rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{532,0.125} with n=2^72.6360
    drop( 5)-(527,67.64) lcomp=73.590 lbc2=-0.830
      sparse-(527,67.64) lcomp=80.500 lbc2=-0.830
    part(65)-(462,67.38) lcomp=76.678 lbc2=-1.660
    part(65)-(397,67.08) lcomp=76.235 lbc2=-3.320
    part(65)-(332,66.69) lcomp=75.709 lbc2=-6.641
    part(65)-(267,66.15) lcomp=75.060 lbc2=-13.281
    part(65)-(202,65.28) lcomp=74.209 lbc2=-26.562
 guess(20u1)-(182,65.28) lcomp=0.000 lbc2=-26.562
        code-(64,65.28) lcomp=72.790 lbc2=-57.882
       WHT(theta=30.00) lcomp=75.153
         lr_all=1.905 lr_wht=4.392
                    lmaxcomp=82.41
                    ltotcomp=83.19
 for the used code, log_2 bc^2 =-31.319243871934051230338929039571812625
 used code family: old+QP+rnd
```

```
  params=[5,1,r][25,15,W][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the chain from [ZJW16] based on LF2 with a claimed complexity of $2^{76.902}$:

```
chain for LPN_{532,0.125} with n=2^73.9830
     drop( 7)-(525,66.98) lcomp=74.972 lbc2=-0.830
       sparse-(525,66.98) lcomp=80.115 lbc2=-0.830
      xor(66)-(459,66.97) lcomp=76.019 lbc2=-1.660
      xor(66)-(393,66.93) lcomp=75.808 lbc2=-3.320
      xor(66)-(327,66.86) lcomp=75.550 lbc2=-6.641
      xor(66)-(261,66.73) lcomp=75.217 lbc2=-13.281
      xor(66)-(195,66.46) lcomp=74.756 lbc2=-26.562
  guess(17u1)-(178,66.46) lcomp=0.000 lbc2=-26.562
          code-(64,66.46) lcomp=73.932 lbc2=-56.630
          WHT(theta=0.00) lcomp=75.293
           lr_all=1.497 lr_wht=4.170
                      lmaxcomp=81.61
                      ltotcomp=82.53
  for the used code, log_2 bc^2 =-30.067365090154126592900296031961631480
  used code family: old+QP+rnd
  params=[3,1,r][25,15,W][18,6,rnd150926][18,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{532,0.125} with n=2^73.9080
     drop( 7)-(525,66.91) lcomp=74.897 lbc2=-0.830
       sparse-(525,66.91) lcomp=79.666 lbc2=-0.830
      xor(66)-(459,66.82) lcomp=75.944 lbc2=-1.660
      xor(66)-(393,66.63) lcomp=75.658 lbc2=-3.320
      xor(66)-(327,66.26) lcomp=75.250 lbc2=-6.641
      xor(66)-(261,65.53) lcomp=74.617 lbc2=-13.281
      xor(66)-(195,64.06) lcomp=73.556 lbc2=-26.562
  guess(17u1)-(178,64.06) lcomp=0.000 lbc2=-26.562
          code-(64,64.06) lcomp=71.532 lbc2=-56.630
         WHT(theta=15.00) lcomp=75.069
           lr_all=1.497 lr_wht=4.170
                      lmaxcomp=81.16
                      ltotcomp=82.17
  for the used code, log_2 bc^2 =-30.067365090154126592900296031961631480
  used code family: old+QP+rnd
  params=[3,1,r][25,15,W][18,6,rnd150926][18,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the chain from [ZJW16] based on LF(4) with a claimed complexity of $2^{74.709}$:

```
chain for LPN_{532,0.125} with n=2^70.5040
     drop(15)-(517,55.50) lcomp=71.504 lbc2=-0.830
       sparse-(517,55.50) lcomp=68.792 lbc2=-0.830
     lf4(162)-(355,55.43) lcomp=119.022 lbc2=-3.320
```

```
    lf4(162)-(193,55.14) lcomp=118.334 lbc2=-13.281
  guess(13u1)-(180,55.14) lcomp=0.000 lbc2=-13.281
         code-(61,55.14) lcomp=62.631 lbc2=-45.275
         WHT(theta=0.00) lcomp=71.743
          lr_all=0.990 lr_wht=3.807
                    lmaxcomp=120.01
                    ltotcomp=120.71
  for the used code, log_2 bc^2 =-31.994004843189402519390178262046306566
  used code family: old+QP+rnd
  params=[3,1,r][25,15,W][19,4,rnd150926][19,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{532,0.125} with n=2^70.3460
     drop(15)-(517,55.35) lcomp=71.346 lbc2=-0.830
       sparse-(517,55.35) lcomp=68.278 lbc2=-0.830
     lf4(162)-(355,54.80) lcomp=118.706 lbc2=-3.320
     lf4(162)-(193,52.61) lcomp=117.070 lbc2=-13.281
  guess(13u1)-(180,52.61) lcomp=0.000 lbc2=-13.281
         code-(61,52.61) lcomp=60.103 lbc2=-45.275
        WHT(theta=25.00) lcomp=71.675
          lr_all=0.990 lr_wht=3.807
                    lmaxcomp=119.70
                    ltotcomp=120.10
  for the used code, log_2 bc^2 =-31.994004843189402519390178262046306566
  used code family: old+QP+rnd
  params=[3,1,r][25,15,W][19,4,rnd150926][19,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926]
```

Here is the chain we obtain by running our algorithm with precision 0.1, after removing the roundings and adjusting the number of queries:

```
chain for LPN_{532,0.125} with n=2^65.3000
       sparse-(532,65.30) lcomp=78.290 lbc2=-0.830
      xor(61)-(471,68.60) lcomp=77.655 lbc2=-1.660
      xor(67)-(404,69.20) lcomp=78.080 lbc2=-3.320
      xor(68)-(336,69.40) lcomp=78.058 lbc2=-6.641
      xor(68)-(268,69.80) lcomp=78.192 lbc2=-13.281
      xor(69)-(199,69.60) lcomp=77.866 lbc2=-26.562
         code-(67,69.60) lcomp=77.237 lbc2=-62.111
        WHT(theta=17.00) lcomp=78.436
                    lmaxcomp=78.44
                    ltotcomp=81.02
  for the used code, log_2 bc^2 =-35.548346362647631165140054416118030948
  used code family: old+QP+rnd
  params=[3,1,r][25,15,W][19,4,rnd150926][19,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926]
```

A.8.3 Chains for $\mathsf{LPN}_{592,1/8}$

Here is the chain proposed in [GJL14] with a claimed complexity of $2^{88.07}$:

```
chain for LPN_{592,0.125} with n=2^72.7000
       sparse-(592,72.70) lcomp=85.748 lbc2=-0.830
     part(70)-(522,72.46) lcomp=81.909 lbc2=-1.660
     part(70)-(452,72.17) lcomp=81.487 lbc2=-3.320
     part(70)-(382,71.81) lcomp=80.989 lbc2=-6.641
     part(70)-(312,71.32) lcomp=80.384 lbc2=-13.281
     part(70)-(242,70.58) lcomp=79.606 lbc2=-26.562
     part(70)-(172,68.99) lcomp=78.502 lbc2=-53.125
  guess(35u3)-(137,68.99) lcomp=0.000 lbc2=-53.125
         code-(64,68.99) lcomp=76.092 lbc2=-69.942
       WHT(theta=100.00) lcomp=76.063
          lr_all=1.524 lr_wht=12.809
                     lmaxcomp=90.40
                     ltotcomp=90.58
  for the used code, log_2 bc^2 =-16.816717432567051155707898232895271319
  used code family: old+QP+rnd
  params=[5,1,r][25,15,W][25,15,W][25,15,W][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{592,0.125} with n=2^77.3910
       sparse-(592,77.39) lcomp=90.513 lbc2=-0.830
     part(70)-(522,77.38) lcomp=86.600 lbc2=-1.660
     part(70)-(452,77.37) lcomp=86.410 lbc2=-3.320
     part(70)-(382,77.36) lcomp=86.194 lbc2=-6.641
     part(70)-(312,77.36) lcomp=85.942 lbc2=-13.281
     part(70)-(242,77.35) lcomp=85.642 lbc2=-26.562
     part(70)-(172,77.34) lcomp=85.266 lbc2=-53.125
  guess(35u3)-(137,77.34) lcomp=0.000 lbc2=-53.125
         code-(64,77.34) lcomp=84.437 lbc2=-69.942
        WHT(theta=33.00) lcomp=83.344
          lr_all=1.524 lr_wht=12.809
                     lmaxcomp=97.68
                     ltotcomp=97.71
  for the used code, log_2 bc^2 =-16.816717432567051155707898232895271319
  used code family: old+QP+rnd
  params=[5,1,r][25,15,W][25,15,W][25,15,W][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926]
```

Here is the chain from [ZJW16] based on LF1 with a claimed complexity of $2^{84.715}$:

```
chain for LPN_{592,0.125} with n=2^79.5570
     drop( 4)-(588,75.56) lcomp=80.464 lbc2=-0.830
       sparse-(588,75.56) lcomp=88.675 lbc2=-0.830
     part(73)-(515,75.29) lcomp=84.757 lbc2=-1.660
     part(73)-(442,74.96) lcomp=84.297 lbc2=-3.320
     part(73)-(369,74.53) lcomp=83.746 lbc2=-6.641
     part(73)-(296,73.91) lcomp=83.056 lbc2=-13.281
     part(73)-(223,72.82) lcomp=82.124 lbc2=-26.562
  guess(16u1)-(207,72.82) lcomp=0.000 lbc2=-26.562
          code-(72,72.82) lcomp=80.517 lbc2=-62.542
          WHT(theta=0.00) lcomp=83.443
           lr_all=1.366 lr_wht=4.087
                      lmaxcomp=90.04
                      ltotcomp=90.75
  for the used code, log_2 bc^2 =-35.979931993882924904548450029191500372
  used code family: old+QP+rnd
  params=[18,6,rnd150926][25,15,W][13,4,rnd150926][18,5,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{592,0.125} with n=2^79.3610
     drop( 4)-(588,75.36) lcomp=80.268 lbc2=-0.830
       sparse-(588,75.36) lcomp=88.561 lbc2=-0.830
     part(73)-(515,75.05) lcomp=84.561 lbc2=-1.660
     part(73)-(442,74.65) lcomp=84.057 lbc2=-3.320
     part(73)-(369,74.10) lcomp=83.437 lbc2=-6.641
     part(73)-(296,73.19) lcomp=82.623 lbc2=-13.281
     part(73)-(223,70.14) lcomp=81.395 lbc2=-26.562
  guess(16u1)-(207,70.14) lcomp=0.000 lbc2=-26.562
          code-(72,70.14) lcomp=77.829 lbc2=-62.542
         WHT(theta=19.00) lcomp=83.334
           lr_all=1.366 lr_wht=4.087
                      lmaxcomp=89.93
                      ltotcomp=90.62
  for the used code, log_2 bc^2 =-35.979931993882924904548450029191500372
  used code family: old+QP+rnd
  params=[18,6,rnd150926][25,15,W][13,4,rnd150926][18,5,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926]
```

Here is the chain from [ZJW16] based on LF2 with a claimed complexity of $2^{83.843}$:

```
chain for LPN_{592,0.125} with n=2^77.9850
     drop( 4)-(588,73.99) lcomp=78.892 lbc2=-0.830
       sparse-(588,73.98) lcomp=86.931 lbc2=-0.830
      xor(73)-(515,73.97) lcomp=83.185 lbc2=-1.660
      xor(73)-(442,73.94) lcomp=82.978 lbc2=-3.320
      xor(73)-(369,73.88) lcomp=82.728 lbc2=-6.641
      xor(73)-(296,73.76) lcomp=82.407 lbc2=-13.281
      xor(73)-(223,73.52) lcomp=81.969 lbc2=-26.562
  guess(14u1)-(209,73.52) lcomp=0.000 lbc2=-26.562
          code-(72,73.52) lcomp=81.227 lbc2=-63.159
          WHT(theta=0.00) lcomp=83.496
          lr_all=1.112 lr_wht=3.907
                     lmaxcomp=88.52
                     ltotcomp=89.46
  for the used code, log_2 bc^2 =-36.596897467910004630929717148293030280
  used code family: old+QP+rnd
  params=[25,15,W][13,4,rnd150926][19,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{592,0.125} with n=2^77.8980
     drop( 4)-(588,73.90) lcomp=78.805 lbc2=-0.830
       sparse-(588,73.90) lcomp=86.873 lbc2=-0.830
      xor(73)-(515,73.80) lcomp=83.098 lbc2=-1.660
      xor(73)-(442,73.59) lcomp=82.804 lbc2=-3.320
      xor(73)-(369,73.18) lcomp=82.380 lbc2=-6.641
      xor(73)-(296,72.37) lcomp=81.711 lbc2=-13.281
      xor(73)-(223,70.74) lcomp=80.577 lbc2=-26.562
  guess(14u1)-(209,70.74) lcomp=0.000 lbc2=-26.562
          code-(72,70.74) lcomp=78.443 lbc2=-63.159
         WHT(theta=30.00) lcomp=83.351
          lr_all=1.112 lr_wht=3.907
                     lmaxcomp=88.37
                     ltotcomp=89.32
  for the used code, log_2 bc^2 =-36.596897467910004630929717148293030280
  used code family: old+QP+rnd
  params=[25,15,W][13,4,rnd150926][19,5,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926]
```

Here is the chain from [ZJW16] based on LF(4) with a claimed complexity of $2^{81.963}$:

```
chain for LPN_{592,0.125} with n=2^78.5130
     drop(18)-(574,60.51) lcomp=79.513 lbc2=-0.830
       sparse-(574,60.51) lcomp=73.677 lbc2=-0.830
     lf4(177)-(397,60.47) lcomp=129.191 lbc2=-3.320
     lf4(177)-(220,60.28) lcomp=128.567 lbc2=-13.281
  guess(16u1)-(204,60.28) lcomp=0.000 lbc2=-13.281
          code-(68,60.28) lcomp=67.956 lbc2=-50.044
          WHT(theta=0.00) lcomp=79.025
          lr_all=1.366 lr_wht=4.087
                     lmaxcomp=130.56
                     ltotcomp=131.28
  for the used code, log_2 bc^2 =-36.762390670153894058017705531513633529
  used code family: old+QP+rnd
  params=[19,5,rnd150926][8,2,iGop][25,15,W][19,4,rnd150926][19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926]
```

Here is the corrected one to reach $\theta = 33\%$:

```
chain for LPN_{592,0.125} with n=2^78.3410
     drop(18)-(574,60.34) lcomp=79.341 lbc2=-0.830
       sparse-(574,60.34) lcomp=73.561 lbc2=-0.830
     lf4(177)-(397,59.78) lcomp=128.847 lbc2=-3.320
     lf4(177)-(220,57.53) lcomp=127.191 lbc2=-13.281
  guess(16u1)-(204,57.53) lcomp=0.000 lbc2=-13.281
          code-(68,57.53) lcomp=65.204 lbc2=-50.044
         WHT(theta=33.00) lcomp=78.959
           lr_all=1.366 lr_wht=4.087
                    lmaxcomp=130.21
                    ltotcomp=130.61
  for the used code, log_2 bc^2 =-36.762390670153894058017705531513633529
  used code family: old+QP+rnd
  params=[19,5,rnd150926][8,2,iGop][25,15,W][19,4,rnd150926][19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926]
```

Here is the chain we obtain by running our algorithm with precision 0.1, after removing the roundings and adjusting the number of queries:

```
chain for LPN_{592,0.125} with n=2^71.6910
       sparse-(592,71.69) lcomp=84.735 lbc2=-0.830
      xor(67)-(525,75.38) lcomp=84.591 lbc2=-1.660
      xor(74)-(451,75.76) lcomp=84.800 lbc2=-3.320
      xor(75)-(376,75.53) lcomp=84.581 lbc2=-6.641
      xor(74)-(302,76.06) lcomp=84.611 lbc2=-13.281
      xor(75)-(227,76.11) lcomp=84.350 lbc2=-26.562
          code-(73,76.11) lcomp=83.939 lbc2=-68.504
         WHT(theta=22.00) lcomp=84.751
                    lmaxcomp=84.80
                    ltotcomp=87.57
  for the used code, log_2 bc^2 =-41.941549516941019516761621630448187382
  used code family: old+QP+rnd
  params=[18,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,
      rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926][19,6,rnd150926
      ][19,6,rnd150926][19,6,rnd150926][19,7,rnd150926]
```

Appendix B

Curriculum Vitae

Name: Sonia Mihaela Bogos
E-mail : `soniam.bogos@gmail.com`
Citizenship : Romanian

Education

2012-2017	**PhD in Computer, Communication and Information Sciences** **Area:** Cryptography Supervised by Prof. Serge Vaudenay Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
2010-2012	**Master in Computer Science** EPFL, Switzerland
2007-2010	**Bachelor in Computer Science** Faculty of Computer Science, Alexandru Ioan Cuza University (UAIC), Iasi, Romania
2009-2010	**Erasmus student** Ecole normale supérieure de Cachan, France Laboratoire Spécification & Vérification

Academic Honors

2016	**Anita Borg Scholarship**
2014	**Outstanding Teaching Assistant Award** EPFL, Switzerland
2012	**Fellowship** from the doctoral school in Computer and Communication Science EPFL, Switzerland
2010-2012	**Excellency Scholarship** EPFL, Switzerland
2009,2008	**Faculty Annual Diploma for Academic Performance - 1st prize** UAIC, Romania

Teaching

– **Supervised nine students** for **research** and **implementation-oriented projects**.

– **Teaching Assistant** for Cryptography and Security, Advanced Cryptography, Student Seminar: Security Protocols and Applications, Initiation à la programmation en C++, Initiation à la programmation en Java, Programmation Orientée Système (C/Unix/Perl).

Languages

Romanian (native), *English* (fluent), *French* (good)

Technical Skills

- **Programming Languages :** Python, C, C++, Java