Generalized Syndrome Decoding Problem and its Application to Post-Quantum Cryptography

Par Simona ETINSKI
Thèse de doctorat de Informatique

Dirigée par Frédéric MAGNIEZ

Présentée et soutenue publiquement le 28 Juin 2023

Devant un jury composé de

Rapporteurs:
Daniel AUGOT, DR, INRIA Saclay, École polytechnique
Elena KIRSHANOVA, Senior research scientist, Technology Innovation Institute

Examinateurs:
Sophie LAPLANTE, PU, CNRS, Université Paris Cité
Nicolas RESCH, Assistant professor, University of Amsterdam
Nicolas SENDRIER, DR, INRIA Paris

Directeurde thèse:
Frédéric MAGNIEZ, DR, CNRS, Université Paris Cité

Membre invité, co-encadrant:
André CHAILLOUX, CR, INRIA Paris

The research was carried out in the COSMIQ team at INRIA Paris.

Acknowledgments

The list of people I would like to thank is rather extensive. Keeping these acknowledgments reasonably short contradicts my desire to thank everyone who helped me to get properly educated, sufficiently encouraged, and well motivated to start, carry on, and eventually finish my Ph.D. studies. Therefore, I decided to keep the acknowledgments fairly succinct and express my gratitude to all people not explicitly mentioned in person.

To start with, I thank my adviser, Frédéric Magniez, for encouraging me to start my Ph.D. studies in theoretical computer science and for all the support throughout the rest of my studies. I then thank my co-advisor, André Chailloux, for helping me start with research in cryptography and guiding me to the successful end of my thesis. The successful completion of my studies, of course, would not be possible without the thesis defense. Therefore, I thank all jury members for accepting to be part of my defense. I thank Sophie Laplante and Nicolas Sendrier for being my examiners. I am especially grateful to my reviewers, Elena Kirshanova and Daniel Augot, and the examiner Nicolas Resch for providing me with valuable insights and constructive criticism on how to improve the manuscript.

I would now like to thank the institutions that provided me with research funding and the research groups that hosted me during my studies. I start with the FSMP and the Paris Cité University, which funded my studies. I am especially thankful to the MathInParis co-fund and Ariela Braini, who supported me both administratively and morally during my Ph.D. I am grateful to INRIA Paris and the COSMIQ team for hosting me for the last three years, as well as to IRIF and its Algorithms and Complexity group that hosted me for a short period at the beginning of my studies and occasionally during the rest of my studies (more precisely, whenever I found a good enough reason to spend my time there). I would like to add to this list CWI and its Algorithms and Complexity group, which hosted me during my research visit in September 2022, as well as the Cryptology group that hosted me for the last three months of my thesis and in which I started my postdoc. I thank the members of these groups for all the time we spent together at the office, at seminars and conferences, during lunches and coffee breaks, running sessions, escape rooms, foosball matches, and barbeques. Though only some of these activities were crucial for my development as a researcher, the others were there to boost my morale and thus contributed to this thesis being a successful project.

I would now like to devote a few sentences to all my teachers at earlier stages of my education and professors later on that encouraged my endeavors to broaden my knowledge and make my first step into science. I feel that these people are never credited enough for the tremendous work they do to prepare us for what we become after, so I use this opportunity to thank them. I hope the situation in my country will improve so that these people get the recognition they deserve and that many new generations will have a chance to learn from them in an environment that nurtures their talents.

Finally, I would like to express my gratitude to my friends and family who are to be credited for all my achievements, my thesis included. I would start by thanking Danica Despotović for her friendship and kindness that comforted me throughout my studies. I would then like to thank my parents, my grandma, and my sister for their love and support throughout my entire educational path that eventually led to this thesis. I cannot ever thank enough, my husband, for his love and support. Without his encouragement, I never would have even started these studies and found enough strength and motivation to bring them to a successful end.

Problème de décodage de syndrome généralisé et son application à la cryptographie post-quantique

Dans cette thèse, nous nous concentrons sur le problème du décodage du syndrome (SDP), sa généralisation, la cryptanalyse et son application à la conception de schémas de signature. Nous introduisons un nouveau problème, que nous appelons le problème de décodage de syndrome généralisé. Dans la partie cryptanalytique de la thèse, nous nous concentrons sur la cryptanalyse classique et quantique du problème de décodage de syndrome généralisé en utilisant les algorithmes de décodage par ensemble d'information. Plus précisément, nous calculons le temps d'exécution de trois algorithmes de (classiques) différents de ce type, que nous appelons les algorithmes de Prange, de Stern/Dumer et de Wagner. Les trois algorithmes sont adaptés pour résoudre des versions spécifiques du problème généralisé qui utilise le poids de Hamming, pris comme référence, et le poids de Lee, pris comme alternative au poids de Hamming. Nous comparons ensuite les temps d'exécution obtenus avec le temps d'exécution de l'algorithme hybride classique-quantique, obtenu en introduisant la recherche de Grover et l'amplification d'amplitude dans l'étape appropriée de l'algorithme de Wagner. Dans la partie de l'article consacrée à la conception de protocoles, nous modifions le protocole d'identification de Stern et le schéma de signature correspondant pour l'adapter au problème de décodage du syndrome généralisé nouvellement introduit. Nous proposons ensuite différentes méthodes pour optimiser l'efficacité du système et fournissons des résultats numériques qui comparent l'efficacité de la construction originale et de notre nouveau système. Le résultat de ce travail est une analyse de la variante nouvellement introduite du problème de décodage de syndrome qui fournit une estimation de la complexité asymptotique du problème, ainsi que de la sécurité concrète du schéma basé sur ce problème. Les résultats indiquent que le choix approprié d'une fonction de poids introduit une version plus difficile du problème de décodage de syndrome et produit donc des protocoles plus efficaces basés sur ce problème.

mots clés: problème du décodage du syndrome, métrique de Lee, décodage par ensemble d'information, schéma de signature de Stern

Generalized Syndrome Decoding Problem and its Application to Post-Quantum Cryptography

In this thesis, we focus on the syndrome decoding problem (SDP), its generalization, cryptanalysis, and its application to digital signature scheme designs. We introduce a new problem, which we refer to as the generalized syndrome decoding problem. In the cryptanalytic part of the thesis, we then focus on the classical and quantum cryptanalysis of the generalized syndrome decoding problem using the information set decoding framework. More precisely, we calculate the running time of three different (classical) information set decoding algorithms, which we refer to as Prange's, Stern's/Dumer's, and Wagner's algorithms. The three algorithms are adapted to solve specific versions of the generalized problem which are given over the Hamming weight, taken as a baseline, and the Lee weight, taken as an alternative to the most commonly used Hamming weight. We then compare the obtained running times with the running time of the hybrid classical-quantum algorithm, obtained by introducing the Grover search and the amplitude amplification in the appropriate step of Wagner's algorithm. In the protocol design part of the paper, we modify Stern's identification protocol, and the corresponding signature scheme, to the newly introduced generalized syndrome decoding problem. To keep the zero-knowledge property of the scheme, we eventually replace the syndrome decoding problem with the permuted kernel one (PKP), for which we show that the average-case SDP reduces to average-case PKP. We then suggest different methods for optimizing the efficiency of the scheme and then provide numerical results that compare the efficiency of the original construction and our newly introduced scheme. The outcome of this work is an analysis of the newly introduced variant of the syndrome decoding problem that estimates the asymptotic complexity of the problem, as well as the concrete security of the scheme based on this problem. The results indicate that the proper choice of a weight function introduces a harder version of the syndrome decoding problem and thus yields more efficient protocols based upon it.

key words: syndrome decoding problem, Lee metric, information set decoding, Stern's signature scheme

Problème de décodage de syndrome généralisé et son application à la cryptographie post-quantique

Le thème central de cette recherche est le problème du décodage du syndrome. Introduit à l'origine dans la théorie du codage, ce problème a trouvé un large éventail d'applications en cryptographie, et plus particulièrement dans un sous-domaine connu sous le nom de cryptographie post-quantique. Dans cette thèse, nous nous concentrons sur la généralisation de ce problème. Présentons donc le problème. Prenons $n, k \in \mathbb{N}$ et soit $\mathrm{wt}(\cdot)$ le poids de Hamming. Le problème du décodage de syndrome décisionnel est défini comme suit.

> **Problem 0.0.1** (Le problème du décodage de syndrome décisionnel). *Étant donné* $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$, $\mathbf{s} \in \mathbb{F}_2^{n-k}$, *et* $w \in \mathbb{N}$, *déterminer s'il existe* $\mathbf{e} \in \mathbb{F}_2^n$ *satisfaisant* $\mathbf{s} = \mathbf{e}\mathbf{H}^T$ *et* $\mathrm{wt}(\mathbf{e}) = w$.

Alors que la méthode originale de décodage de syndrome a été introduite pour fournir un moyen efficace de décoder les codes linéaires, le problème de décodage de syndrome, dérivé de cette méthode, s'avère être NP-complet. Plus important encore, pour des paramètres bien choisis, ce problème s'avère difficile même pour les meilleurs algorithmes qui s'y attaquent. En tant que tel, il est considéré comme une bonne base potentielle pour différents protocoles cryptographiques.

Néanmoins, en pratique, nous observons que les protocoles basés sur ce problème manquent souvent d'efficacité. Le but de cette thèse est donc d'augmenter la difficulté du problème de décodage de syndrome et, par conséquent, d'améliorer l'efficacité des protocoles basés sur ce problème. La thèse est structurée comme suit.

Chapitre 1 Nous avons donné une brève introduction et une vue d'ensemble des concepts pertinents dans le contexte de cette thèse. Nous commençons le chapitre par une introduction qui explique la motivation de ce travail et donne une vue d'ensemble de cette thèse. Dans la même section, nous présentons les conventions de notation utilisées tout au long de la thèse. Nous introduisons ensuite les définitions pertinentes de la théorie du codage, des algorithmes et de la complexité, et de la cryptographie.

Chapitre 2 Dans ce chapitre, nous généralisons le problème du décodage de syndrome au-delà de l'alphabet binaire et du poids de Hamming, le cadre le plus courant dans la théorie du codage et la cryptographie. Pour ce faire, nous redéfinissons le problème en introduisant les fonctions de poids élémentaires. Nous obtenons ainsi une nouvelle version généralisée du problème définie sur un alphabet arbitraire de nombres premiers et un poids élémentaire arbitraire. Plus précisément, le problème est maintenant défini comme suit. Prenons q un nombre premier, et $\mathrm{wt}_M(\cdot)$ une fonction de poids élémentaire. Le problème du décodage de syndrome généralisée est défini comme suit.

Problem 0.0.2 (Le problème du décodage de syndrome généralisée). *Étant donné* $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, *et* $w \in \mathbb{N}$, *déterminer s'il existe* $\mathbf{e} \in \mathbb{F}_q^n$ *satisfaisant* $\mathbf{s} = \mathbf{e}\mathbf{H}^T$ *et* $\mathrm{wt}_M(\mathbf{e}) = w$.

Nous montrons ensuite que la version moyenne du problème de décodage du syndrome généralisé est réductible en temps polynomial à la version moyenne du problème du noyau permuté. L'ingrédient crucial de cette réduction est la capacité de calculer la surface de la sphère dans un espace vectoriel doté d'une fonction de poids élémentaire arbitraire. Pour calculer la surface de la sphère, nous généralisons la méthode précédemment connue pour le poids de Lee uniquement. La généralisation de cette méthode a servi de base pour démontrer la réduction susmentionnée et déterminer la complexité asymptotique du problème du point de vue de la cryptanalyse, présentée au chapitre 3.

Chapitre 3 Nous adaptons le cadre de décodage par ensemble d'informations, qui sont les meilleures attaques génériques contre le problème de décodage du syndrome en poids de Hamming, à notre cadre plus général. Le cadre comprend les quatre étapes suivantes.

1. *étape de permutation* : L'algorithme permute les colonnes de la matrice de contrôle de parité $\mathbf{H}$.

2. *étape de décomposition* : L'algorithme effectue une élimination Gaussienne partielle sur $\mathbf{H}$ pour fournir une sous-instance du problème.

3. *mSDP étape* : L'algorithme trouve plusieurs solutions à la sous-instance de décodage du syndrome.

4. *étape de test* : L'algorithme vérifie si l'une des solutions à la sous-instance permet de résoudre le problème initial.

Nous utilisons ce cadre pour déterminer la complexité asymptotique du problème de décodage du syndrome généralisé dans des contextes classiques et quantiques. Plus précisément, nous analysons quatre algorithmes classiques, appelés algorithme de Prange[Pra62], algorithme de Lee-Brickel[LB88], algorithme de Stern/Dumer[Ste88] et algorithme de Wagner[Wag02]. Nous montrons que ces trois premiers algorithmes peuvent être considérés comme des variantes différentes de l'algorithme de Wagner. À partir de l'algorithme de Wagner, nous dérivons ensuite un algorithme hybride classique-quantique qui combine l'approche classique de l'algorithme de Wagner avec la recherche de Grover et l'amplification d'amplitude pour obtenir une accélération quadratique.

À la fin du chapitre, nous présentons les résultats numériques sur le temps d'exécution asymptotique de ces algorithmes de décodage en poids de Hamming et en poids de Lee. Nous illustrons certains de ces résultats dans les figures suivantes, où $R := \frac{k}{n}$, $\omega = \frac{w}{n}$, et α est l'exposant du temps d'exécution de l'algorithme (exprimé en $\log_2$).
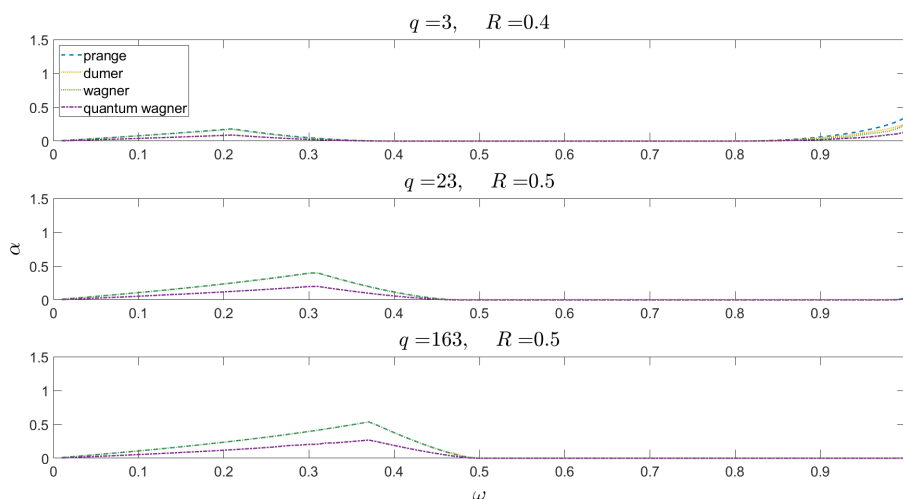

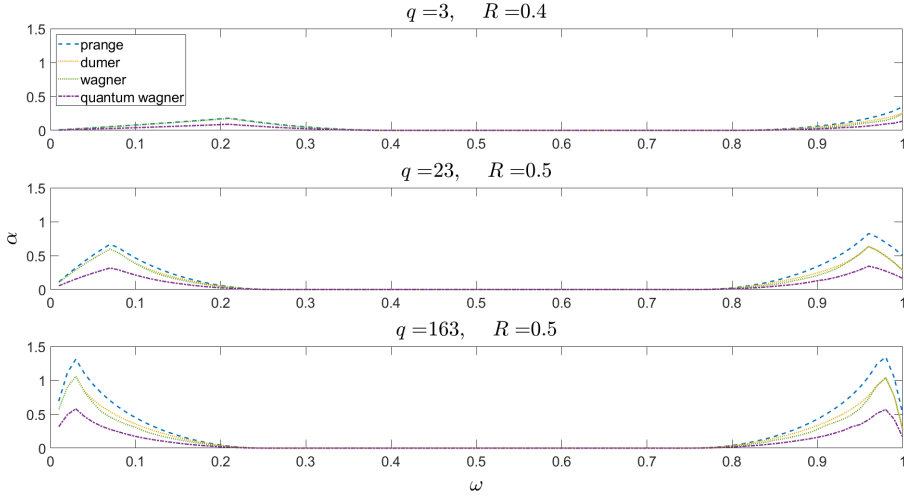Figure 1: Complexité de l'A-SDP de Hamming : quatre algorithmes de l'ISD

Figure 2: Complexité de l'A-SDP de Lee : quatre algorithmes de l'ISD

Les résultats indiquent que le problème dans le cadre du poids de Lee peut être considéré comme une version plus cohérente du problème de décodage du syndrome lorsque la taille de l'alphabet augmente. De plus, les résultats numériques sur les instances les plus difficiles du problème, présentés dans le tableau suivant, indiquent que le problème dans le cadre du poids de Lee est plus difficile que dans le cadre du poids de Hamming pour toutes les tailles d'alphabet supérieures à 3.

Pour q, R et ω donnés, nous exprimons alors la dureté relative d'une instance par deux coefficients, α et $\hat{\alpha}$, définis comme suit:

$$\alpha := \lim_{n \to \infty} \frac{\log_2 t^W(n)}{n}, \quad \hat{\alpha} := \lim_{n \to \infty} \frac{\log_q t^W(n)}{n},$$

où $t^W(n)$ est le temps d'exécution de l'algorithme de Wagner (quantique). Le temps d'exécution asymptotique est obtenu pour le choix optimal des paramètres de l'algorithme de Wagner qui donnent le temps d'exécution le plus court de l'algorithme de Wagner (quantique).

Hamming weight, $\mathrm{wt}_H(\cdot)$								
q	Wagner's algorithm				quantum Wagner's algorithm			
	R	ω	α	$\hat{\alpha}$	R	ω	α	$\hat{\alpha}$
3	0.370	1.000	0.269	0.170	0.369	1.000	0.148	0.093
13	0.456	0.311	0.356	0.096	0.453	0.314	0.180	0.049
43	0.459	0.368	0.459	0.085	0.459	0.368	0.230	0.042
163	0.463	0.405	0.541	0.074	0.463	0.405	0.271	0.037
643	0.468	0.427	0.602	0.065	0.475	0.420	0.316	0.034

Table 1: Instances les plus difficiles de A-SDP avec le poids de Hamming

Lee weight, $\mathrm{wt}_L(\cdot)$								
q	Wagner's algorithm				quantum Wagner's algorithm			
	R	ω	α	$\hat{\alpha}$	R	ω	α	$\hat{\alpha}$
3	0.370	1.000	0.269	0.170	0.369	1.000	0.148	0.093
13	0.475	0.955	0.522	0.141	0.501	0.962	0.283	0.076
43	0.459	0.955	0.794	0.146	0.467	0.957	0.429	0.079
163	0.445	0.968	1.117	0.152	0.462	0.971	0.607	0.083
643	0.437	0.980	1.455	0.156	0.455	0.982	0.794	0.085

Table 2: Instances les plus difficiles de A-SDP avec le poids de Lee

La raison pour laquelle nous introduisons deux coefficients pour exprimer le temps d'exécution asymptotique, à savoir α et $\hat{\alpha}$, est la suivante. La manière courante d'exprimer le temps d'exécution asymptotique d'un algorithme est sous la forme $t(n) = 2^{\alpha(n)}$, où $n \to \infty$, et $\alpha(n)$ est déterminé par l'algorithme. Néanmoins, il apparaît que cette représentation ne reflète pas exactement la difficulté du problème du syndrome dans le cadre généralisé auquel nous nous intéressons. Nous introduisons donc un autre coefficient du temps d'exécution asymptotique, $\hat{\alpha}(n)$, obtenu lorsque $t(n)$ est donné comme $t(n) = q^{\hat{\alpha}(n)}$ et $n \to \infty$. Ce coefficient semble refléter plus précisément la difficulté du problème car il est cohérent avec les observations précédentes concernant le problème de décodage du syndrome sur le poids de Hamming pour des tailles d'alphabet $q > 2$ (voir, par exemple, [Pet10]).

Nous observons que, tant dans le cas du poids de Hamming que dans le cas du poids de Lee, le taux de codage des instances les plus difficiles du problème

se situe dans l'intervalle $[0.35, 0.5]$. Ceci était déjà prévu par les graphiques des résultats numériques obtenus précédemment. Nous observons également que, dans le cadre du poids de Hamming et pour $q > 3$, pour la valeur de R donnant les instances les plus difficiles du problème, le poids normalisé atteint ce que l'on appelle la limite de *Gilbert-Varshamov*, définie au chapitre 1. Les instances les plus difficiles du problème sur le poids de Hamming pour $q = 3$ ou sur le poids de Lee pour tout q observé, en revanche, sont obtenues dans le régime de poids élevé. A notre connaissance, ce régime de paramètres ne correspond à aucune borne connue et il est généralement omis dans l'analyse numérique bien qu'il produise effectivement les instances les plus difficiles du problème. Nous soulignons donc l'importance de prendre en compte le régime de poids élevé lors de l'analyse du problème de décodage de syndrome généralisé.

Nous observons ensuite que les deux coefficients (à savoir α et $\hat{\alpha}$), dans le cadre classique comme dans le cadre quantique, sont plus élevés dans le cas du poids de Lee. Là encore, les graphiques des résultats numériques obtenus précédemment l'indiquaient déjà. Cela suggère en outre que les instances A-SDP les plus difficiles sur le poids de Lee sont plus difficiles que les instances A-SDP les plus difficiles sur le poids de Hamming. Observons alors uniquement les valeurs de α dans les deux cas. Nous voyons alors que α augmente avec l'augmentation de q dans tous les cas. Cela donne l'impression que la complexité du problème augmente avec l'augmentation de la taille de l'alphabet. Cependant, dans les études précédentes de ce problème pour le poids de Hamming, il a été montré qu'en fait, ce n'est pas le cas. Observons alors $\hat{\alpha}$. Dans le cas du poids de Hamming, à la fois pour le cadre classique et quantique, il y a une chute soudaine de $\hat{\alpha}$ lorsque l'on passe de $q = 3$ à $q = 13$, et le coefficient continue à diminuer lorsque la taille de l'alphabet augmente. Dans le cas du poids de Lee, en revanche, la chute est moins importante et, en fait, le coefficient recommence à augmenter pour des tailles d'alphabet comprises entre $q = 43$ et $q = 163$. La question ouverte est donc de savoir quel serait le cas limite. Peut-on espérer que pour des tailles d'alphabet vraiment élevées, la complexité sera plus élevée que pour $q = 3$?

En remplaçant le problème du décodage du syndrome dans le poids de Lee par le problème du décodage du syndrome dans le poids de Hamming, nous améliorons l'efficacité du système, mais pas suffisamment pour qu'il devienne applicable dans la pratique. Il est donc optimisé davantage en utilisant des générateurs pseudo-aléatoires, ce qui a effectivement augmenté l'efficacité du schéma de manière significative, mais qui était susceptible de faire l'objet

d'une nouvelle attaque. Nous expliquons ensuite la vulnérabilité de cette méthode d'optimisation et proposons une méthode pour atténuer cette attaque. Nous terminons le chapitre en présentant les résultats numériques des tailles de signature de la signature numérique que nous avons conçue. Nous présentons ces résultats ici.

Tailles de signature (en kB)				
q	Non-optimized scheme		Optimized scheme	
	wt_H	wt_L	wt_H	wt_L
2	253.05	253.05	26.21	26.21
3	116.54	116.54	21.81	21.81
5	138.54	95.48	27.62	21.41
7	126.47	90.94	28.29	22.71
13	113.23	79.27	29.38	23.29

Table 3: Taille des signatures des schémas optimisés et non optimisés

Nous observons ici que la diminution de la taille de la signature obtenue par l'introduction de générateurs pseudo-aléatoires est nettement plus importante que celle introduite par la modification de la fonction de poids. Néanmoins, nous remarquons que, comme nous l'avons montré dans la sous-section précédente, l'utilisation de générateurs pseudo-aléatoires peut potentiellement conduire à une fuite de sécurité dans certains contextes. En revanche, le remplacement du poids de Hamming par le poids de Lee n'entraîne aucun coût en termes de sécurité. Compte tenu de tous ces éléments, le remplacement de la fonction de poids peut être considéré comme une technique d'optimisation non autonome qui, en combinaison avec d'autres techniques, peut conduire à une réduction supplémentaire de la taille des signatures et à des schémas potentiellement applicables dans le monde réel. Une autre observation concerne le choix de la taille de l'alphabet. En effet, nous observons que la plus petite taille de signature est obtenue pour $q = 5$, une taille d'alphabet qui n'est pas couramment utilisée dans la conception de protocoles cryptographiques. Il pourrait donc s'agir d'un choix intéressant pour la conception de nouveaux protocoles.

Directions futures Dans nos travaux ultérieurs, nous aimerions aborder les questions suivantes. La première, et la plus naturelle, serait de savoir

si nous pouvons combiner d'autres techniques d'optimisation plus récentes, telles que *MPC-in-the-head*, pour obtenir des signatures numériques plus efficaces basées sur le problème du décodage de syndrome généralisé. Du côté de la cryptanalyse, nous serions intéressés de savoir s'il serait possible d'utiliser des techniques de Décodage par Ensemble d'Information plus avancées, comme celles basées sur les représentations ou la recherche du plus proche voisin, pour obtenir une meilleure attaque sur le problème généralisé, ou en particulier, sa version en poids de Lee. D'autres algorithmes de recherche quantique seraient-ils plus performants que notre approche basée sur la recherche de Grover et l'amplification d'amplitude?

CONTENTS

CHAPTER ⬤1

PRELIMINARIES

We start this chapter with a brief introduction that explains the main motivation for the research done on the topic of the syndrome decoding problem and gives a high-level overview of this thesis. In the same section, we provide the notational conventions used throughout the thesis. We then introduce the basic concepts in the areas relevant to this thesis, namely, the basic concepts

in coding theory, algorithms and complexity, and cryptography.

In the section on coding theory, we introduce linear codes and related concepts which provide the starting point for the research presented in this thesis. We primarily rely on the standard coding theory literature, such as [Ber68], and [RU08], and we say a special thank you to Mary Wootters whose lecture notes made some concepts more accessible to us and thus better explained in this thesis. In this section, we assume that a reader is familiar with fundamental concepts in abstract algebra and linear algebra, such as notions of a finite field and a vector space. We also assume some familiarity with properties of norms and metrics, but the basic intuition should be clear from the context even without strong familiarity with the topic.

The section on computational models and algorithms introduces the two computational models we are primarily interested in this thesis, namely, the classical circuit model and the quantum circuit model, and the corresponding algorithms. In this section, we focused on elaborating on the quantum computing part, as the one we considered to be less common in the standard computer science curriculum. We thus primarily rely on the standard quantum computing literature such as [NC16], as well as on the lecture notes and the course on quantum computing COMS 4281, given by Henry Yuen. For readers already familiar with quantum computing, this part can be rather elementary and considered redundant, so we advise these readers to go quickly through it or skip it entirely. For readers less familiar with quantum computing, it can be useful to draw the analogs between the two computational models.

The section on computational complexity introduces basic complexity classes and fundamental concepts such as reductions that are relevant in the context of this thesis. For that, we rely on the standard complexity theory literature such as [AB09]. We finish this chapter with a section on cryptography, where we introduce the basic concepts of cryptanalysis and protocol design that are relevant in the context of our research. For that, we rely on the basic cryptographic literature, such as [KL14], as well as on some concepts already introduced in the part on complexity in the previous section. Though most of the cryptographic preliminaries are given in this section, some of them are placed in the preliminary part of the chapters where we consider them relevant.

1.1 Introduction

The central topic of this research is the *syndrome decoding problem*. Originally introduced in coding theory, this problem has found a wide range of applications in cryptography, and more specifically, in its subfield known as the *post-quantum cryptography*. In this thesis, we focus on the generalization of this problem. While the original syndrome decoding method was introduced to provide an efficient way for decoding linear codes, our goal is to provide a problem that cannot be solved efficiently and thus be used as a basis of cryptographic protocols. We thus aim to make this problem as difficult as possible and potentially improve the efficiency of the protocols based upon this problem.

The thesis is structured as follows. In the second chapter, we introduce the notion of *generalized syndrome decoding problem* and we focus on the analysis of its complexity from the complexity theory point of view. In the third chapter, we observe this problem from the *cryptanalytic* perspective. Namely, we analyze the state-of-the-art attacks against this problem, known under the name of *information set decoding* algorithms. In Chapter 4, we use the results from the analysis of the problem to construct a digital signature scheme based on the generalized version of the syndrome decoding problem. We conclude the thesis with a summary of obtained results, open problems, and future directions.

1.1.1 Notation

Throughout the rest of this thesis, we use the following notation. We denote by $\mathbb{N}$ the set of natural numbers. We take $\mathbb{Z}^+$ to be the set of non-negative integers, $\mathbb{R}^+$ to be the set of non-negative real numbers, and $\mathbb{F}_q$ to be a finite field of size q, where q is a prime number. We denote by a small, non-bold letter a scalar in any of these sets and by a bold small letter a vector in a vector space defined over these sets. Via a bold capital letter, we denote a matrix. For example, we have that $x \in \mathbb{F}_q$ is a scalar in the finite field $\mathbb{F}_q$; $\mathbf{x} \in \mathbb{F}_q^n$, $n \in \mathbb{N}$, is a vector in n-dimensional vector space, where n is some natural number; and $\mathbf{X} \in \mathbb{F}_q^{m \times l}$, $m, l \in \mathbb{N}$, is an m times l dimensional matrix taking values from $\mathbb{F}_q$, where m, l are some natural numbers. We then denote by $\xleftarrow{\$}$ sam-

pling a scalar, vector, or matrix from a certain distribution of elements. If not stated differently, we assume that the underlying distribution is uniform over the given set. For $n \in \mathbb{N}, r \in \mathbb{R}^+$, we use the following notation convention $[n] := \{0, 1, \ldots, n-1\}$ and $[r] := \{i \in \mathbb{R}^+ \mid i \leq r\}$. We denote by $\{x_i\}_{i \in [n]}$ the set of elements $\{x_0, x_1, \ldots, x_{n-1}\}$. Similarly, we denote by $(x_i)_{i \in [n]}$ a vector $(x_0, x_1, \ldots, x_{n-1})$. Finally, we use the asymptotic notation to express the upper/lower bounds. Namely, we use $g(n) = O(f(n))$ to express that the function $g(\cdot)$ is upper bounded by the function $f(\cdot)$. Equivalently, we use $f(n) = \Omega(g(n))$ to express that the function $f(\cdot)$ is lower bounded by the function $g(\cdot)$.

1.2 Coding theory

Coding theory is the field devoted to designing protocols that enable secure communication over public, potentially noisy channels. The underlying assumption is that the communication needs to be protected against random noise occurring during transmission due to the imperfections of a transmission channel.

$$S \xrightarrow{\quad \mathbf{m} \quad} \rightsquigarrow \xrightarrow{\quad \tilde{\mathbf{m}} \quad} R$$

Figure 1.1: Coding theory: basic setting

In the basic setting, depicted in Figure 1.1, a sender S sends a message $\mathbf{m}$, i.e. a string of characters, to a receiver R. The message is sent through a public communication channel (for example, an optical cable), depicted by a wavy curve in Figure 1.1. Given the underlying imperfections of physical communication channels, we assume that certain errors will occur during message transmission. Subsequently, certain message characters are altered and the message obtained by the receiver, denoted as $\tilde{\mathbf{m}}$, is different from the original message. The goal of coding theory, therefore, is to enable the detection and correction of errors that occurred during message transmission and recover the original message on the receiver's side.

$$S \xrightarrow{\quad \mathbf{m} \quad} \boxed{\mathrm{Enc}(\cdot)} \xrightarrow{\quad \mathbf{c} \quad} \rightsquigarrow \xrightarrow{\quad \tilde{\mathbf{c}} \quad} \boxed{\mathrm{Dec}(\cdot)} \xrightarrow{\quad \tilde{\mathbf{m}} \quad} R$$
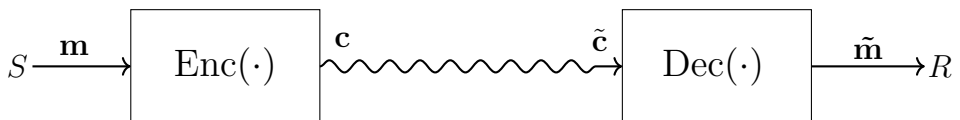
Figure 1.2: Coding theory: basic setting

The most common approach to this problem is to design a protocol that enables *message encoding* on the sender's side and *message decoding* on the receiver's side. A sender encodes its message using a function, denoted as $\mathrm{Enc}(\cdot)$ in Figure 1.2, that adds redundancy to the message. The redundancy is then used to protect the data during transmission and recover the original message on the receiver's side. The decoding function, denoted as $\mathrm{Dec}(\cdot)$ in Figure 1.2, is applied on the receiver's side to detect and correct errors that occurred during transmission. The receiver then recovers the original message by removing the redundancy.

1.2.1 Message encoding

In this thesis, we focus on the analysis of linear codes. We thus assume that each message, $\mathbf{m}$, introduced in the previous subsection, can be represented as a vector in a finite-dimensional vector space $\mathbb{F}_q^k$, where q is a prime number[1], $\mathbb{F}_q$ denotes the underlying finite field, also known as the *alphabet*, and $k \in \mathbb{N}$ is message length, also known as the *code dimension*. This representation is depicted in Figure 1.3, where each message is a vector (red dot) in a one-dimensional vector space. We remark here that we will consider only finite fields that correspond to integers modulo prime numbers. Therefore, the elements of the field of size q will be denoted as integers in $\{0, 1, \ldots, q-1\}$, and all the arithmetic operations will be modulo q.

Figure 1.3: Message space for $k = 1, \ q = 3$

In this setting, the encoding $\mathrm{Enc}(\cdot)$ is defined as a linear map $\mathrm{Enc}(\cdot) : \mathbb{F}_q^k \to \mathbb{F}_q^n$ that takes as an entry a vector from a message space and maps into a vector space with the same underlying alphabet but of larger dimension, $n \in \mathbb{N}$, also known as the *code length*. The encoding is depicted in Figure 1.4, where its domain (i.e. message space) is represented on the left side and its co-domain is represented on the right side.
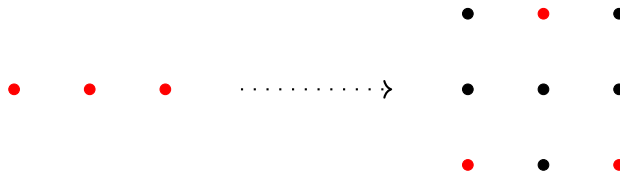
Figure 1.4: Linear encoding for $q = 3, k = 1, n = 2$

Linear code   A linear code, $\mathcal{C}$, is a linear subspace obtained by encoding, $\mathrm{Enc}(\cdot)$. It can also be seen as a set of all the encoded messages, also known as the *codewords*, which are in Figure 1.4 depicted as red dots in the vector space on the right and formally given by the following definition.

---

[1]Strictly speaking, it can also be a prime power but, in this thesis, we are not interested in the finite fields over prime powers.

Definition 1.2.1 (Linear code). *Let q be a prime number and k, n be positive integer values satisfying $n \geq k$. Let then $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ be a generator matrix that defines a linear map $\mathrm{Enc}(\cdot) : \mathbb{F}_q^k \to \mathbb{F}_q^n$ as:*

$$\forall \mathbf{x} \in \mathbb{F}_q^k, \ \mathrm{Enc}(\mathbf{x}) := \mathbf{x}\mathbf{G}.$$

A linear code, $\mathcal{C}$, is defined as a linear subspace of size q^k of a vector space $\mathbb{F}_q^n$, where a basis of the subspace is given by rows of the generator matrix $\mathbf{G}$, i.e.:

$$\mathcal{C} := \left\{ \mathbf{c} \in \mathbb{F}_q^n \mid (\exists \mathbf{m} \in \mathbb{F}_q^k) \ \mathbf{c} = \mathrm{Enc}(\mathbf{m}) \right\}.$$

As it can be seen from Definition 1.2.1, the choice of a linear map (and, consequently, the choice of the obtained code), depends crucially on the choice of a *generator matrix* $\mathbf{G}$.[2] The question then is what is a good choice of a generator matrix that would enable us to correct as many as possible errors while having the communication as efficient as possible.

Efficiency and error correcting capability   The measure of the efficiency of a linear code is given as a *code rate*, $R \in [0, 1]$. The code rate is defined as a proportion between the code dimension and the code length, i.e. $R := \frac{k}{n}$. It is not hard to see that the highest possible code rate $R = 1$ corresponds to a code for which the code dimension is equal to the code length. Without loss of generality, we can assume that, in that case, the code is actually the original message space. This further implies that encoding a message added no redundancy to it, so the error-correcting capability of the code is as high as the one of the original message space. If we then assume that each message (i.e. a codeword in this case) has a non-zero probability to occur, each mistake leads to another message (i.e. another codeword), so this error cannot be either detected or corrected. This code, therefore, has no error-correcting capability. On the other hand, for the code rate $R \approx 0$, the code dimension is negligible in comparison to the code length, i.e. $k \ll n$. In that case, given a huge amount of redundancy added to a message, even a generator matrix that is taken uniformly at random would probably be good enough to encode a message such that a certain number of errors can be detected and potentially

---

[2]We always assume it is of a full rank.

even corrected.[3] Nevertheless, it would yield an extremely inefficient encoding. Choosing a good code, therefore, comes down to optimizing the trade-off between the code rate and the error-correcting capability of the code.

1.2.2 Message decoding

In contrast to the previously described encoding process that comes down to matrix-vector multiplication, decoding is a more involved process. It requires making certain assumptions about the communication system we observe. The first assumption to be made is about the noise model, i.e. we need to assume which type of errors can occur. Another question is then how we obtain the original codeword $\mathbf{c}$ from a *noisy codeword* $\tilde{\mathbf{c}}$, i.e. the codeword altered by the communication channel. Some common approaches to this task, commonly referred to as the *decoding* task are presented in the following subsections. Once the original codeword is found, finding the original message comes down to removing the redundancy from the codeword.

Noise model The most common noise model is the additive noise model. We thus have that a *noisy codeword* $\tilde{\mathbf{c}} \in \mathbb{F}_q^n$, obtained by the receiver, is of the form $\tilde{\mathbf{c}} = \mathbf{mG} + \mathbf{e}$, where $\mathbf{e} \in \mathbb{F}_q^n$ is a random variable, commonly referred to as an *error vector*. The distribution of this random variable is determined by the physical properties of the communication channel. One of the most common settings is given by the communication channel known as the *binary symmetric one*, where the alphabet is binary, i.e. $q = 2$, and an error at each coordinate occurs with some probability $p \in [0, 1]$. The number of errors that occur, i.e. the number of non-zero coordinates of $\mathbf{e}$, can thus be seen as a random variable distributed according to Binomial distribution with parameters n and p.

Distance function Upon choosing the noise model, one can decide on the decoding strategy to use. One of the possible strategies is to say that the original message corresponds to a codeword 'closest' to the received noisy codeword, which is also referred to as *minimum distance decoding*. the question is then how we decide what the closes codeword is or, even more generally,

---

[3]More details on this will be given in the next subsection.

how we measure the distance between two codewords. Naturally, we decide to introduce a *distance function* (also known as *metric*) into our vector space, and we denote it by $\text{dist}(\cdot)$. We thus obtain two metric spaces, one corresponding to the set of original messages, and one corresponding to the set of encoded messages.

Encoding in this setting is depicted in Figure 1.5, where we present metric spaces as a unit distance graph[4]. In Figure 1.5 the distance function over which the metric space is defined as either the *Hamming distance*, which is the most common choice of a metric when the alphabet size is $q = 2$, or the so-called *Lee distance*, which is the metric commonly chosen when $q > 2$. Both of these metrics are defined in Definition 1.2.2. The two distance functions, and corresponding metric spaces, are identical for $q = 2$, as well as for $q = 3$, by definition. The difference between the two becomes apparent for higher alphabet sizes, and it will be discussed in more detail in the next chapter.

Definition 1.2.2 (Hamming and Lee distances). *Let $n \in \mathbb{N}$ be the dimension of a vector space $\mathbb{F}_q^n$. The Hamming distance, $\text{dist}_H : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{N}$, and the Lee distance, $\text{dist}_L : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{N}$, are defined as:*

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n, \mathbf{x} = (x_0, x_1, \ldots, x_{n-1}), \mathbf{y} = (y_0, y_1, \ldots, y_{n-1}),$$
$$\text{dist}_H(\mathbf{x}, \mathbf{y}) := |\{i \in \{0, 1, \ldots, n-1\} \mid x_i \neq y_i\}|,$$
$$\text{dist}_L(\mathbf{x}, \mathbf{y}) := \sum_{i=1}^{n} \min(|x_i - y_i|, q - |x_i - y_i|).$$
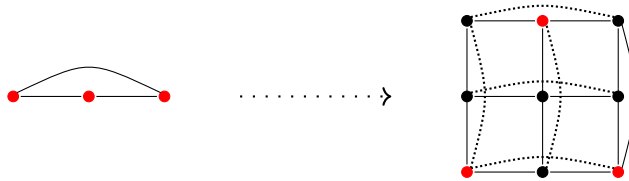
Figure 1.5: Linear encoding in the Hamming metric/Lee metric for $q = 3$, $k = 1$, and $n = 2$

---

[4]In the unit distance graph, each vertex represents an element of a metric space, and each edge connects two elements at distance one in the given metric space.

Weight function   Given a distance function, we can easily derive a corresponding *weight function*, denoted as $\mathrm{wt}(\cdot)$, and defined as the distance of any vector from the origin of the vector space. We thus obtain the following definitions of the *Hamming weight* and the *Lee weight*.

Definition 1.2.3 (Hamming and Lee weights). *Let $n \in \mathbb{N}$, q be a prime number, and $\mathbb{F}_q^n$ be a finite dimensional vector space. The Hamming weight, $\mathrm{wt}_H : \mathbb{F}_q^n \to \mathbb{N}$, and the Lee weight, $\mathrm{wt}_L : \mathbb{F}_q^n \to \mathbb{N}$, are defined as:*

$$\forall \mathbf{x} \in \mathbb{F}_q^n, \mathbf{x} = (x_0, x_1, \ldots, x_{n-1}),$$
$$\mathrm{wt}_H(\mathbf{x}) := |\{i \in \{0, 1, \ldots, n-1\} \mid x_i \neq 0\}|,$$
$$\mathrm{wt}_L(\mathbf{x}) := \sum_{i=1}^{n} \min(x_i, q - x_i).$$

Error detecting and error correcting capabilities

The minimum distance between two different codewords is called the *minimum distance* of a code.[5] Given the linearity of the code, the minimum distance can be defined equivalently as the minimum weight of non-zero codewords. More formally, the minimum distance, $d \in \mathbb{N}$, is given as:

$$d := \min_{\substack{\mathbf{x}, \mathbf{y} \in \mathcal{C} \\ \mathbf{x} \neq \mathbf{y}}} \mathrm{dist}(\mathbf{x}, \mathbf{y}) = \min_{\substack{\mathbf{x}, \mathbf{y} \in \mathbb{C} \\ \mathbf{x} \neq \mathbf{y}}} \mathrm{wt}(\mathbf{x} - \mathbf{y}).$$

   This parameter of a linear code can be regarded as an implicit measure of the error-detecting capability of the code. The reasoning behind it is the following. Let us observe a code with a distance strictly greater than t, i.e. $d > t$, for some $t \in \mathbb{N}$. Let us then assume that a receiver obtains a potentially noisy codeword $\tilde{\mathbf{c}}$ that is at distance t from the closest codeword $\mathbf{c}$. We note that, by the definition of the minimum distance, the obtained codeword cannot be valid as it is at a distance less than d from a valid codeword. Therefore, we conclude that $\tilde{\mathbf{c}}$ is noisy, i.e. it contains an error. Following a similar line reasoning, we can see that one can detect an error in $\tilde{\mathbf{c}}$ if $\mathrm{dist}(\tilde{\mathbf{c}}, \mathbf{c}) < d$. Given that $\tilde{\mathbf{c}} = \mathbf{c} + \mathbf{e}$ and that the code is linear, we can reformulate this condition

---

[5]Given that the metric space we observe is finite, such a minimum always exists.

to $\text{dist}(\tilde{\mathbf{c}}, \mathbf{c}) = \text{dist}(\tilde{\mathbf{c}} - \mathbf{c}, \mathbf{0}) = \text{dist}(\mathbf{e}, \mathbf{0}) = \text{wt}(\mathbf{e}) < d$. This statement is expressed more succinctly in the following proposition.

Proposition 1.2.1 (Error-detecting capability). *Let us fix a weight function* $\text{wt}(\cdot)$. *A linear code* $\mathcal{C}$ *with the minimum distance* d *can detect an error* $\mathbf{e}$ *when its weight is smaller than* d, *i.e.* $\text{wt}(\mathbf{e}) < d$.

Furthermore, if we want to be able to *correct errors* (instead of only detecting them), we would need to increase the minimum distance even more, as expressed via Proposition 1.2.2.

Proposition 1.2.2 (Error-correcting capability). *Let us fix a weight function* $\text{wt}(\cdot)$. *A linear code* $\mathcal{C}$ *with the minimum distance* d *can correct an error* $\mathbf{e}$ *when its weight is smaller than* $\lfloor \frac{d-1}{2} \rfloor$, *i.e.* $\text{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$.

Rigorous proofs of these propositions can be found in coding theory resources, such as [RU08], and an informal proof of Proposition 1.2.1 is already given when we explained the reasoning behind the error-detecting capability of a code. An informal proof of Proposition 1.2.2 goes as follows. Let us imagine that around each codeword in the metric space corresponding to $\mathbb{F}_q^n$ and a distance function $\text{dist}(\cdot)$ we have a ball of radius $\lfloor \frac{d-1}{2} \rfloor$ and that each ball is centered around a corresponding codeword. Notice here that, given that the minimum distance is d, the two balls cannot overlap as the closest centers are at a distance d that is strictly greater than $2\lfloor \frac{d-1}{2} \rfloor$. By minimum distance decoding, each of the noisy codewords lying in any of these balls will be decoded as the center of the ball, as it will be its closest valid codeword. So we have that any noisy codeword at the distance at most $\lfloor \frac{d-1}{2} \rfloor$ from the center of any of these balls is correctable. Namely, if $\text{dist}(\mathbf{c}, \tilde{\mathbf{c}}) < \lfloor \frac{d-1}{2} \rfloor$, the error is correctable. Again, by the linearity of the code and additivity of the noise, we have that $\text{dist}(\mathbf{c}, \tilde{\mathbf{c}}) = \text{dist}(\tilde{\mathbf{c}} - \mathbf{c}, \mathbf{0}) = \text{wt}(\tilde{\mathbf{c}} - \mathbf{c}) = \text{wt}(\mathbf{e})$, so the condition translates into $wt(\mathbf{e}) < d/2$.

Bounds on codes

Knowing that the minimum distance of a code is an implicit measure of its error-detecting and error-correcting capability, we would like to design a

code with a distance as high as possible so that it can detect/correct as many as possible errors. The question now is how high it can be. Namely, can we design a code with an arbitrarily high minimum distance? If so, what is the efficiency of such a code, i.e., what is the highest code rate we can expect? The answer to this question is given by so-called *bounds* on codes. There are numerous bounds that have been proven over the years, but here we will consider only two of them, namely, the *Gilbert-Varshamov bound* and the *Hamming bound*, that are relevant in our context. The statement of these bounds is given by the following two theorems.

Theorem 1.2.1 (Gilbert-Varshamov bound (for linear codes)). *Let q be a primer number, let $d, n \in \mathbb{N}$ satisfying $d < n$, and let $\mathrm{dist}(\cdot)$ be an arbitrary distance function. There exists a linear code $\mathcal{C}$ with alphabet size q, code length n, minimum distance d, and code rate $R \in [0,1]$ satisfying:*

$$R \geq 1 - \frac{\log_q(\mathrm{vol}_q(n, d-1)) - 1}{n},$$

where $\mathrm{vol}_q(n, d-1) \in \mathbb{N}$ denotes a volume of a ball of radius $d-1$ in a vector space $\mathbb{F}_q^n$ equipped $\mathrm{dist}(\cdot)$.

The theorem was originally proven independently by Gilbert in 1952, and Varshamov in 1957, and some simplified versions of these proofs can be found in the aforementioned coding theory sources. Less formally, the theorem states that, in the metric space of our choice, we can fix the alphabet size, q, the code length, n, and the minimum distance of a code, d, and always obtain a code of the rate at least, R, given by the Gilbert-Varshamov bound. The theorem thus basically states a possible result and provides us with a lower bound on the code rate. This theorem was originally proven for the Hamming distance but proofs of this statement (eg. in [Deb23]), can be easily adapted to general distance functions. The question then is whether there exists an impossibility result that states the upper bound on the code rate for a fixed minimum distance. An answer is given in the following theorem.

Theorem 1.2.2 (Hamming bound). *Let q be a prime number, let $d, n \in \mathbb{N}$ satisfying $d < n$, and let $\mathrm{dist}(\cdot)$ be an arbitrary distance function. The code rate $R \in [0,1]$ of a linear code $\mathcal{C}$ with alphabet size q, code length n and*

> *minimum distance d, is upper-bounded:*
>
> $$R \leq 1 - \frac{\log_q(\mathrm{vol}_q(n, \frac{d-1}{2}))}{n},$$
>
> *where $\mathrm{vol}_q(n, \frac{d-1}{2}) \in \mathbb{N}$ denotes a volume of a ball of radius $(d-1)/2$ in a vector space $\mathbb{F}_q^n$ equipped with a distance function $\mathrm{dist}(\cdot)$.*

Less formally, what the Hamming bound states can be proven as follows. Let us observe a metric space defined over a vector space $\mathbb{F}_q^n$ and a distance function $\mathrm{dist}(\cdot)$. Let then $\mathcal{C}$ be a linear code of the length n, dimension k, and distance d, defined over the given vector space. Around each codeword $\mathbf{c} \in \mathcal{C}$, we then take a ball of radius $\frac{d-1}{2}$, centered at the codeword $\mathbf{c}$. Notice here that, by definition of the minimum distance, the centers of the balls are at the distance of at least d, so none of these balls overlap. Therefore, the overall volume of the balls is given as $q^k \mathrm{vol}_q(n, \frac{d-1}{2})$, where $\mathrm{vol}_q(n, \frac{d-1}{2}) \in \mathbb{N}$ denotes a volume of a ball of radius $(d-1)/2$ in a vector space $\mathbb{F}_q^n$. The overall volume needs to be smaller than the volume of the whole vector space $\mathbb{F}_q^n$, given as q^n, so we obtain the following inequality:

$$q^k \mathrm{vol}_q(n, \frac{d-1}{2}) \leq q^n,$$

which yields the expression given in Theorem 1.2.2. The theorem thus basically states an impossibility result and provides us with an upper bound on the code rate.

The question now is how big is the gap between the two bounds presented above, namely, the lower bound given in Theorem 1.2.1 and the upper bound given in Theorem 1.2.2. Alternatively, how can we find the best trade-off between the code rate and the minimum distance? To answer these questions, we would need first to be able to calculate the volume of a ball in an arbitrary metric space.

Ball volume Let us first recall a definition of a ball. A ball $\mathcal{B}_q(\mathbf{x}, r)$, for $n, r \in \mathbb{N}$, is a set of vectors at distance at most r from some point $\mathbf{x} \in \mathbb{F}_q^n$ in a vector space equipped with a distance function $\mathrm{dist}(\cdot)$. More formally, a ball $\mathcal{B}_q(\mathbf{x}, r)$ is given by the following expression:

$$\mathcal{B}_q(\mathbf{x}, r) := \{\mathbf{y} \in \mathbb{F}_q^n \mid \mathrm{dist}(\mathbf{x}, \mathbf{y}) \leq r\}.$$

Less formally, a ball in a Hamming/Lee metric space is depicted in 1.6, where cyan dots represent vectors of a ball centered at vector denoted as $\mathbf{x}$.
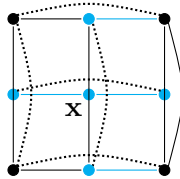
Figure 1.6: Hamming/Lee ball for $q = 3$, $n = 2$, and $r = 1$

Hamming ball   In the case of the Hamming metric space, calculating the volume of a ball comes down to counting the number of vectors with a certain number of positive entries. Therefore, the ball volume, $\mathrm{vol}_q(\cdot)$, can be expressed as:

$$\mathrm{vol}_q(n, d-1) = \sum_{i=0}^{d-1} \binom{n}{i}(q-1)^i,$$

where $\binom{n}{i}$ denotes a binomial coefficient calculated as $\binom{n}{i} := \frac{n!}{i!(n-i)!}$.

For other distance functions, including the Lee metric, calculating the volume of a ball is a more involved task. In the next chapter, we suggest an approach for calculating the ball volume for the class of the weight functions that we refer to as the *elementwise weight functions*. We thus generalize the calculation of the ball volume for different metric spaces based on these functions.

Decoding methods

As discussed previously, the most intuitive decoding strategy is to say that the original message corresponds to a codeword with the shortest distance from the received, potentially noisy, codeword. This reasoning yields the following decoding method.

Method 1.2.1 (Minimum distance decoding). *Given a noisy codeword $\tilde{\mathbf{c}}$, a linear code $\mathcal{C}$ and a distance function $\mathrm{dist}(\cdot)$, find a codeword $\mathbf{c} \in \mathcal{C}$ such*

that:
$$\text{dist}(\tilde{\mathbf{c}}, \mathbf{c}) := \min_{\mathbf{y} \in \mathcal{C}} \text{dist}(\tilde{\mathbf{c}}, \mathbf{y}).$$
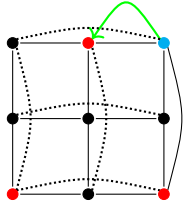
(a) Uniquely decodable
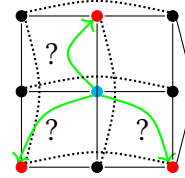
(b) Non-uniquely decodable

Figure 1.7: Decoding in the Hamming metric/Lee metric for $q = 3$, $k = 1$, and $n = 2$

Syndrome decoding Alternatively, given that the noisy codeword can be expressed as $\tilde{\mathbf{c}} = \mathbf{c} + \mathbf{e}$, decoding can be seen as a search for the error vector $\mathbf{e}$ of the minimum weight. The question then is how we can make use of this fact. The answer to it is given through a method known as *syndrome decoding*. To describe this method, we first need to introduce two additional coding theory notions, the first one being a notion of *parity-check matrix*. Namely, a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ that satisfies $\mathbf{G}\mathbf{H}^T = \mathbf{0}$, where $\mathbf{0}$ is a matrix of all zeros in $\mathbb{F}_q^{k \times (n-k)}$, is called a *parity-check* matrix. We thus obtain an alternative definition of a linear code (equivalent to Definition 1.2.1).

Definition 1.2.4 (Linear code)**.** *Let q be a prime number and k, n be positive integer values satisfying $n \geq k$. Let $\text{Lin} : \mathbb{F}_q^n \times \mathbb{F}_q^{(n-k) \times n} \to \mathbb{F}_q^{n-k}$ be a linear map defined as $\text{Lin}(\tilde{\mathbf{c}}, \mathbf{H}) := \tilde{\mathbf{c}}\mathbf{H}^T = \mathbf{s}$, for some $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times k}$, $\tilde{\mathbf{c}} \in \mathbb{F}_q^n$, and $\mathbf{s} \in \mathbb{F}_q^{n-k}$. A linear code, $\mathcal{C}$, is the kernel of $\text{Lin}(\cdot)$, given as:*

$$\mathcal{C} := \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{c}\mathbf{H}^T = \mathbf{0}\}.$$

Now, let us make use of this alternative definition and start by calculating the so-called *syndrome*, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, corresponding to the noisy codeword. Assuming the noise is additive and that the code is linear, we obtain

$$\mathbf{s} := \tilde{\mathbf{c}}\mathbf{H}^T = (\mathbf{x}\mathbf{G} + \mathbf{e})\mathbf{H}^T = \mathbf{x}\mathbf{G}\mathbf{H}^T + \mathbf{e}\mathbf{H}^T = \mathbf{e}\mathbf{H}^T.$$

Finding the closest valid codeword then comes down to finding the error of a minimum weight that corresponds to the syndrome, hence, the name of the method.

Method 1.2.2 (Syndrome decoding). *Given a noisy codeword $\tilde{\mathbf{c}}$, a linear code $\mathcal{C}$, defined by its parity-check matrix $\mathbf{H}$, and a weight function $\mathrm{wt}(\cdot)$, first calculate the syndrome $\mathbf{s}$ as:*

$$\mathbf{s} = \tilde{\mathbf{c}}\mathbf{H}^T,$$

and then find an error $\mathbf{e}$ satisfying:

$$\mathrm{wt}(\mathbf{e}) = \min_{\tilde{\mathbf{e}} \in \tilde{\mathcal{E}}} \mathrm{wt}(\tilde{\mathbf{e}}), \quad \tilde{\mathcal{E}} := \{\tilde{\mathbf{e}} \in \mathbb{F}_q^n \mid \tilde{\mathbf{e}}\mathbf{H}^T = \mathbf{s}\}.$$

As this decoding method is at the core of the research presented in this thesis, we will focus on it during the rest of this thesis. More precisely, in Chapter 2, we will explain how this method can be translated into a so-called the *syndrome decoding problem* that is used as a basis of different cryptographic protocols including ours. In Chapter 3, we will give a more detailed explanation of how efficiently this method can be used for decoding a particular class of linear codes called *random linear codes*, which we will describe in short. Finally, in Chapter 4, we will give an example of how the syndrome decoding problem can be applied to the design of cryptographic protocols.

Good, bad, and random codes

Given all that we have seen so far, what would be a good choice for a linear code? Namely, how should we choose a generator matrix (or, equivalently, a parity check matrix)? As we already discussed, from the coding theory perspective, we should search for a code that maximizes the trade-off between the code rate and the minimum distance. Given the Gilbert-Varshamov bound, for a given minimum distance, we know what rate we can always achieve. The Hamming bound, on the other hand, tells us what is impossible to achieve. The gap between the two leaves space for code designers to find a sweet spot between what we can always achieve and what is the best we can hope for. From what we learned in practice, finding this sweet spot is not an easy task. Moreover, one can argue that even if we find a linear code with

a fairly good trade-off, it does not necessarily mean that we have an efficient algorithm to decode it. What we have also learned from practice is that codes with good decoding algorithms are even harder to find.

From a cryptographic perspective, however, this is good news. Namely, in cryptography, we often search for codes that have no efficient decoding algorithms. A class of codes that is particularly interesting from the cryptographic perspective is the class of so-called *random codes*. In the case of linear codes, random codes are those obtained by sampling a generator matrix (or a parity-check matrix) uniformly at random from $\mathbb{F}_q^{k \times n}$ (or from $\mathbb{F}_q^{(n-k) \times n}$ in the case of the parity-check matrix). As codes with no particular structure, apart from their inherent linearity, we learned from practice that these have a wide range of implementations in cryptography. This is mainly due to the fact that it is extremely hard to find an efficient decoding algorithm for decoding these codes, as we will elaborate on in Chapter 3. We remark here, however, that it is not strictly necessary for a code to be random to yield a good basis for a cryptographic protocol. Famous examples of cryptographic protocols based on problems of decoding linear codes, such as [McE78], would contradict that claim. In Chapter 4, we will thus elaborate a bit more on how we can benefit from the inherent randomness of the codes we use for our protocol design.

Glossary of linear codes

We here give a brief recapitulation of the basic terms introduced in this chapter that will be useful for the rest of this thesis:

alphabet, $\mathbb{F}_q$ - a finite set of characters; a finite field

alphabet size, $q \in \mathbb{N}$ - the size of an alphabet; the size of the finite field corresponding to an alphabet

code, $\mathcal{C} \subseteq \mathbb{F}_q^n$ - a set of encoded messages; a linear subspace of a finite-dimensional vector space containing all the codewords

code dimension, $k \in \mathbb{N}$ - the length of a message; the dimension of a message space

minimum distance, $d \in \mathbb{N}$ - a minimal distance between two distinct codewords; an implicit measure of the error-detecting and error-correcting

capability

code length, $n \in \mathbb{N}$ - the length of encoded messages; the dimension of the vector space containing all the codewords and noisy codewords

code rate, $R \in [0,1]$ - the proportion of code dimension and the code length; an implicit measure of the code efficiency

codeword, c $\in \mathbb{F}_q^n$ - an encoded message; an element of the code; a vector in the code sub-space; an output of the encoding

encoding, $\text{Enc}(\cdot) : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$ - adding redundancy to the message; a linear map that maps a message into a codeword

Gilbert-Varshamov bound, GV - an upper bound on the highest code rate of a code with a fixed minimum distance

error, error vector, e $\in \mathbb{F}_q^n$ - a noise; a random vector in a vector space of dimension n

generator matrix, G $\in \mathbb{F}_q^{k \times n}$ - a matrix that defines the encoding

message, m $\in \mathbb{F}_q^k$ - an array of characters; a vector in a message space; an input of an encoding algorithm

message space, $\mathbb{F}_q^k$ - a set of all the messages; a finite-dimensional vector space containing all the messages

noisy codeword, $\tilde{\mathbf{c}}$ $\in \mathbb{F}_q^n$ - an encoded message with additive noise added to it; a codeword with error

parity-check matrix, H $\in \mathbb{F}_q^{(n-k) \times n}$ - a matrix that defines a linear encoding

syndrome, s $\in \mathbb{F}_q^{n-k}$ - a product of the parity check matrix and the error vector

1.3 Computational models and algorithms

A *computational model* can be seen as an abstract description of general-purpose computational devices that have common internal logic and come with the same level of computing power (scaled with the size of the computational devices). In this thesis, we are interested in two types of computational models, namely, the *classical circuit model* and the *quantum circuit model*. The first one describes classical (digital) general-purpose computational devices, also known as *classical computers*, which are considered to be today's state-of-the-art technology. The quantum circuit model, on the other hand, describes general-purpose quantum computational devices that promise to achieve a computational advantage over classical computers if technological development overcomes practical difficulties in creating full-scale (or at least middle-scale) quantum devices. In the rest of this section, we will present an overview of the fundamental concepts describing the two models focusing on their comparison.

1.3.1 Basic units of information

Classical bits   Classical computers operate on *boolean values*, taken from the set $\{0, 1\}$. We thus take the basic unit of classical information to be the *bit* that can take a value of either 0 or 1. The overall data then can be represented as a sequence of zeros and ones, or, more mathematically, as a vector in $\{0, 1\}^n$, where $n \in \mathbb{N}$. A computation then can be presented as evaluating a vectorial boolean function, $f : \{0, 1\}^k \to \{0, 1\}^n$, $k, n \in \mathbb{N}$, that takes as an input a vector in $\{0, 1\}^k$, performs logical operators such as $\neg, \wedge, \vee, \oplus$, etc. on it, and returns the output in $\{0, 1\}^n$.

Quantum bits (qubits)

The basic unit of information in quantum computing is called *quantum bit* or, simply, *qubit*. In terms of linear algebra, a qubit can be described as a vector (point) on the unit-distance sphere in the two-dimensional complex space, $\mathbb{C}^2$. Similarly, multiple qubits can be described as a point on a unit-distance sphere in the higher-dimensional complex space, $\mathbb{C}^{2^n}$, where $n \in \mathbb{N}$ is the

number of qubits. Each qubit (or multi-qubit) thus can be represented using a *basis* of the corresponding complex vector space.

Standard basis One of the most common basis choices is an orthonormal basis known as the *standard basis*. In the case of $\mathbb{C}^2$, it is formed by vectors $|0\rangle, |1\rangle$, defined as:

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ |1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

where $|\cdot\rangle$ is the so-called *Dirac* or the *bra-ket* notation of vectors that are commonly encountered in quantum computing and, more generally, in quantum mechanics. Using the standard basis, we can present an arbitrary single-qubit quantum state, $|\psi\rangle \in \mathbb{C}^2$, as:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle,$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$ are the so-called *complex amplitudes* that need to satisfy $|\alpha_0|^2 + |\alpha_1|^2 = 1$. If we now want to present a multi-qubit state, we need to extend our standard basis to $\mathbb{C}^{2^n}$. This is done through the tensor product, denoted by $\otimes$, of basis vectors in $\mathbb{C}^2$. For example, for two-qubit states, we obtain the standard basis given as:

$$|0\rangle \otimes |0\rangle \equiv |0,0\rangle \equiv |00\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

$$|0\rangle \otimes |1\rangle \equiv |0,1\rangle \equiv |01\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix},$$

$$|1\rangle \otimes |0\rangle \equiv |1,0\rangle \equiv |10\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix},$$

$$|1\rangle \otimes |1\rangle \equiv |1,1\rangle \equiv |11\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Similarly, for an arbitrary number of qubits, we obtain the standard basis comprising of following vectors:

$$|00\ldots0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \ |00\ldots1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \ \ldots, \ |111\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

In the standard basis of $\mathbb{C}^{2^n}$, we can then present an n-qubit state, $|\phi\rangle \in \mathbb{C}^{2^n}$, as:

$$|\phi\rangle = \sum_{i\in\{0,1\}^n} \alpha_i \, |i\rangle \, ,$$

where $\alpha_i \in \mathbb{C}, i \in \{0,1\}^n$, are coefficients satisfying $\sum_{i\in[n]} |\alpha_i|^2 = 1$.

Hadamard basis   Another commonly used basis is the so-called *Hadamard basis*, formed by the following two basis vectors:

$$|+\rangle := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \quad |-\rangle := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

It is not hard to verify that this basis is also orthonormal and that the quantum state $|\psi\rangle$, previously given via the standard basis, can be presented via the Hadamard basis, as :

$$|\psi\rangle = \beta_0 \, |+\rangle + \beta_1 \, |-\rangle \, ,$$

where $\beta_0, \beta_1 \in \mathbb{C}$ satisfy:

$$\beta_0 = \frac{\alpha_0 + \alpha_1}{\sqrt{2}}, \ \beta_1 = \frac{\alpha_0 - \alpha_1}{\sqrt{2}}.$$

As in the case of the standard basis, the Hadamard basis can be extended to a multi-qubit basis using the tensor product. For the two-qubit state, the

Hadamard basis comprises the following vectors:

$$|+\rangle \otimes |+\rangle \equiv |+, +\rangle \equiv |++\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix},$$

$$|+\rangle \otimes |-\rangle \equiv |+, -\rangle \equiv |+-\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix},$$

$$|-\rangle \otimes |+\rangle \equiv |-, +\rangle \equiv |-+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix},$$

$$|-\rangle \otimes |-\rangle \equiv |-, -\rangle \equiv |--\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}.$$

The Hadamard basis of $\mathbb{C}^{2^n}$, where $n \in \mathbb{N}$, is then obtained by combining the basis vectors $\{|b_i\rangle\}_{i \in 2^n}$, given as:

$$|b_0\rangle := |+ + \cdots +\rangle = \frac{1}{2^{n/2}} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \cdots \\ 1 \\ 1 \end{pmatrix}, \quad |b_1\rangle := |+ + \cdots -\rangle = \frac{1}{2^{n/2}} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \cdots \\ 1 \\ -1 \end{pmatrix}, \ldots,$$

$$|b_{2^n-2}\rangle := |- - \cdots +\rangle = \frac{1}{2^{n/2}} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ \cdots \\ 1 \\ 1 \end{pmatrix}, \quad |b_{2^n-1}\rangle := |- - \cdots -\rangle = \frac{1}{2^{n/2}} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ \cdots \\ 1 \\ -1 \end{pmatrix}.$$

An n-qubit state, $|\phi\rangle \in \mathbb{C}^{2^n}$, with coefficients $\beta_i \in \mathbb{C}, i \in \{0,1\}^n$, satisfying $\sum_{i \in \{0,1\}^n} |\beta_i|^2 = 1$, is then given as:

$$|\phi\rangle = \sum_{i \in \{0,1\}^n} \beta_i |b_i\rangle .$$

As in the one-qubit case, we can derive the coefficients of $\{\beta_i\}_{i \in \{0,1\}^n}$ from the coefficients $\{\alpha_i\}_{i \in \{0,1\}^n}$ representing the n-qubit state $|\phi\rangle$ in the standard basis. For each $i \in \{0,1\}^n$, we thus obtain:

$$\beta_i = \frac{\sum_{j \in \{0,1\}^n} (-1)^{i \cdot j} \alpha_j}{2^{n/2}},$$

where $i \cdot j$ is the bitwise dot product.

1.3.2   Basic processing units

Logic gates   *Logic gates* (or, simply, *gates*) are the basic processing units in the classical circuit model. As each gate corresponds to a logical operator, on the input of a gate, we have entries of the corresponding logical operator, and on the output the result of the operation. Some of the common gates are depicted in Figure 1.8, where all $x_0, x_1 \in \{0,1\}$ represent inputs and each $y_0 \in \{0,1\}$ is an output of a gate.
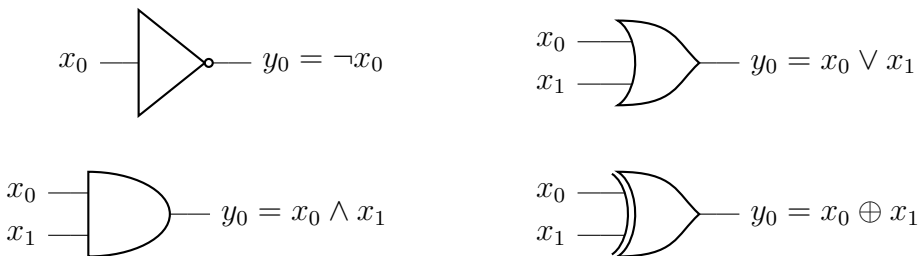
Figure 1.8: Classical NOT, OR, AND, and XOR gates.

Quantum gates

As in the case of classical computing, the basic processing unit in the quantum case is a logic gate, commonly referred to as either *quantum logic gate*, or

simply *quantum gate.* In the classical case, logical gates correspond to *logical operators* acting commonly on one-bit or two-bit inputs. In the quantum case, on the other hand, each logic gate corresponds to a *unitary operator* that acts on one qubit or multiple qubits.

Single-qubit quantum gates   Let us denote by U a unitary operator that takes one qubit quantum state as an input, $|\psi_{in}\rangle \in \mathbb{C}^2$, and outputs a one-qubit quantum state as an output, $|\psi_{out}\rangle \in \mathbb{C}^2$. In the standard quantum computing notation, the action of the corresponding quantum gate is then given as:

$$|\psi_{in}\rangle \;\boxed{U}\; |\psi_{out}\rangle$$

Figure 1.9: One-qubit quantum gate.

To be consistent with a vector representation of quantum states, we present each unitary operator as a unitary matrix in the computational basis that corresponds to the representation of a qubit processed by the gate. Applying a quantum gate U to a qubit $|\phi_{in}\rangle$ thus comes down to calculating the matrix inner product between a vector representation of the qubit and a unitary matrix $U \in \mathbb{C}^{2\times 2}$ corresponding to the quantum gate. The output $|\psi_{out}\rangle$ is then obtained as:

$$|\phi_{out}\rangle = U\,|\phi_{in}\rangle\,.$$

Without loss of generality, we choose to represent both qubits and quantum gates in the standard basis.

Multi-qubit quantum gates   Let then U be a unitary operator taking as an input a n-qubit state $\phi_{in} \in \mathbb{C}^{2^n}$ and outputting $\phi_{out} \in \mathbb{C}^{2^n}$. If we take $n = 3$, the action of the corresponding quantum gates is presented as:

$$|\phi_{in}\rangle \;\boxed{U}\; |\phi_{out}\rangle \;\equiv\; |\phi_{in}\rangle \;\xrightarrow{\;3\;}\boxed{U}\xrightarrow{\;3\;}\; |\phi_{out}\rangle$$

Figure 1.10: Multi-qubit quantum gate.

We can generalize the matrix representation of unitary operators from operators acting on a single qubit to multi-qubit operators. We thus obtain a unitary matrix $U \in \mathbb{C}^{2^n \otimes 2^n}$ corresponding to the quantum operator. Applying the corresponding quantum gate U to a qubit $|\psi_{in}\rangle$ then comes down to calculating the matrix inner product between a vector representation of the qubit and the unitary matrix U. As in the single-qubit case, the output $|\phi_{out}\rangle$ is then obtained as:

$$|\phi_{out}\rangle = U |\phi_{in}\rangle,$$

and we can choose to represent both qubits and quantum gates in the standard basis.

Quantum gates examples

Pauli gates   The four gates depicted in Figure 1.11, along with the corresponding unitary matrices, are known as *(single-qubit) Pauli gates*.

$$I := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$Y := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$Z := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Figure 1.11: Pauli gates.

When acting on multi-qubits, these gates are combined using tensor product into the multi-qubit Pauli gates, $\mathcal{P}_n$, given as:

$$\mathcal{P}_n := \{U_1 \otimes U_2 \otimes \cdots \otimes U_n \mid U_i \in \{I, X, Y, Z\}\}.$$

Hadamard gates   The single-qubit Hadamard single is depicted in Figure 1.12, along with the corresponding unitary matrix. We observe here that the Hadamard gate effectively rotates the one-qubit standard basis into the

$$-\boxed{H}- \qquad H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Figure 1.12: Single-qubit Hadamard gate.

one-qubit Hadamard basis. Similarly, by applying the Hadamard gate on each qubit of a multi-qubit state expressed via the standard basis, we obtain the Hadamard-basis representation of a multi-qubit state. The effect of the Hadamard basis on the single-qubit standard basis state is depicted in Figure 1.13.

$$|0\rangle -\boxed{H}- |+\rangle \qquad |1\rangle -\boxed{H}- |-\rangle$$

Figure 1.13: Standard to Hadamard basis.

The effect on the multi-qubit Hadamard gate on a multi-qubit quantum state, expressed via the standard basis, is depicted in Figure 1.14. Here the states

$$|\psi\rangle := \sum_{i \in \{0,1\}^n} \alpha_i |i\rangle \; \boxed{H} \; |\phi\rangle := \sum_{i \in \{0,1\}^n} \beta_i |b_i\rangle$$

Figure 1.14: Standard to Hadamard basis.

$|\psi\rangle, |\phi\rangle \in \mathbb{C}^{2^n}$ are expressed via the standard basis and the Hadamard basis, respectively, and the relation between the coefficients of the two representations is given by:

$$\beta_i = \frac{\sum_{j \in \{0,1\}^n} (-1)^{i \cdot j} \alpha_j}{2^{n/2}}.$$

We observe here that the basis change, from the standard basis of state $|\psi\rangle$ to the state $|\phi\rangle$ in the Hadamard basis, thus comes down to applying a single multi-qubit Hadamard gate.

Clifford gates We now introduce a set of gates comprising of the elements from the so-called *Clifford group*, hence the name *Clifford gates*. Being part of a group, each gate in the set can be obtained as a combination of other gates within the set. Moreover, as the generators of the group are the *Hadamard* gate, H, the *phase gate*, S, and the *controlled-not gate*, $CNOT$, each gate in

the set can be obtained as a combination of these three gates. The three gates, along with the corresponding unitary matrices, are presented in Figure 1.15.

$$H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$P := \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

$$CNOT := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 1.15: Clifford group generators.

The set of Clifford gates is considered to be interesting for various reasons. First, it contains the Pauli gates. To see that, we observe that:

$$I = HH, Z = SS, X = HZH, Y = SXS^{\dagger},$$

where $\dagger$ denotes the Hermitian conjugate. Since all the Pauli gates can be generated using only H and S, which are generators of the Clifford group, the four Pauli gates belong to the Clifford group, too. These gates can be seen as a natural generalization of the logical operator $\neg$ and, as such, regarded as a valuable asset in the computational model. In addition to that, the group contains the Hadamard and the controlled-not gates. These gates are seen as rather useful because they appear to capture the inherently quantum nature of the quantum circuit model. Namely, the two gates combine to produce the so-called *entanglement* of qubits, which is an inherently quantum effect.

Nevertheless, despite the indications that this set should suffice for expressing the computational advantage of the quantum circuit model over the classical one, it turns out that, in fact, it is not true. Namely, it can be shown that a quantum circuit consisting of the Clifford gates only can be efficiently simulated by a classical circuit [Got98]. However, as we will show in short, this gate set is just one gate away from the *universal quantum gate set*, which yields a model that is strongly believed to have a computational advantage over the classical model.

Measurement

We now introduce the only non-reversible processing units encountered in the quantum circuit model, called *measurements*. We use measurement to access the result of a computation that otherwise, by postulates of quantum mechanics, would be inaccessible to us. By measuring a quantum state, we *collapse* it to one of the basis states that can be represented via classical binary values, i.e. $\{0, 1\}$. The outcome of measurement thus becomes a sequence of basis states that corresponds to a sequence of (classical) binary values.

In the general theory of quantum computing, there exist different types of measurements. Here we will present one that is relevant in our context and which we refer to as a *measurement with respect to a computational basis*. Mathematically speaking, it can be seen as a projection of a quantum state on the states of a computational basis.

Single-qubit measurement   In the single-qubit case, the measurement with respect to a basis is calculated as the inner product between the given quantum state and a basis vector. If we now take $|\psi\rangle \in \mathbb{C}^2$ to be a single-qubit state to measure, the projections on the vectors in the standard basis are given as:

$$\alpha_0 = \langle\psi|0\rangle, \quad \alpha_1 = \langle\psi|1\rangle,$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$. The state ϕ then can be presented as:

$$\psi = \langle\psi|0\rangle\,|0\rangle + \langle\psi|1\rangle\,|1\rangle = \alpha_0\,|0\rangle + \alpha_1\,|1\rangle.$$

The probabilities that ψ collapses to either the basis state $|0\rangle$ or the basis state $|1\rangle$ i.e. the probabilities of obtaining values 0 and 1 as the outcome of computation, are calculated as:

$$|\langle\psi|0\rangle|^2 = |\alpha_0|^2, \quad |\langle\psi|1\rangle|^2 = |\alpha_1|^2.$$

Multi-qubit measurement   In the multi-qubit case, we discriminate over two measurements, namely, the *partial measurement* and the *full measurement*. In the case of a partial measurement with respect to a computational basis, we project an n-qubit state onto a subspace of dimension $m \in \mathbb{N}$, $m \leq n$, spanned by a subset of the basis states. We thus obtain a new state that

comprises measured states that collapsed and the rest of the states, which are not measured strictly speaking but may potentially collapse as we will explain in short. To obtain the full measurement, we let $m = n$ and thus measure all the states.

A particularly interesting case occurs when we perform a partial measurement on an *entangled state*. This state is created by combining, for example, the Hadamard and the $CNOT$ gate, and we can define it as any quantum state that is not *separable*. In our linear-algebra interpretation of quantum computing, this means that we cannot write down a multi-qubit state as a tensor product of two other states belonging to smaller vector spaces.
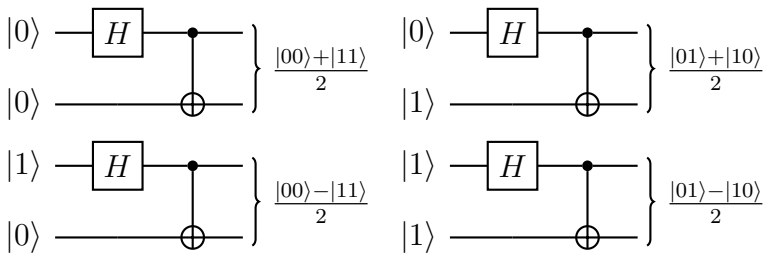
Figure 1.16: Four Bell states.

The most common example of entangled states is the so-called *Bell states*, presented in Figure 1.16. We observe here that indeed none of these states can be written down as a tensor product of two single-qubit states. Moreover, after measuring the first qubit of any $|\psi_i\rangle$, $i \in [3]$, the measurement of the second qubit is already predetermined. For example, if the outcome of the measurement of the first qubit of $|\psi_0\rangle$ is $|0\rangle$, we know that the measurement of the second qubit will always be $|0\rangle$. On the other hand, if the measurement of the first qubit gives $|1\rangle$ as the outcome, the second state will automatically collapse to state $|1\rangle$, too. The same goes for measuring any other $|\psi_i\rangle$. This effect is widely exploited in *quantum protocol* design and *quantum algorithms*, and we will see some examples of how it can be used in the rest of the section.

1.3.3 Circuits

Boolean circuits Different logic gates, with the corresponding inputs and outputs, can be combined into a so-called *boolean circuit* that evaluates a

boolean function. An example of a circuit that evaluates a boolean function $f : \{0,1\}^k \to \{0,1\}^n$ is given in Figure 1.17, where the input of a circuit is given as $(x_0, x_1, x_2) \in \{0,1\}^3$ and the output $y_0 \in \{0,1\}$ is evaluated as:

$$y_0 = f(x_0, x_1, x_2) = (\neg(x_0 \vee x_1)) \oplus (\neg x_0 \wedge (x_1 \vee x_2)).$$
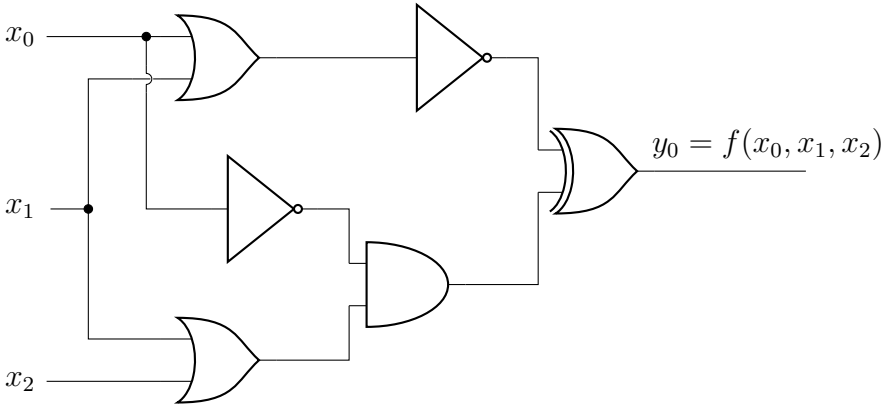
Figure 1.17: Example of a boolean circuit.

Quantum circuits

Similarly to classical gates, quantum gates can be combined into a *quantum circuit*. An example of a quantum circuit that combines different single-qubit and multi-qubit gates is given in Figure 1.18. The example illustrates the gen-
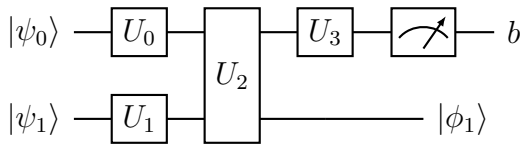
Figure 1.18: A quantum circuit.
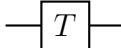
eral pattern of most of the quantum circuits: the *input*, given as qubit states $|\psi_0\rangle$ and $|\psi_1\rangle$, the *internal computation*, given by unitaries U_0, U_1, U_2 and U_3, and the *output* given as a classical state b and a qubit state $|\phi_1\rangle$. The number of input/output qubits, of course, varies, as well as the internal logic determined by the choice of the unitaries to use, but the general pattern remains the same for most of the quantum circuits.

We observe here that the overall action of a quantum circuit, up to measurement, can be presented by a unitary matrix that determines the internal logic of the whole circuit. Let us denote this unitary matrix by U. In the circuit example given in Figure 1.18, we then have:

$$U = (U_3 \otimes I)U_2(U_0 \otimes U_1).$$

Universal gate set   From what we have seen so far, it appears that the choice of the set of available logical/quantum gates determines which boolean circuits can be evaluated. An intuition says that a more diverse gate set should imply more diverse boolean circuits. Nevertheless, it can be shown that we can construct a gate set of finite size with logical gates that can be combined into a boolean circuit that calculates arbitrary boolean functions. We call such a set a *universal (classical) gate set.* Similarly, from Solovay–Kitaev theorem, we know that from a properly chosen set of quantum gates, we can construct a quantum circuit that approximates (arbitrarily well) any unitary.

Two prominent examples of (classical) universal gate sets are $\{\text{NOT}, \text{AND}\}$ and $\{\text{NOT}, \text{OR}\}$. These sets, in fact, can be used to form the so-called universal gates by combining the whole set into just one gate. We thus obtain NAND and NOR gates, each of which alone can form a circuit that can evaluate an arbitrary boolean function. As already indicated, the set of Clifford gates, though not universal on its own, can be extended to yield a universal set of quantum gates. It can be shown that, in fact, the union of the Clifford set, and any other gate that is not in the Clifford group form a universal set of quantum gates [NC16]. For example, the T gate, depicted in Figure 1.19, along with the Clifford gates forms a universal set. More generally, the T gate in the universal set can be replaced with almost any phase-shift, depicted in 1.20, for which $\theta \in [0, 2\pi]$.

$$\boxed{T} \qquad T := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

Figure 1.19: The T gate.

The question which now comes naturally is how do we choose which universal gate set to use? It can be shown that different choices of universal gate sets provide different trade-offs between the number of gates in the

$$P(\theta) := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

Figure 1.20: The P gate.

gate set (i.e. the number of physical gates that need to be fabricated) and the so-called *circuit complexity*, which will be explained in short. In addition to that, to decide which gate set to choose, we would need to take into account the efficiency of the physical implementation of different gates. In this thesis, however, we do not go so deeply into the circuit implementation, and we just state that there exists a universal gate set (arbitrary one) that enables one to calculate an arbitrary boolean function/unitary. We thus do not make any assumption on the optimality of a universal gate set implementation. With a given gate set, the question now is how efficiently we can calculate boolean functions (resp. unitaries) by evaluating boolean circuits (resp. quantum circuits).

Circuit complexity The most relevant property for circuit implementation is its *complexity*. It can be evaluated either through the overall *number of gates* in the circuit or through the so-called *circuit depth* that counts the number of gates on the longest path from the input to the output of the circuit. Both of these measures can be related to the time necessary for performing a calculation, and, as such, these are taken as basic measurements of circuit efficiency.

1.3.4 Algorithms

So far, we have identified each computation with a circuit calculating a boolean function/unitary. In the classical setting, we can move to a higher level of abstraction and identify each computation with a finite sequence of *instructions* that correspond to a part of a circuit. Though certain quantum algorithms, like *quantum walks* that we will mention a bit later in this subsection, can also be seen as a higher-level abstraction of quantum circuits, the quantum algorithms relevant to the context of this thesis, as well as most of the well-known examples of quantum algorithms, can be presented simply as quantum circuits. We will now present a few examples of these algorithms.

Quantum algorithms for promise problems

The first example is an algorithm that is used for solving a specific *promise problem* that we will explain in short. It is not used in this thesis but we present it here because it is considered to be the first example to illustrate the computational advantage of quantum computing over classical computing. In addition to that, different quantum algorithms that came after are based on this algorithm. As such, it is considered to be one of the crucial steps in the development of quantum computing.

Deutsch's algorithm Deutsch algorithm aims to solve a computational task of deciding if a given function $f : \{0, 1\} \rightarrow \{0, 1\}$ is either *constant* or *balanced*. It does so through the function calls, that we refer to as *queries*, which enable us to learn the type of the function. Namely, if the function is constant, we have that $\forall x, y \in \{0, 1\}, f(x) = f(y)$. On the other hand, if the function is balanced, then $\forall x, y \in \{0, 1\}, f(x) \neq f(y)$.

To see the promised advantage of the Deutsch algorithm over classical algorithms, let us first consider what would be the required number of calls that a classical algorithm makes to function $f(\cdot)$ in order to determine its nature. It turns out that any classical algorithm needs to make at least two calls to decide if $f(\cdot)$ is constant or balanced. In the quantum setting, on the other hand, as shown by Deutsch, after just one call to the unitary that corresponds to $f(\cdot)$, the algorithm can perfectly discriminate if the function is constant or balanced.

CHAPTER 1

To see that, let us observe the Deutsch algorithm presented in Figure 1.21. First, we notice that in order to make the function $f(\cdot)$ reversible, and thus
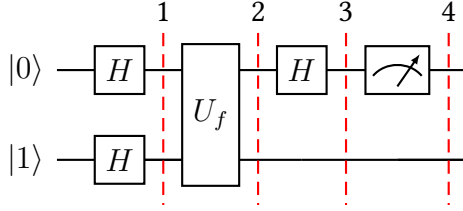
Figure 1.21: Deutsch algorithm.

computable by a quantum circuit, we need to encode it as a unitary. We denote this unitary by U_f and define it as follows:

$$U_f \ket{x, y} = \ket{x, y + f(x)},$$

where $x, y \in \{0, 1\}$. We can then evaluate $f(\cdot)$ by applying U_f to an input $\ket{\psi} \in \mathbb{C}^{2^2}$. The unitary U_f is commonly referred to as the *oracle*, and we say that we *query* it by evaluating the result when the oracle is applied to a certain input $\ket{\psi}$. The goal of the quantum algorithm is then to decide if $f(\cdot)$ is constant or balanced by making as small as possible number of queries to the oracle U_f.

Let us now analyze the Deutsch algorithm by observing the quantum states at each time step of the algorithm. We take as an input the state $\ket{\psi} = \ket{01}$ and denote the quantum state of the two qubits at moment $t \in \{1, 2, 3, 4\}$ by $\ket{\psi_t}$. After the first two Hadamard gates, we thus obtain the state $\ket{\psi_1}$ given as:

$$\ket{\psi_1} = \frac{\ket{00} - \ket{01} + \ket{10} - \ket{11}}{2}.$$

After applying the unitary U_f to the state $\ket{\psi_1}$, we obtain the state $\ket{\psi_2}$ given as:

$$\begin{aligned}
\ket{\psi_2} = U_f \ket{\psi_1} &= \frac{U_f \ket{00} - U_f \ket{01} + U_f \ket{10} - U_f \ket{11}}{2} \\
&= \frac{\ket{0, 0 \oplus f(0)} - \ket{0, 1 \oplus f(0)} + \ket{1, 0 \oplus f(1)} - \ket{1, 1 \oplus f(1)}}{2} \\
&= \frac{\ket{0, f(0)} - \ket{0, 1 \oplus f(0)} + \ket{1, f(1)} - \ket{1, 1 \oplus f(1)}}{2}.
\end{aligned}$$

After applying the final Hadamard gate on the first qubit of the state $|\psi_2\rangle$, we obtain $|\psi_3\rangle$ as:

$$|\psi_3\rangle = \frac{|+, f(0)\rangle - |+, 1 \oplus f(0)\rangle + |-, f(1)\rangle - |-, 1 \oplus f(1)\rangle}{2}.$$

Let us now observe what would be the state $|\psi_3\rangle$ if the function is constant:

- case 1 - $f(0) = f(1) = 0$:

$$|\psi_3\rangle = \frac{|+, 0\rangle - |+, 1\rangle + |-, 0\rangle - |-, 1\rangle}{2}$$

$$= |+\rangle\,|-\rangle + |-\rangle\,|-\rangle = \left(|+\rangle + |-\rangle\right)|-\rangle = |0\rangle\,|-\rangle.$$

- case 2 - $f(0) = f(1) = 1$:

$$|\psi_3\rangle = \frac{|+, 1\rangle - |+, 0\rangle + |-, 1\rangle - |-, 0\rangle}{2}$$

$$= -|+\rangle\,|-\rangle - |-\rangle\,|-\rangle = -\left(|+\rangle + |-\rangle\right)|-\rangle = -|0\rangle\,|-\rangle.$$

After the measurement, we obtain $|\pm\langle 0|0\rangle|^2 = 1$, $|\pm\langle 0|1\rangle|^2 = 0$, which implies that the algorithm outputs state $|0\rangle$ with probability 1. On the other hand, if the function is balanced we have:

- case 1 - $f(0) = 0, f(1) = 1$:

$$|\psi_3\rangle = \frac{|+, 0\rangle - |+, 1\rangle + |-, 1\rangle - |-, 0\rangle}{2}$$

$$= |+\rangle\,|-\rangle - |-\rangle\,|-\rangle = \left(|+\rangle - |-\rangle\right)|-\rangle = |1\rangle\,|-\rangle.$$

- case 2 - $f(0) = 1, f(1) = 0$:

$$|\psi_3\rangle = \frac{|+, 1\rangle - |+, 0\rangle + |-, 0\rangle - |-, 1\rangle}{2}$$

$$= -|+\rangle\,|-\rangle + |-\rangle\,|-\rangle = \left(-|+\rangle + |-\rangle\right)|-\rangle = -|1\rangle\,|-\rangle.$$

After the measurement, we obtain $|\pm\langle 1|0\rangle|^2 = 0$, $|\pm\langle 1|1\rangle|^2 = 1$, which implies that the outcome of the algorithm is $|1\rangle$ with probability 1.

The Deutsch algorithm thus perfectly distinguishes the two cases with only one query to U_f. As we saw earlier, classically we need to make at least two calls to $f(\cdot)$ to determine if it is constant or balanced. Therefore, the quantum algorithm achieves an advantage in the number of queries to the function. The question now is how does the Deutsch algorithm achieve this advantage? To explain this, we would need to go deeper into the theory of quantum mechanics which is out of the scope of this thesis. We will thus just say that the key ingredient in obtaining this speed-up was the proper combination of the Hadamard gates that enabled the algorithm to take advantage of two quantum effects known as *superposition* and *interference*.

Similar quantum algorithms   Some other quantum algorithms can be seen as a generalization of the Deutsch algorithm to functions with more inputs/outputs and different promises on the possible outputs. For example, the *Deutsch/ Jozsa algorithm*[DJ92] generalizes Deutsch's algorithm by taking a function $f : \{0, 1\}^n \to \{0, 1\}$ as input while keeping the promise on the output unchanged, namely, that the function could be either constant or balanced. The goal of the algorithm then is to decide whether the function is constant or balanced. Another example is so-called *Simon's algorithm*[Sim97] that takes as an input a function $g : \{0, 1\}^n \to \{0, 1\}^n$ with a promise that there exists $\mathbf{x}, \mathbf{y}, \mathbf{s} \in \{0, 1\}^n$ for which $g(\mathbf{x}) = g(\mathbf{y})$ if and only if $\mathbf{x} = \mathbf{y} + \mathbf{s}$. The goal of the algorithm is to find $\mathbf{s}$ by querying $g(\cdot)$ the smallest possible number of times. These algorithms use ideas similar to one of the Deutsch algorithms and exploit quantum effects such as superposition and interference to obtain an advantage over classical algorithms. In the case of the two above-described algorithms, the advantage is, in fact, rather high, more precisely, it is exponential in the number of queries in comparison to the best classical algorithms.

Shor's algorithm   One of the most prominent examples of more evolved quantum algorithms that follow the same line of reasoning as the previously introduced Deutsch algorithm is the so-called *Shor's algorithm*[Sho94]. Its significance lies in its ability to solve the so-called *integer factorization prob-*

lem and the *discrete logarithm problem* efficiently[6]. As these computational tasks are used as bases of a significant amount of the current state-of-the-art cryptographic protocols, Shor's algorithm gave the initial motivations for the development of a whole new area of cryptography, known as the *post-quantum cryptography/quantum-safe cryptography*, which aims to design protocols resistant to attacks via quantum devices. Most of the current proposals for quantum-safe protocols thus aim to avoid the attack via Shor's algorithm by not introducing promises that can potentially be exploited by Shor's algorithm.

Non-promise problems

We will now present a quantum algorithm that, in contrast to the already described ones, does not require any particular promise on the output of the function that is taken as the input to the algorithm. In return, the quantum algorithm provides a smaller but still significant speed-up over classical algorithms solving the same computational task. It was originally introduced as an algorithm for database search but then found a wide variety of applications in different areas, including cryptography. In this thesis, we use this algorithm as the basis of the hybrid quantum-classical approach introduced in Chapter 3.

Grover's algorithm   What we refer to as *Grover's algorithm* (also known as Grover's search) is a quantum algorithm that solves the following computational task. Given a function $f : \{0,1\}^n \rightarrow \{0,1\}$, find an $\mathbf{x}^* \in \{0,1\}^n$ such that $f(\mathbf{x}^*) = 1$. For now, we will assume there exists only one such $\mathbf{x}^*$ that yields $f(\mathbf{x}^*) = 1$, and the goal of the algorithm is to find it.

To see the promised advantage of Grover's search, let us first see how many classical calls to $f(\cdot)$ we need to make in order to find $\mathbf{x}^*$ (assuming all the values of $\mathbf{x} \in \{0,1\}^n$ we used for function calls were different). It turns out that, in the classical setting, the number of queries is lower-bounded by 2^n, which implies that, in the worst case, we would need to call $f(\cdot)$ on every possible input to be sure that we will find the desired $\mathbf{x}^*$. In the quantum setting, on the other hand, the number of calls to V_f is $O(2^{n/2})$, which is the

---

[6]We will explain what we mean by *efficient* in the following subsection

square root of the number of queries that is necessary in the classical case. We will now explain where the speed-up, obtained by Grover's algorithm, comes from.

The quantum circuit corresponding to the Grover algorithm is presented in figure 1.22. First, let us observe that in the case of Grover's algorithm, we
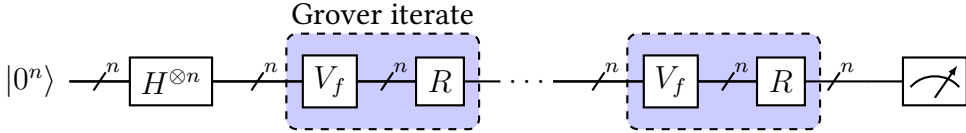
Figure 1.22: Grover's algorithm.

encode $f(\cdot)$ via the quantum oracle V_f that we define as follows:

$$V_f \ket{\mathbf{x}} = (-1)^{f(\mathbf{x})} \ket{\mathbf{x}},$$

where $\mathbf{x} \in \{0,1\}^n$. Let us then observe what happens at each step of Grover's algorithm. After applying the Hadamard gate on n input qubits, the algorithm obtains the following state:

$$\ket{\psi_1} = \frac{1}{2^{n/2}} \sum_{\mathbf{x}\in\{0,1\}^n} \ket{\mathbf{x}},$$

which is basically a uniform superposition of all the basis states of the standard basis. After obtaining the superposition, the algorithm applies the so-called Grover iterate that comprises of the quantum oracle V_f and the so-called *difussion operator*, R. The diffusion operator is defined as follows;

$$R = 2\ket{\psi_1}\bra{\psi_1} - I,$$

where $\ket{\cdot}\bra{\cdot}$ denotes the outer product, and I the identity matrix of rank n. The Grover iterate is then repeated $2^{n/2}$ times. The purpose of each repetition is to put the original superposition of states, $\ket{\psi_1}$, a step closer to the solution $\ket{\mathbf{x}^*}$.

To see that, let us observe what happens during each repetition of the Grover iterate. We denote by $\ket{\phi} \in \mathbb{C}^{2^n}$ a state orthogonal to the solution $\ket{x^*}$, namely:

$$\ket{\phi} = \frac{1}{\sqrt{2^n - 1}} \sum_{\mathbf{x}\neq\mathbf{x}^*} \ket{\mathbf{x}}.$$

In each iteration, the algorithm first applies V_f to the state $|\psi_i\rangle$, $i \in [n]$, obtained in the previous iteration, reflecting the state $|\psi_i\rangle$ around state $|\phi\rangle$,[7] and then applies the diffusion operator, R, to reflect the newly obtained state around $|\psi_1\rangle$. Combining the two reflections occurring in the same plane, the algorithm thus rotates the original state of uniform superposition, $|\psi_1\rangle$, toward the solution of the problem.

Let us now take $\theta \in [0, 2\pi]$ to be the angle between $|\psi_1\rangle$ and $|\phi\rangle$. Given that $V_f |\psi_1\rangle$ is obtained as a reflection of $|\psi_1\rangle$ around $|\phi\rangle$, the angle between $|\psi_1\rangle$ and $V_f |\psi_1\rangle$ is 2θ. By reflecting the state $V_f |\psi_1\rangle$ around $|\psi_1\rangle$, we obtain the state $RV_f |\psi_1\rangle$, which is at the angle distance 2θ from the original state $|\psi_1\rangle$. We thus observe that, after each application of Grover iterate, the initial uniform superposition $|\psi_1\rangle$, as well as the state $|\phi\rangle$, are rotated for angle 2θ toward the solution $\mathbf{x}^*$. Since we know that the state $|\phi\rangle$ was originally at the angle distance $\pi/2$ from the state $|\mathbf{x}^*\rangle$, we can calculate the expected number of repetitions of Grover iterate after which Grover's algorithm rotates state $|\psi_1\rangle$ as close as possible to state $|\mathbf{x}^*\rangle$. We then obtain the expected number of repetitions $r \in \mathbb{N}$ given as $r \approx \frac{\pi}{4\theta}$. Given that $|\mathbf{x}^*\rangle$ and $|\phi\rangle$ are orthonormal, $|\psi_1\rangle$ can be written as:

$$|\psi_1\rangle = \cos\theta |\phi\rangle + \sin\theta |\mathbf{x}^*\rangle,$$

so we obtain $\sin\theta = \frac{1}{2^{n/2}}$, which for small θ implies $\theta = O(2^{n/2})$. We thus know that after approximately $2^{n/2}$ applications of Grover iterate, we will obtain the solution $|\mathbf{x}^*\rangle$ with a high probability. This result finally yields that after $O(2^{n/2})$ queries to the unitary V_f, we will find $|\mathbf{x}^*\rangle$ with a high probability.

Let us now observe what would happen in the case when there is more than one $|\mathbf{x}^*\rangle$ that yields $f(\mathbf{x}^*) = 1$, and let's say we know the number of solutions is equal to some $m \in \mathbb{N}$. We then modify Grover's algorithm so that the initial superposition of all the basis states iteratively rotates toward the superposition of all the solution states. We thus obtain an algorithm that finds any of these solutions in the expected number of calls equal to V_f that is equal to $O(\sqrt{\frac{2^n}{m}})$.

---

[7]We note here that the state $|\phi\rangle$ is introduced for the sake of clear analysis, but the algorithm does not calculate it at any point.

Amplitude amplification The amplitude amplification can be seen as a generalized version of the previously introduced Grover algorithm. In contrast to other algorithms that we presented so far, it does not aim to solve a particular computational task but to improve the success probability of a certain algorithm. In Figure 1.23 we present a quantum circuit that performs amplitude amplification.
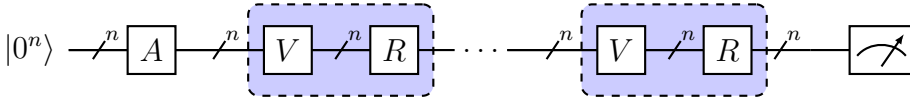
Figure 1.23: Amplitude amplification.

As it can be shown that any classical circuit can be simulated by a quantum circuit with at most polynomial overhead, every classical algorithm can be efficiently simulated via a quantum one. Therefore, without loss of generality, we can assume that the algorithm $\mathcal{A}$, whose success probability we would like to boost, is quantum. Nevertheless, we assume that it does not perform a measurement at the end and, as such, can be presented using a unitary matrix we denote by A. If we denote by $p \in [0, 1]$ the success probability of $\mathcal{A}$, the corresponding unitary can be presented as:

$$A \left| 0^n \right\rangle = \sqrt{p} \left| \mathbf{x}^* \right\rangle + \sqrt{1 - p} \left| \phi \right\rangle,$$

where $\left| \mathbf{x}^* \right\rangle$ is a projection on the subspace $\mathcal{S} \subseteq \mathbb{C}^{2^n}$ spanned by the solutions of the task that $\mathcal{A}$ aims to solve, and $\left| \phi \right\rangle$ is the projection on the subspace orthogonal to $\mathcal{S}$.

Let us now analyze the algorithm presented in Figure 1.23. The initial state, which we denote by $\left| \psi_1 \right\rangle$, is obtained by applying A to the n-qubit state $\left| 0^n \right\rangle$. This state is then iteratively rotated toward state $\left| \mathbf{x}^* \right\rangle$ using reflection operators V and R. More precisely, in each iteration, the algorithm first applies V to the state $\left| \psi_i \right\rangle$, where V is defined as:

$$V = I - 2\Pi.$$

The unitary V is then followed by the unitary R defined as:

$$R = \left| \psi_1 \right\rangle \left\langle \psi_1 \right| - I.$$

Similarly to Grover's search, it can be shown that each iteration rotates the initial state for a constant angle given as twice the size of the initial angle difference between $|\psi\rangle$ and $A|0^n\rangle$. It can also be shown that after $O(1/\sqrt{p})$ steps, the algorithm $\mathcal{A}$ finds a solution to the computational task with a probability close to 1. We remark here that, in the classical setting, we can also boost the probability of success of an algorithm. The expected number of calls to do so, however, is $\Theta(1/p)$. This gap will be crucial for the speed-ups we obtained in this thesis when we applied Grover's search and amplitude amplification to solve certain computational problems.

Quantum search As we already remarked, amplitude amplification can be seen as a generalization of Grover's algorithm where the initial state of uniform superposition in Grover's search is replaced with $A|0^n\rangle$, and the operators V and R are generalized to operate on the subspaces. A further step in the generalization is obtained through the framework of the quantum walks which also aim to solve search problems in the setting similar to the ones from Grover's search and amplitude amplification.

We further remark that the speedup obtained by either of these algorithms, namely, Grover's search, the amplitude amplification, and quantum walks, is expected to be quadratic. As such, the speed-up appears to be less impressive than the one obtained by algorithms such as Deutsch-Josza, Shor's algorithm, etc. Nevertheless, we observe that the computational tasks that these algorithms aim to solve come 'without promises', in contrast to the problems for which we obtain exponential speed-ups. These algorithms thus can be seen as a more generic tool that can be used as a search tool in a wide range of domains.

Oracles and *queries*

An *oracle* is an abstract representation of an algorithm, commonly encountered in the analysis of algorithms, used for presenting an algorithm as a *black box*. When modeling algorithms as oracles, we do not take into account what is the internal logic of an algorithm or its running time, but just which function the algorithm evaluates. This can be rather useful in settings similar to what we presented in the analysis of Deutsch's algorithm, where the focus of the analysis is not on the exact running time, but on the comparison of the

number of calls to the function. We commonly refer to these calls as *queries*, and to the corresponding number of calls as the *query complexity*. Query complexity is a rather common measure of algorithms efficiency, especially in the quantum setting.

The major differences between classical and quantum circuit models

So far, we described two models, drawing parallels between their basic units as well as between the measures of their efficiency. Nevertheless, we did not say which of the two models we should choose and why we should do so. From the discussion so far, it appears that the quantum computational model is more general, and we will discuss this observation in short. Does it mean that it is necessarily better? Does it have any computational advantage over the classical one? To answer these questions, we will highlight the differences between the two models and try to understand if any of these provide us with a computational advantage.

Bits versus qubits   The first difference we notice is in the number of potential states that classical and quantum bits can store. Namely, while classical bits are points in $\mathbb{F}_2$, quantum bits can take one of the infinitely many values on the unit-distance surface sphere in $\mathbb{C}^2$. It gives the impression that infinitely many classical data can be stored within just a single qubit with appropriate encoding. Nevertheless, by the laws of quantum mechanics, this type of encoding is not possible. As we already remarked, to access information stored in a qubit at the end of a calculation, we need to collapse it into a classical bit by measuring it. But, as a measurement is a destructive process that enables us to learn only one bit of classical information per qubit, the rest of the information is inevitably lost. To make use of the quantum bits, we need to find a way to exploit their quantum nature while processing the data, but before measuring it.

Non-reversible versus reversible gates   A major difference between the two gate types, namely, classical and quantum ones, is in its *reversibility*. While classical gates are not necessarily reversible, all quantum gates are reversible by the postulates of quantum mechanics. Non-reversible classical gates, however, can be 'turned into' reversible ones by adding additional

entries and exits to these gates. Moreover, each classical gate can be simulated via a quantum gate with at most polynomially many additional inputs/outputs. Quantum gates, on the other hand, are not necessarily efficiently simulatable by the classical ones. Though certain gate sets, such as Clifford gates, can be simulated by classical gates efficiently, this is not true for arbitrary quantum gate sets. Universal quantum gate sets, for example, cannot be efficiently simulated on classical devices. The question then is whether this difference gives us any computational advantage.

The (potential) computational advantage To answer this question, we first observe that, since qubits can be seen as a generalization of bits, and quantum gates as a generalization of classical ones, all classical circuits can be simulated by the quantum ones. Furthermore, this simulation can be done efficiently. The question then is how we can use this generalization to our advantage. The first step toward it would be to use some inherently quantum effects, such *superposition*, *interference*, and *entanglement*, to our benefit. The examples of quantum circuits (i.e. quantum algorithms) given in this section illustrate this approach. The second step would be to actually implement these algorithms and compare their performances with the classical algorithms in a real-life setting. With the current state of the development of quantum devices, however, this comparison is not possible on a larger scale since large-scale quantum devices, which are supposed to demonstrate the full advantage of using quantum algorithms, are not (yet) produced. Whether these devices will be ever produced and how much advantage they would give remains an open question.

1.4   Computational complexity

The computational complexity can be observed as an even higher level of an abstract view of the computation. It deals with the proper definition of the computational tasks, which we commonly refer to as *computational problems*, and determining their difficulty. The problems are then categorized into what we know as complexity classes that form hierarchies of problems with respect to their difficulty. In the following subsection, we will explain some of the notions of this field that are relevant to this thesis.

Computational problems

A computational problem can be broadly defined as any computational task that can be solved using an algorithm. The input of the algorithm thus corresponds to the problem definition, while its output describes a desired solution to the problem. Based on the possible desired outcome, we divide problems into so-called *decision problems* and search problems. In the case of decision problems, the desired output of the algorithm is a binary answer yes or no (equivalently, 1 or 0). In the case of search problems, the desired output is commonly given as a string representing the answer. In the following chapters, we will give examples of both types of problems.

Instance of a problem   A concrete instantiation of the problem, which we refer to as an *instance* of the problem, is given via particular input to the algorithm and the desired output. Each problem thus can be seen as a set of all instances that correspond to the same computational task. Instances of particular interest in the study of computational complexity are those whose size grows beyond bound as these reflect how the problem's difficulty scales up with the size of the input.

Computational complexity   The *complexity* of a computational problem is taken as a fundamental measure of the problem's difficulty. It is determined as the efficiency of the best algorithm solving an instance of the problem when the instance size grows beyond bounds. The algorithm's efficiency is then commonly measured as either the running time or the memory con-

sumption of the algorithm solving this instance, and the efficiency is expressed as a function of the input size. In this thesis, we primarily rely on the running time as a measure of the algorithm's efficiency.

Complexity classes

Given the problem's difficulty, i.e. its computational complexity, each problem can be categorized into a complexity class, which is defined with respect to a computational model. We will now define computational classes that are relevant in the context of this thesis.

> **Definition 1.4.1** (Deterministic polynomial time, **P**). *The deterministic polynomial time class, **P**, consists of decision problems that can be solved by a deterministic algorithm running in time polynomial in its input size.*

The complexity class **P** is considered to be of particular importance for the theory of computation as it appears to encompass computational problems whose solution is computationally feasible using classical computational devices. The class **NP**, which we introduce next, on the other hand, is believed to contain problems that are computationally infeasible for both classical and quantum devices. The exact relation between the two classes, however, is still an open question in complexity theory and one of the biggest open problems in theoretical computer science.

> **Definition 1.4.2** (Non-deterministic polynomial time, **NP**). *The non-deterministic polynomial time class, **NP**, consists of decision problems that can be solved by a non-deterministic algorithm running in time polynomial in its input size.*

Decision versus search problems

So far, we discussed the complexity of decision problems only. We observe here that finding a solution to a search problem directly implies that one exists while finding no solution implies that none exists. Therefore, solving a search problem is at least as difficult as solving its decision version. Moreover, it can

be shown that for **NP**-complete problems, the opposite is also true and that solving a decision problem implies solving its search version. For proof of this statement, see, for example, [AB09]). The decision and search versions are thus equivalent for **NP**.

Hard and *complete* problems

What we refer to as *hard problems* are the problems that are at least as difficult as any other problem in a given computational class. To define these more formally, we introduce the notion of reduction, formally given in 1.4.3.

> **Definition 1.4.3** (Karp reduction, $\preceq$). *We say that a decision problem* P_1 *is polynomial-time Karp reducible (or simply polynomial-time reducible) to a decision problem* P_2*, i.e.* $P_1 \preceq_p P_2$*, if there is a polynomial-time algorithm* A *that on input* $\mathbf{x} \in \{0, 1\}^*$ *outputs* $A(\mathbf{x})$ *that is a solution to* P_2 *if and only if* $\mathbf{x}$ *is a solution to* P_1*.*

The above-given definition, in fact, implies that if there exists an algorithm A that acts as a polynomial-time reduction from P_1 to P_2, then P_2 is at least as difficult as P_1. To see that, let us assume that we have an algorithm A_2 solving P_2 efficiently. We can then use the reduction algorithm A to transform an instance of P_1, given as the input $\mathbf{x} \in \{0, 1\}^*$ to an algorithm solving P_1, into an input of A_2 given as $A(\mathbf{x})$. We can then use A_2 to decide if $A(\mathbf{x})$ is a solution to P_2 and, consequently, if $\mathbf{x}$ is a solution to P_1. A_2 thus can be used for solving P_1 with at most polynomial overhead, which further implies P_1 cannot be more difficult than P_2.

We can now extend this reasoning to the whole class and define *hard* and *complete* problems, as given in Definition 1.4.4.

> **Definition 1.4.4** (Hard and complete problems). *We say that a decision problem A is hard for a class C if* $B \preceq_p A$ *for all* $A \in C$*. If A is hard for* C *and* $A \in C$*, then we call A a complete problem for C.*

Problems that are of particular interest for this thesis are hard problems for the class **NP**, also referred to as **NP**-hard. Some famous examples of these problems are the decision version of *traveling salesman problem*, *boolean satisfiability problem*, *subset sum problem*, etc.

1.5   Cryptography

Cryptography can be broadly defined as the study of secure communication. More precisely, it studies computational problems that are believed to be difficult to solve and uses these to build cryptographic protocols that protect our data and achieve secure communication. Later in the chapter, we will give a more precise definition of what to mean by "secure" and give different examples of how computational problems can be used to protect data through cryptographic protocols. For now, we would like to introduce the two major subfields known as the *private-key* (or *symmetric*) cryptography and the *public key* (or *asymmetric*) cryptography. The difference between the two subfields can be illustrated through the so-called *data encryption* protocol, presented in Figure 1.24 and Figure 1.25, where the former corresponds to the symmetric key setting and the latter one to the asymmetric key setting.
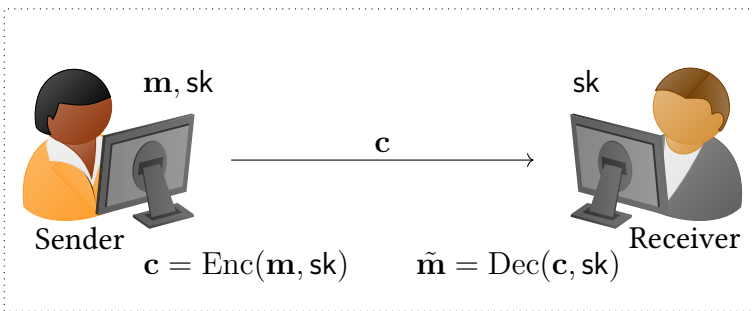
Figure 1.24: Symmetric key encryption
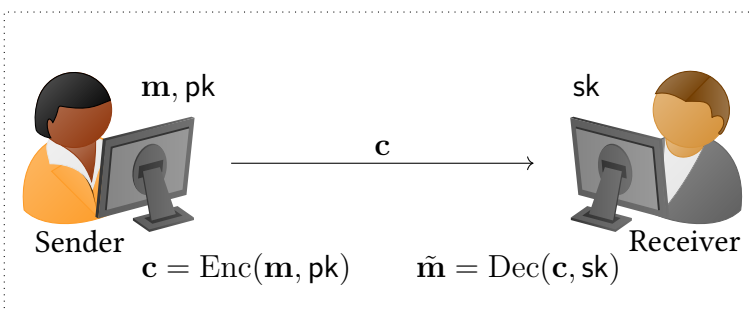
Figure 1.25: Asymmetric key encryption

The goal of this protocol is to enable secure transmission of a message $\mathbf{m} \in \{0, 1\}^*$, also known as the *plaintext*, between the two parties known

as the *sender* and the *receiver*, also known as the *honest parties*. The protocol starts by the *key generation* in which, depending on the setting, the function $\mathrm{KGen} : \mathbb{N} \to \{0,1\}^*$ produces either the matching *public key* and *secret key* pair, $(\mathsf{sk}, \mathsf{pk}) \in \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}, n \in \mathbb{N}$, or simply the secret key $\mathsf{sk} \in \{0,1\}^n, n \in \mathbb{N}$. Again depending on the setting, the sender then obtains either the public key, pk, or the secret key, sk, and the receiver obtains the secret key, sk. In the secret key setting, the sender then uses its secret key to *encrypt* the message $\mathbf{m}$ using an encryption function (algorithm) $\mathrm{Enc} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$. The function takes as an input the message $\mathbf{m}$, the secret key sk, and then outputs the encrypted message, also known as the *ciphertext*, $\mathbf{c} \in \{0,1\}^*$. Similarly, in the asymmetric key setting, the sender uses the encryption function $\mathrm{Enc}(\cdot)$ to encrypt the message $\mathbf{m}$ by taking as an input $\mathbf{m}$ and, this time, the public key pk, and then outputting the ciphertext $\mathbf{c}$. The ciphertext is then sent to the *receiver* through a (potentially insecure) public channel. Upon receiving the ciphertext $\mathbf{c}$, the receiver *decrypts* it using the function (algorithm) $\mathrm{Dec} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$. The algorithm takes as an input the ciphertext $\mathbf{c}$, the secret key sk and outputs the deciphered message, $\tilde{\mathbf{m}} \in \{0,1\}^*$. If the sender's key matches the receiver's key, the receiver obtains the original plaintext, $\mathbf{m}$. Namely, we should have:

$$\tilde{\mathbf{m}} := \mathrm{Dec}(\mathrm{Enc}(\mathbf{m}, \mathsf{pk}), \mathsf{sk}) = \mathbf{m}.$$

The choice of using a symmetric or asymmetric approach extends beyond encryption protocols, and the difference in symmetry introduces a rather different approach to the analysis of the protocols and the underlying computational problems in these two settings. In this thesis, we focus on the design and analysis of public key protocols.

1.5.1 Security

In this section we define more formally what we mean by "secure" and how we can prove that a certain cryptographic model is secure. To do so, we will introduce models of potential *attackers* whose goal would be to either partially or fully steal the secret information, temper the data, or perform any adversarial action.

Perfect security What we call the *perfect security* characterizes an ideal setting in which a cryptographic protocol reveals no information even to a dishonest party, also known as an *adversary* or an *attacker*, with unbounded computational power. In the *data encryption setting*, for example, we say that a protocol is *perfectly secure* if observing a ciphertext (or many of them) do not reveal any information on the message being sent even in the presence of an all-powerful adversary. This definition seems rather natural and can be further extended to other cryptographic protocols. Nevertheless, it turns out that protocols satisfying this notion of security are inefficient. Luckily, this notion is not really mandatory in the real-life setting in which we can expect that the adversary is not unbounded. We thus introduce yet another notion of security, known as *computational security*.

Computational security The *computational security* can be seen as a better suited to the real-life setting alternative to the perfect security notion. We call a protocol *computationally secure* if it reveals information with a very small probability (we will specify in short what we mean by "very small") to any computationally bounded adversary rather than unbounded ones. This relaxation allows us to assume that certain computational tasks are difficult for a bounded adversary and, as such, can be used as bases for computationally secure protocols. In this thesis, we will use computational security as a main criterion for evaluating the protocol's security.

Security definitions To prove claimed (computational) security of a protocol, we first introduce *security definitions* (also known as *security notions*) that establish the *security goal*, i.e. criterion that needs to be satisfied in order to claim security. Though security goals can be rather different and should be chosen according to the protocol at hand, there exist standard goals that are considered to be good practices. In this thesis, we will use these for verifying the security of our protocols. To give a full security definition, in addition to the security goal, we also need to specify an *attacker model*, which determines the computational power of an adversary. As we already mentioned, in this thesis we are primarily interested in computational security, rather than the perfect one, so we will consider only the computationally bounded attackers.

Hardness assumptions and reductions   Once the security definition is established, we can decide which *hardness assumption*, i.e. the assumption on the difficulty of a problem underlying a protocol, to use in order to claim that a certain security definition is satisfied. To show that certain protocol is computationally secure, we then commonly rely on *security reductions*. These basically show that breaking a protocol implies solving a computational problem that we believe is difficult.

Security parameter   To give a more precise definition of "small" probability and an "efficient" adversary, we first define the so-called *security parameter*. It is commonly denoted by n, and it is used for expressing the computational resources necessary for attacking the protocol, as well as the success probability of an attack. For example, for an encryption scheme, the security parameter is given as the length of the secret key chosen in the key generation process. The running time of an algorithm that aims to recover the secret key, with a certain success probability, thus can be expressed as the function of the length of the secret key. We then define an *efficient adversary* as an algorithm running in time $\mathsf{poly}(n)$, and we describe a "very small" success probability through the notion of *negligible function*, defined as follows.

Definition 1.5.1 (Negligible function, $\mathsf{negl}(n)$). *A function $f : \mathbb{N} \to \mathbb{N}$ is negligible, denoted as $\mathsf{negl}(n)$, if for every positive polynomial $\mathsf{poly}(n)$, there exists $n_0 \in \mathbb{N}$ such that $\forall n > n_0$, the following is satisfied:*

$$f(n) < 1/\mathsf{poly}(n).$$

Asymptotic and concrete security   In this thesis, we will be primarily interested in the *asymptotic complexity* of the attacks corresponding to the limit case where the security parameter grows beyond bound. In practice, however, the so-called *concrete security* estimates are obtained in the non-asymptotic regime, which allows us to evaluate more accurately what is the security of a certain protocol in a real-life setting. We use the concrete approach when calculating the concrete security parameters of our protocol design in Chapter 4. In the concrete setting, we say that a protocol is (t, ϵ)-secure if an adversary running in time at most t succeeds to break the protocol with probability at most ϵ, where both t and ϵ are some constant values.

Broadly speaking, "breaking" a protocol refer to learning enough secret information so that we can consider that the scheme is not secure anymore. In the case of encryption protocol, for example, learning the secret key would reveal information about all the future messages being sent via the same protocol. This effectively corresponds to breaking the scheme's security.

Cryptanalysis

The goal of cryptanalysis is to estimate the computational resources necessary for attacking a certain cryptographic protocol. The attack can aim either at breaking the protocol construction, i.e. invalidating the claims about the security of the construction, or breaking the problem that is used as the protocol basis, i.e. invalidating the assumption on the problem's difficulty. While the security of the protocols is commonly guaranteed through the so-called *security proofs* that we will introduce in short, the security of the assumption is based on the claimed difficulty of the computational problem.

In the setting of public-key cryptography, i.e. in our setting, we are primarily interested in the so-called *average-case* complexity of computational problems. It is estimated as the asymptotic running time of the best algorithms solving problem instances sampled from a particular distribution of inputs. Namely, we estimate the running time of the given algorithm solving an average-case instance of the computational problem in the limit case when the security parameter grows beyond bound. We thus obtain an estimate of how the problem behaves on average, rather than in the worst case, which is considered to be the most relevant in a cryptographic setting.

Classical and quantum setting   As we have already explained, in this thesis, we are concerned with two models of computation describing general-purpose computational devices, one corresponding to classical computing and one corresponding to quantum computing. As either of these devices can potentially be used to run algorithms that attack cryptographic protocols, we will calculate the expected running time of these algorithms and base our security estimates on these results.

Post-quantum cryptography   As briefly mentioned in the section on the computational models, protocols that are believed to be secure against quantum attacks (and implicitly also against classical ones) are commonly referred to as *post-quantum*, *quantum safe*, or *quantum resistant*. The subfield of cryptography that analyzes problems that are believed to be post-quantum, and builds protocols based on these,e is commonly referred to as post-quantum cryptography.

In this thesis, we are focused on the design of a protocol based upon computational problems derived from decoding linear codes, originally introduced in coding theory, and hence referred to as the *code-based* problems. These problems are believed to be quantum resistant and, as such, they represent one of the major interests of post-quantum cryptography. One of the central problems in this area, and the focus of this thesis, is the so-called *syndrome decoding problem*, derived from the syndrome decoding method introduced in the previous chapter. We will give a more precise definition of this problem in the next chapter. The first known example of a public key protocol based on this problem is the so-called *McEliece encryption scheme*, introduced in [McE78]. So far, the scheme resisted all the cryptanalytic efforts, both classical and quantum, and, as such, it is regarded as one of the most prominent candidates for public key encryption schemes in the post-quantum era.

1.5.2   Digital signature schemes

Digital signature schemes are public key protocols that enable proving message *authenticity* (also known as message *integrity*). In Figure 1.26, we present a basic setting in which the first party, also known as the *signer*, signs a message $\mathbf{m} \in \{0,1\}^*$, and the second party, also known as the *verifier*, verifies its authenticity.
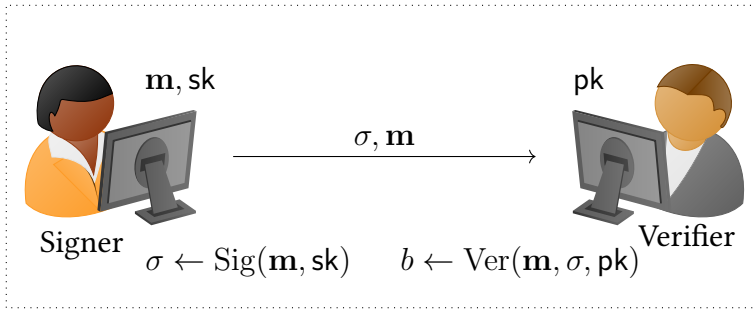
Figure 1.26: **Digital signature scheme**

The protocol proceeds as follows. It starts with the *key generation*, in which the signer calls function $\mathrm{KGen} : \mathbb{N} \to \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}$ to produce the matching *public key* and *secret key* pair, $(\mathsf{sk}, \mathsf{pk}) \in \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}, n \in \mathbb{N}$. Once the keys are generated, the signer keeps the secret key to itself and broadcasts the public key. The signer then signs a message of its choice, $\mathbf{m} \in \{0,1\}^*$, using signing algorithm $\mathrm{Sig} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^l, l \in \mathbb{N}$. Namely, it calls the signing algorithm on the chosen message and the secret key and thus obtains the so-called *signature*, $\sigma := \mathrm{Sig}(\mathbf{m}, \mathsf{sk})$. The signer then sends the pair $(\sigma, \mathbf{m})$, commonly referred to as the *signed message*, to the verifier. Upon receiving the signed message, the verifier checks its validity using a verification algorithm, $\mathrm{Ver} : \{0,1\}^* \times \{0,1\}^l \times \{0,1\}^n \to \{0,1\}$. More precisely, using the message $\mathbf{m}$, the signature σ, and the public key pk as the inputs of the verification algorithm, the verifier calculates its response as $b = \mathrm{Ver}(\mathbf{m}, \sigma, \mathsf{pk})$. If the message is authentic, meaning that it is chosen by the signer and has not been modified during the transmission, the verifier accepts the signature with overwhelmingly high probability and its verification algorithm returns $b = 1$. Otherwise, the verifier rejects the signature with overwhelmingly high probability and its verification algorithm returns $b = 0$.

In Definition 1.5.2, we present the digital signature scheme more formally. Our definition corresponds to the one given in [KL14].

Definition 1.5.2 (Digital signature scheme, DSS). *A digital signature scheme, Σ, is a public key protocol comprising of the following three algorithms running in time $\mathsf{poly}(n)$:*

- KGen : $\mathbb{N} \rightarrow \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}$ - *a probabilistic algorithm that takes as an input $1^n \in \mathbb{N}$ and outputs a pair of matching secret key and public key pair,* $(\mathsf{sk}, \mathsf{pk}) \in \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}, n \in \mathbb{N}$, *obtained as*

$$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{KGen}(1^n),$$

  where 1^n denotes n in unary representation;

- Sig : $\{0,1\}^* \times \{0,1\}^n \rightarrow \{0,1\}^l, l \in \mathbb{N}$ - *a probabilistic algorithm that takes as an input a message $\mathbf{m} \in \{0,1\}^*$ and the secret key sk, and it outputs the signature $\sigma \in \{0,1\}^l$ obtained as*

$$\sigma \leftarrow \mathrm{Sig}(\mathbf{m}, \mathsf{sk});$$

- Ver : $\{0,1\}^* \times \{0,1\}^l \times \{0,1\}^{\mathsf{poly}(n)} \rightarrow \{0,1\}$ - *a deterministic algorithm that takes as an input signature, σ, the message $\mathbf{m}$ and the public key, pk, and outputs $b \in \{0,1\}$. Namely, it outputs*

$$b \leftarrow \mathrm{Ver}(\mathbf{m}, \sigma), \mathsf{sk}), \mathsf{pk}).$$

We observe here that b is a random variable given over the choice of $(\mathsf{sk}, \mathsf{pk})$. It is required that, except with probability $O(1 - 1/\mathsf{negl}(n))$ over the choice of $(\mathsf{sk}, \mathsf{pk})$,

$$b = \mathrm{Ver}(\mathbf{m}, \mathrm{Sig}(\mathbf{m}, \mathsf{sk}), \mathsf{pk}) = 1$$

for every message $\mathbf{m}$.

(Un)forgability of digital signature schemes

To prove that a certain digital signature scheme is (computationally) secure, we rely on a security definition known as the *existentially unforgeable under an adaptive chosen-message attack* (EUF-CMA). To explain this notion on an intuitive level, let us call a valid signature any σ that originated from the signer, and call a forgery any valid signature, $\tilde{\sigma}$, that was not obtained from the signer. A EUF-CMA secure signature scheme guarantees that an adversary is not able to output a forgery even if it obtains signatures on many

other messages of its choice. To define EUF-CMA security more formally, let us first introduce the following security experiment.

Definition 1.5.3. *Let* $\Sigma = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Ver})$ *be a digital signature scheme, and* $n \in \mathbb{N}$ *be a security parameter. Let then* $A_1 : \mathbb{N} \times \{0,1\}^n \to \{0,1\}^*$, *and* $A_2 : \mathbb{N} \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^{l \cdot q} \to \{0,1\}^* \times \{0,1\}^l$ *be two polynomial time algorithms run by an adversary* A, *and let* l, q *be some constants in* $\mathbb{N}$.

For the given n, Σ, A, *a forging experiment,* $\mathrm{EUF\text{-}CMA}(n, \Sigma, A)$, *consists of the following steps:*

Experiment EUF-CMA(n, Σ, A)

1 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^n)$

2 : for $i \in [q]$:

3 : $\quad \mathbf{m}_i \leftarrow A_1(1^n, \mathsf{pk})$

4 : $\quad \mathcal{Q}.add(\mathbf{m}_i)$

5 : $\quad \sigma_i \leftarrow \mathsf{OSign}(\mathbf{m}_i)$

6 : $(\mathbf{m}, \sigma) \leftarrow A_2(1^n, \mathsf{pk}, \{\mathbf{m}\}_{i \in [q]}, \{\sigma_i\}_{i \in [q]})$

7 : if $\mathbf{m} \notin \mathcal{Q}$:

8 : $\quad$ return $\mathsf{Ver}(\mathbf{m}, \sigma, \mathsf{pk})$

9 : else:

10 : $\quad$ return 0

---

where $\mathsf{OSign}(\cdot)$ *is a signing oracle that on an input message* $\mathbf{m}_i \in \{0,1\}^*, i \in [q]$, *returns a signature of this message,* $\sigma_i \in \{0,1\}^l$.

We observe here that the outcome of this experiment, which we denote by $\mathrm{out}_{\mathrm{EUF\text{-}CMA}(n,\Sigma,A)}$, is a random variable given over the randomness in A_1 and A_2. We can now define the notion of *existentially unforgeability under an adaptive chosen-message attack*.

Definition 1.5.4 (Existentially unforgeable under an adaptive chosen-message attack (EUF-CMA))**.** *A digital signature scheme* $\Sigma = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Ver})$

> is existentially unforgeable under an adaptive chosen-message attack, i.e.
> EUF-CMA secure, if for any probabilistic polynomial-time adversaries A,
> there exists a negligible function, $\mathsf{negl}(n)$, such that:
>
> $$\Pr[\text{out}_{\text{EUF-CMA}(n,\Sigma,A)} = 1] \leq \mathsf{negl}(n).$$

This notion, originally introduced in [GMR88], is taken as the standard notion of security for digital signature schemes.

Before continuing on the topic of digital signature schemes, we would like to introduce the notion of (cryptographic) hash functions, which will be relevant in the context of the two types of digital signature constructions we will introduce afterward.

Hash functions

Informally speaking, a hash function can be observed as *compressing* function that takes a vector of arbitrary length and outputs a shorter vector of a fixed size. Given that the input is longer than the output, this function is necessarily non-injective. We thus say that it is always possible to find a *collision* in a hash function, meaning that we can always find at least one pair of vectors that map to the same output. A common requirement for a "good" hash function is that the collisions are as "balanced" as possible, meaning that the number of collisions per output is similar for all the outputs of the given hash function. The requirement naturally translates into the common security definition for the cryptographic hash functions that we will introduce in short. Let us now define a cryptographic hash function used in our protocol design, known as the *unkeyed* or *deterministic* cryptographic hash function.

> **Definition 1.5.5** ((Unkeyed) cryptographic hash function). *An unkeyed cryptographic hash function, $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^l$, is a deterministic polynomial time algorithm that on input $\mathbf{x} \in \{0,1\}^*$ outputs a string $\mathbf{y} \in \{0,1\}^l$ of a constant length $l \in \mathbb{N}$.*

We observe here that, by the introduced definition, a cryptographic hash function is inherently deterministic. In Chapter 4, we will show that the application of a deterministic hash function in the cryptographic setting can po-

tentially lead to new cryptographic attacks that are commonly unperceived by the standard cryptographic analysis.

Collision resistance A common security requirement for the cryptographic hash functions is given through the notion of *collision resistance*. To define it, let us introduce the following security experiment.

Definition 1.5.6. *Let* $\Gamma = (\mathrm{KGen}, \mathcal{H})$ *be a cryptographic hash function, and* $n \in \mathbb{N}$ *be a security parameter. Let then* $A_1 : \mathbb{N} \times \{0,1\}^n \rightarrow \{0,1\}^k \times \{0,1\}^k$, *be a polynomial time algorithm run by an adversary* A, *and let* k *be a constant in* $\mathbb{N}$. *For the given* n, Γ, A, *a collision-resistance experiment,* $\mathrm{Hash\text{-}Coll}(n, \Gamma, A)$, *consists of the following steps:*

Experiment Hash-Coll(n, Γ, A)
$1:$ $\mathrm{k} \leftarrow \mathrm{KGen}(1^n)$
$2:$ $\mathbf{x}_0, \mathbf{x}_1 \leftarrow A_1(1^n, \mathrm{k})$
$3:$ if $\mathbf{x}_0 \neq \mathbf{x}_1 :$
$4:$ return $\mathcal{H}(\mathbf{x}_0) = \mathcal{H}(\mathbf{x}_1)$
$5:$ else:
$6:$ return 0

As in the case of EUF-CMA notion, we observe that the outcome of this experiment, which we denote by $\mathrm{out}_{\mathrm{Hash\text{-}Coll}(n, \Gamma, A)}$, is a random variable given over the randomness in A_1. We can now define the notion of collision resistance.

Definition 1.5.7 (Collision resistance). *A cryptographic hash function,* $\mathcal{H}(\cdot)$, *is collision resistant if for any probabilistic polynomial-time adversary,* A, *there exists a negligible function,* $\mathsf{negl}(n)$, *such that:*

$$\Pr[\mathrm{out}_{\mathrm{Hash\text{-}Coll}(n, \Gamma, A)} = 1] \leq \mathsf{negl}(n).$$

Non-interactive commitment schemes

Broadly speaking, a *non-interactive commitment scheme* can be described as a cryptographic protocol that enables one to commit to a message by sending the corresponding committing value, also known as the *commitment*, that satisfies the following two properties:

- *hiding*: the commitment does not reveal any information about the message,

- *binding*: for a computationally bounded party that commits to a certain message $\mathbf{m}$ using commitment value $\mathbf{c}$, it should be infeasible to find another message $\tilde{\mathbf{m}} \neq \mathbf{m}$ that commits to the same $\mathbf{c}$.

More formally, a commitment scheme is given by the following definition.

Definition 1.5.8 (Commitment scheme, Comm). *A commitment scheme, K, is a cryptographic protocol comprising of the following algorithms running in time* $\mathrm{poly}(n)$*:*

- $\mathrm{Gen} : \mathbb{N} \to \{0,1\}^*$ - *a probabilistic algorithm that takes as an input* $1^n \in \mathbb{N}$ *and outputs public parameters* $\mathrm{params} \in \{0,1\}^*$ *where* 1^n *denotes the security parameter in unary representation;*

- $\mathrm{Com} : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^m, l, m \in \mathbb{N}$ - *a deterministic algorithm that takes as an input a message,* $\mathbf{m} \in \{0,1\}^n$*, and the random vector* $\mathbf{r} \in \{0,1\}^l$*, and then outputs the commitment value* $\mathbf{c} \in \{0,1\}^m$*, obtained as*

$$\mathbf{c} \leftarrow \mathrm{Com}(\mathbf{m}, \mathbf{r}).$$

The (computational) *hiding* and *binding* properties are then given via the following experiments.

Definition 1.5.9. *Let* $K = (\mathrm{Gen}, \mathrm{Com})$ *be a commitment scheme, and*

$n \in \mathbb{N}$ *be a security parameter. Let then*

$$A_1 : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^n \times \{0,1\}^n, \quad A_2 : \mathbb{N} \times \{0,1\}^m \times \{0,1\},$$

$$A_3 : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^m \times \{0,1\}^n \times \{0,1\}^l \times \{0,1\}^n \times \{0,1\}^l,$$

be three polynomial time algorithms run by an adversary A, and let l, and m be some constants in $\mathbb{N}$.

For the given n, K, A, a hiding experiment, $\mathrm{Hiding}(n, K, A)$*, consists of the following steps:*

Experiment Hiding(n, K, A)

1 : $\mathrm{params} \leftarrow \mathrm{Gen}(1^n)$

2 : $\mathbf{m}_0, \mathbf{m}_1 \leftarrow A_1(1^n, \mathrm{params})$

3 : $b \xleftarrow{\$} \{0,1\}$

4 : $\mathbf{r} \xleftarrow{\$} \{0,1\}^l$

5 : $\mathbf{c} \leftarrow \mathrm{Com}(\mathrm{params}, \mathbf{m}_b, \mathbf{r})$

6 : $\tilde{b} \leftarrow A_2(1^n, \mathbf{c})$

7 :   return $b = \tilde{b}$

A binding experiment, $\mathrm{Binding}(n, K, A)$*, consists of the following steps:*

Experiment Binding(n, K, A)

1 : $\mathrm{params} \leftarrow \mathrm{Gen}(1^n)$

2 : $(\mathbf{c}, \mathbf{m}_0, \mathbf{r}_0, \mathbf{m}_1, \mathbf{r}_1) \leftarrow A_3(1^n, \mathrm{params})$

3 :   if $\mathbf{m}_0 \neq \mathbf{m}_1$:

4 :   return $\mathrm{Com}(\mathrm{params}, \mathbf{m}_0, \mathbf{r}_0) = \mathrm{Com}(\mathrm{params}, \mathbf{m}_1, \mathbf{r}_1) = \mathbf{c}$

5 :   else :

6 :   return 0

We observe here that the outcomes of these experiments, which we denote

by $\text{out}_{\text{Hiding}(n,K,A)}$ and $\text{out}_{\text{Binding}(n,K,A)}$, are random variables given over the randomness in A_1, A_2, and A_3. We can now give a security definition that needs to be satisfied for every commitment scheme we claim secure.

Definition 1.5.10. *A commitment $K = (\text{Gen}, \text{Com})$ is secure if for any probabilistic polynomial-time adversaries A, there exits a negligible function,* $\text{negl}(n)$, *satisfying the following:*

$$\Pr[\text{out}_{\text{Hiding}(n,K,A)} = 1] \leq 1/2 + \text{negl}(n),$$

$$\Pr[\text{out}_{\text{Binding}(n,K,A)} = 1] \leq \text{negl}(n).$$

It is not hard to see that cryptographic hash functions are a natural choice for constructing commitment schemes in practice. In this thesis, we will thus assume that the commitment scheme used in our protocol design is instantiated using cryptographic hash functions.

DSS using hash functions

As we already briefly mentioned, the two most common approaches to designing digital signature schemes use hash functions in their constructions. The first approach is known as the *hash-and-sign paradigm*. It uses hash functions to reduce the length of the initial message to be signed. Though this approach is rather intuitive and provably secure, it is considered to be rather inefficient. The second approach is done in two steps: in the first step, we construct an *identification scheme*, which is then transformed into a digital signature scheme using the *Fiat-Shamir transformation* in the second step. In this thesis, we use the second approach, so we will now introduce it formally. To do so, let us start by defining an *identification scheme*.

Identification scheme   An identification scheme can be broadly defined as an interactive protocol that allows one party, known as the *prover*, to prove its identity, i.e. to *authenticate* itself, to another party, known as the *verifier*. In this thesis, we are interested in a particular variant of identification schemes called a Σ-*protocols*, or three-round protocols, depicted in Figure 1.27 and defined formally in Definition 1.5.11.
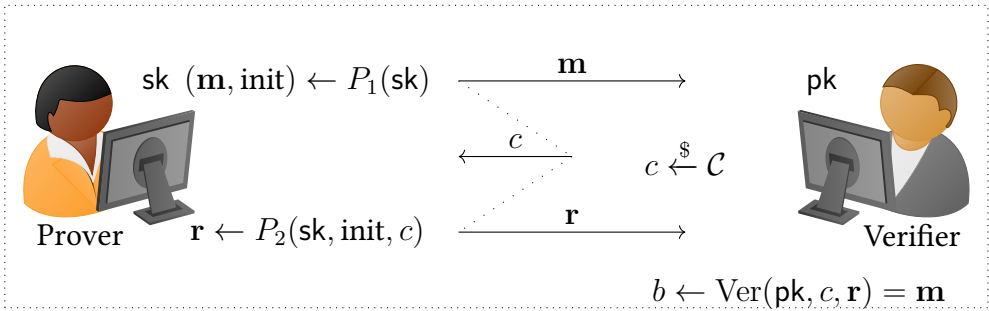
Figure 1.27: **Sigma (3-round) protocol**

As in the case of digital signature schemes, the protocol starts with the key generation. During the key generation, the prover calls function $\mathrm{KGen} : \mathbb{N} \to \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}$ to produce the matching *secret key* and *public key* pair, $(\mathsf{sk}, \mathsf{pk}) \in \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}, n \in \mathbb{N}$. Once the keys are generated, the prover keeps the secret key to itself and broadcasts the public key. The prover then runs $P_1 : \{0,1\}^n \to \{0,1\}^l \times \mathcal{S}, l \in \mathbb{N}$, to produce a message $\mathbf{m} \in \{0,1\}^l$ and obtains the initial state $\mathsf{init} \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states. The prover then sends the message $\mathbf{m}$ to the verifier. Upon receiving the message, the verifier samples a so-called *challenge*, $c \in \mathcal{C}$, where $\mathcal{C}$ is a finite set of symbols, and sends the challenge back to the prover. The prover then uses $P_2 : \{0,1\}^n \times \mathcal{S} \times \mathcal{C} \to \{0,1\}^m, m \in \mathbb{N}$, to produce the response $\mathbf{r} \in \{0,1\}^m$, and then sends the response to the verifier. Upon receiving the response, the verifier checks the consistency between the original message and the response using function $\mathrm{Ver} : \{0,1\}^{\mathsf{poly}(n)} \times \mathcal{C} \times \{0,1\}^m \to \{0,1\}^l$. If the two values are consistent, the verifier accepts the proof of identity with an overwhelmingly high probability, and it returns $b = 1$. Otherwise, the verifier rejects the proof with an overwhelmingly high probability and returns $b = 0$.

Definition 1.5.11 (Identification scheme, IS). *An identification scheme, Π, is a public key protocol comprising of the following algorithms running in time* $\mathsf{poly}(n)$*:*

- $\mathrm{KGen} : \mathbb{N} \to \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}$ *- a probabilistic algorithm that takes as an input $1^n \in \mathbb{N}$ and outputs a pair of matching secret key*

and public key, $(\mathsf{sk}, \mathsf{pk}) \in \{0,1\}^n \times \{0,1\}^{\mathsf{poly}(n)}, n \in \mathbb{N}$, obtained as

$$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{KGen}(1^n),$$

where 1^n denotes the security parameter in unary representation;

- $P_1 : \{0,1\}^n \to \{0,1\}^l \times \mathcal{S}, l \in \mathbb{N}$ - a probabilistic algorithm that takes as an input the secret key, sk, and outputs the initial state $\mathrm{init} \in \mathcal{S}$ and a message $\mathbf{m} \in \{0,1\}^l$, obtained as

$$(\mathbf{m}, \mathrm{init}) \leftarrow P_1(\mathsf{sk}),$$

- $P_2 : \{0,1\}^n \times \mathcal{S} \times \mathcal{C} \to \{0,1\}^m, m \in \mathbb{N}$ - a probabilistic polynomial time algorithm that takes as an input the secret key, sk, the initial state init, and the challenge, c, and then outputs the response $\mathbf{r} \in \{0,1\}^m, m \in \mathbb{N}$, obtained as

$$\mathbf{r} \leftarrow P_2(\mathsf{sk}, \mathrm{init}, c),$$

- $\mathrm{Ver} : \{0,1\}^{\mathsf{poly}(n)} \times \mathcal{C} \times \{0,1\}^m \to \{0,1\}^l$ - a deterministic algorithm that takes as an input the public key, pk, the challenge, c, the response $\mathbf{r}$, and verifies the consistency between the original message and the response. Namely, it outputs

$$b \leftarrow \mathrm{Ver}(\mathsf{pk}, c, \mathbf{r}) = \mathbf{m}.$$

In Chapter 4, we will give an example of a Σ-protocol which we will then fully analyze. We observe here that even from the high-level description of the scheme, it is not hard to see that identification schemes are a direct application of the so-called *interactive proofs*. The basic properties of interactive proofs, known as the *completness* and *soundness*, thus translate into basic properties of identification schemes. Namely, we require that the scheme is *complete*, meaning that the honest prover is able to convince the verifier of its identity with an overwhelmingly high probability. More formally, we state the following.

Definition 1.5.12 (Completness). *We say that an identification scheme,* $\Pi = (\mathrm{KGen}, P_1, P_2, \mathrm{Ver})$, *is complete if there exists a negligible function,* $\mathsf{negl}(n)$, *for which the following is satisfied :*

$$\Pr[\mathrm{Ver}(\mathsf{pk}, c, P_2(\mathsf{sk}, \mathsf{init}, c)) = 1] \geq 1 - \mathsf{negl}(n),$$

where init *is generated as the output of* $P_1(\mathsf{sk})$.

We also require that the scheme is *sound*, meaning that a dishonest prover cannot convince the verifier into accepting its proof with overwhelmingly high probability. This property captures the requirement that a cheating polynomial-time adversary, which does not hold the secret key, cannot convince the verifier to accept its proof of identity. More formally, the soundness is given via the following definition.

Definition 1.5.13 (Soundness). *Let* $\Pi = (\mathrm{KGen}, P_1, P_2, \mathrm{Ver})$ *be an identification scheme, and* $n \in \mathbb{N}$ *be a security parameter. Let then* $A_1 : \{0,1\}^n \to \{0,1\}^l \times \mathcal{S}, l \in \mathbb{N}$ *and* $A_2 : \{0,1\}^n \times \mathcal{S} \times \mathcal{C} \to \{0,1\}^m, m \in \mathbb{N}$, *be polynomial time algorithms run by an adversary* A. *For the given* n, Π, A, *the soundness experiment* $\mathrm{Soundness}(n, \Pi, A)$ *consists of the following steps:*

Experiment $\mathrm{Soundness}(n, \Pi, A)$
$1:$ $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{KGen}(1^n)$
$2:$ $(\mathbf{m}, \mathsf{init}) \leftarrow A_1(\mathsf{pk}),$
$3:$ $c \xleftarrow{\$} \mathcal{C},$
$4:$ $\mathbf{r} \leftarrow A_2(\mathsf{pk}, \mathsf{init}, c),$
$5:$ return $\mathrm{Ver}(\mathsf{pk}, c, \mathbf{r}) = \mathbf{m}$

We say that an identification scheme has soundness s, *where* s *is a constant in* $[0, 1]$, *if for any polynomial-time adversary,* A, *the probability that* $\mathrm{Soundness}(n, \Pi, A)$ *outputs true is at most* s.

Finally, the common requirement from an identification scheme is that it is *zero-knowledge*. This means that if the prover is honest, i.e. it knows

the secret key, the verifier does not learn anything from their communication apart from the fact that the prover knows the secret key.[8] A common way to formalize this notion is to say that there exists a polynomial time algorithm, called a *simulator*, that has access only to the public key and that can reproduce the *transcript* generated between the prover and the verifier, i.e. messages exchanged during their interaction. If there exists a simulator that can do that, the verifier has as much information from interacting with the prover as by running the simulator. This further implies that the verifier does not obtain any additional information from its interaction with the prover, and we say the protocol is zero-knowledge.

In this thesis, we are only interested in the case where the verifier *honestly* interacts with the prover, which corresponds to the honest-verifier zero-knowledge property defined as follows.

Definition 1.5.14. *Let* $\Pi = (\mathrm{KGen}, P_1, P_2, \mathrm{Ver})$ *be an identification scheme, and* $n \in \mathbb{N}$ *be a security parameter. Let then* $\mathrm{Sim} : \{0,1\}^n \to \{0,1\}^l \times \mathcal{C} \times \{0,1\}^m$ *be a polynomial time algorithm (the simulator) and* $A : \{0,1\}^n \to \{0,1\}^l \times \mathcal{C} \times \{0,1\}^m \to \{0,1\}$ *be a distinguishing algorithm run by an adversary,* A.

For the given n, Π, Sim, A, *the honest verifier zero-knowledge experiment* $\mathrm{HVZK}(n, \Pi, \mathrm{Sim}, A)$ *consists of the following steps:*

---

[8]If the prover is dishonest, i.e. it does not know the secret key, the verifier cannot learn the secret key from the prover.

Experiment $\mathrm{HVZK}(n, \Pi, \mathrm{Sim}, A)$

1 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{KGen}(1^n)$

2 : $b \xleftarrow{\$} \{0, 1\}$

3 : if $(b = 0)$:

4 : $\quad (\mathbf{m}, \mathrm{init}) \leftarrow P_1(\mathsf{sk})$,

5 : $\quad c \xleftarrow{\$} \mathcal{C}$,

6 : $\quad \mathbf{r} \leftarrow P_2(\mathsf{sk}, \mathrm{init}, c)$,

7 : $\quad \widetilde{b} \leftarrow A(\mathbf{m}, c, \mathbf{r})$

8 : if $(b = 1)$:

9 : $\quad (\tilde{\mathbf{m}}, \tilde{c}, \tilde{\mathbf{r}}) \leftarrow Sim(\mathsf{pk})$

10 : $\quad \widetilde{b} \leftarrow A(\tilde{\mathbf{m}}, \tilde{c}, \tilde{\mathbf{r}})$

11 : return $\widetilde{b} = b$

We say that an identification scheme is honest-verifier zero-knowledge if there exists a polynomial time simulator, Sim, *such that for any polynomial-time adversary, A, the probability that* $\mathrm{HVZK}(n, \Pi, Sim, A)$ *outputs true is at most* $\frac{1}{2} + \mathsf{negl}(n)$.

Fiat-Shamir transformation, FS   The *Fiat-Shamir transformation*, denoted by FS, can be broadly defined as removing interactions from an interactive scheme and thus constructing a non-interactive protocol provably secure. Fiat and Shamir introduced the transformation in [FS86] as a heuristic method used for transforming an interactive proof system into a digital signature. The heuristic is later proven by Pointcheval and Stern in [PS96], and since that time we refer to it as *transform* rather than heuristic.

This thesis uses FS to transform a Σ-protocol, known as the *Stern identification scheme*, into a digital signature scheme. We will give a more detailed explanation of this protocol, as well as of how it is transformed into a digital signature, in Chapter 4. For now, we give a formal definition of the Fiat-Shamir transformation that can be applied to an arbitrary three-round protocol.

Definition 1.5.15 (Fiat-Shamir transformation). *Let $n \in \mathbb{N}$ be a security parameter and $\Pi = (\mathrm{KGen}_{\mathrm{IS}}, P_1, P_2, \mathrm{Ver}_{\mathrm{IS}})$ be an identification scheme. We can construct a signature scheme, $\Sigma = (\mathrm{KGen}, \mathrm{Sig}, \mathrm{Ver})$, using the following procedure.*

Fiat-Shamir transformation,
FS

$$
\begin{aligned}
&1: \quad (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{KGen}_{\mathrm{IS}}(1^n) \\
&2: \quad \mathrm{KGen}(1^n) := (\mathsf{sk}, \mathsf{pk}) \\
&3: \quad (\mathbf{m}', \mathsf{init}) \leftarrow P_1(\mathsf{sk}) \\
&4: \quad c \leftarrow \mathcal{H}(\mathbf{m}, \mathbf{m}') \\
&5: \quad \mathbf{r} \leftarrow P_2(\mathsf{sk}, \mathsf{init}, c) \\
&6: \quad \mathrm{Sig}(\mathbf{m}, \mathsf{sk}) := (\mathbf{m}', c, \mathbf{r}) \\
&7: \quad \tilde{\mathbf{m}} \leftarrow \mathrm{Ver}_{\mathrm{IS}}(\mathsf{pk}, c, \mathbf{r}) \\
&8: \quad \mathrm{Ver} := (\mathcal{H}(\tilde{\mathbf{m}}, \mathbf{m}) = c),
\end{aligned}
$$

where we take $\mathcal{H} : \{0, 1\}^* \to \mathcal{C}$ as a random oracle.[a]

---

[a]In practice, we commonly assume that a hash function acts as a random oracle.

We observe here that the key generation part is the same in both protocols (line 1-2). The main idea underlying this transformation is conveyed by the signing algorithm in which the signer, acting as a prover in IS, runs the identification protocol by itself, as given by lines 3-6. The verifier then only needs to verify that the challenge was generated honestly, i.e. using the declared function $\mathcal{H}(\cdot)$ (line 8).

The above-described method is proven to be secure in the so-called *random-oracle model*. The proof can be found either in the original paper by Pointcheval and Stern or in some introductory literature such as [KL14], and the high-level idea of the proof can be explained as follows. From the FS description, we can see that the challenge c, calculated as $\mathcal{H}(\mathbf{m}', \mathbf{m})$, bounds signature to a specific message $\mathbf{m}$. Since $\mathcal{H}(\cdot)$ acts as a random oracle, which maps inputs into uniformly random challenges in $\mathcal{C}$, changing either $\mathbf{m}$ or $\mathbf{m}'$ would change the generated challenge, and consequently change the signature, with an overwhelmingly high probability. Therefore, for a malicious

signer, who does not hold sk, constructing a valid signature on message $\mathbf{m}$ is at least as hard as impersonating the prover in IS. More precisely, proofs of the Fiat-Shamir transform imply that from an identification scheme which has small soundness and is honest-verifier zero-knowledge, the resulting signature scheme is EUF-CMA secure.

Protocol optimization   Though already more efficient than the protocols obtained through the hash-and-sign paradigm, protocols obtained through the above-described approach can often be further optimized. One way to do so would be using *pseudo-random generators.* Informally speaking, pseudo-random generators can be described as efficient polynomial-time algorithms that transform short random vectors into longer, random-looking ones, also known as *pseudorandom vectors.* We commonly refer to the original vector as a *seed*, and to the algorithm in use as *pseudorandom generator.* To define pseudorandom generators more formally, we would first need to define more precisely what we mean by *pseudorandom.* We present both definitions in Definition 1.5.16.

Definition 1.5.16 (Pseudorandom generation). *Let $n \in \mathbb{N}$ be a security parameter and $l : \mathbb{N} \to \mathbb{N}$ be given as $l(n) = \mathsf{poly}(n)$. A deterministic polynomial time algorithm, G, is called a pseudorandom generator if, on an input* $\mathrm{seed} \in \{0,1\}^n$*, it outputs a vector of length $l(n)$ and satisfies the following:*

- ***Expansion criteria****: $\forall n \in \mathbb{N}$, $l(n) > n$,*

- ***Pseudorandomness criteria****: For any polynomial time algorithm D, there exists a negligible function, $\mathsf{negl}(n)$, such that:*

$$| \Pr[D(G(\mathrm{seed})) = 1] - \Pr[D(\mathbf{r}) = 1]| \leq \mathsf{negl}(n),$$

where the first probability is taken over the choice of seed*, sampled uniformly at random from the set $\{0,1\}^n$, and the randomness of D. The second probability is taken over the choice of* $\mathbf{r} \in \{0,1\}^{l(n)}$*, sampled uniformly at random from the set $\{0,1\}^{l(n)}$, and the randomness of D.*

In Chapter 4, where we explain our protocol design, we will give a more

detailed explanation of how one can benefit from the use of pseudorandom generators to increase the efficiency of a cryptographic scheme.

CHAPTER 2

GENERALIZED SYNDROME DECODING PROBLEM

The syndrome decoding problem is one of the fundamental problems in code-based cryptography, derived from the corresponding syndrome decoding method introduced in the previous chapter. In this chapter, we generalize this problem beyond the binary alphabet and the Hamming weight, which is the most common setting in both coding theory and cryptography. We do so by redefining the problem with respect to the elementwise weights and we then observe the problem for an arbitrary prime-number alphabet sizes and arbitrary elementwise weight function. We also show how to compute the

volumes of spheres for these weight functions, which is an important part of our analysis of Information Set Decoding algorithms, and which appears in [CDE21].

2.1 Combinatorial definitions

Here we introduce the combinatorial definitions necessary for deriving the combinatorial results presented throughout the chapter.

Definition 2.1.1 (Compositions). *Let $q, n \in \mathbb{N}$. The set of compositions of n into q parts, denoted as $\mathcal{R}$, is then given as:*

$$\mathcal{R} = \{\mathbf{r} = (r_i)_{i \in [q]} \in \mathbb{N}_0^q \mid \sum_{i \in [q]} r_i = n\}.$$

We have (see, for example, [Bog00]) that $|\mathcal{R}| = \binom{n+q-1}{q-1}$.

Definition 2.1.2 (Restricted compositions). *Let $q, n \in \mathbb{N}$ and $\mathrm{wt} : \mathbb{F}_q^n \to \mathbb{N}$ denote a weight function. The set of restricted compositions of n into q parts with weight restriction w, denoted as $\mathcal{R}_w$, is defined as:*

$$\mathcal{R}_w = \{\mathbf{r} \in \mathcal{R} : \sum_{i \in [q]} r_i wt(i) = w\}.$$

In our setting, we will associate a composition $\mathbf{r} \in \mathcal{R}$ with the vectors $\mathbf{x} \in \mathbb{F}_q$ st. the number of zeros in $\mathbf{x}$ is r_0, the number of ones is r_1 and so on.

Definition 2.1.3. *The set of vectors with composition $\mathbf{r}$ is defined as*

$$C_{\mathbf{r}} = \{\mathbf{x} \in \mathbb{F}_q^n : |\{j \in [n] : x_j = i\}| = r_i, \ \forall i \in [q]\},$$

Notice that for each $\mathbf{x} \in \mathbb{F}_q^n$, there is a unique $\mathbf{r} \in \mathcal{R}$ st. $\mathbf{x} \in C_{\mathbf{r}}$.

We thus obtain a simple formula for the size of C_r that is presented in Lemma 2.1.1.

Lemma 2.1.1.

$$|C_{\mathbf{r}}| = \binom{n}{\mathbf{r}} \equiv \binom{n}{r_0, \ldots, r_{q-1}} = \frac{n!}{r_0! r_1! \ldots r_{q-1}!}.$$

> where $\binom{n}{\mathbf{r}}$ is the multinomial coefficient.

Proof. Let us fix a composition $\mathbf{r}$ and count the number of vectors $\mathbf{x}$ with this composition. We first choose the r_0 different positions for the "0" coordinates from the n possible positions, then the r_1 different positions for the "1" coordinates from the $n - r_0$ remaining positions, and so on. This means we have

$$
|C_{\mathbf{r}}| = \binom{n}{r_0} \times \binom{n - r_0}{r_1} \times \cdots \times \binom{n - r_0 - \cdots - r_{q-2}}{r_{q-1}}
$$

$$
= \frac{n!}{(n - r_0)! r_0!} \times \frac{(n - r_0)!}{(n - r_0 - r_1)! r_1!} \times \cdots \times \frac{(n - r_0 - \cdots - r_{q-2})!}{0! r_{q-1}!}
$$

$$
= \frac{n!}{r_0! r_1! \ldots r_{q-1}!}
$$

where we used $\sum_i r_i = n$ and $0! = 1$. $\qquad\square$

2.2 Problem generalization

To define the generalized syndrome decoding problem, let us start by defining the notion of *elementwise weight function*, denoted by $\mathrm{wt}_M(\cdot)$.

Definition 2.2.1 (Elementwise weight function). *Let $n \in \mathbb{N}$, q be a prime number and $\mathrm{dist} : \mathbb{F}_q \to \mathbb{N}$ be a distance function (metric).*

We then define a weight function over an element, $\mathrm{wt}_m : \mathbb{F}_q \to \mathbb{N}$ as:

$$\forall x \in \mathbb{F}_q, \mathrm{wt}_m(x) = \mathrm{dist}(x, 0),$$

where 0 denotes the zero elements of the finite field. The elementwise weight function, $\mathrm{wt}_M : \mathbb{F}_q^n \to \mathbb{N}$, is then given as follows:

$$\forall \mathbf{x} \in \mathbb{F}_q^n, \mathbf{x} = (x_i)_{i \in [n]}, \mathrm{wt}_M(\mathbf{x}) = \sum_{i \in [n]} \mathrm{wt}_m(x_i).$$

Let us now take $n, k \in \mathbb{N}$, let q be a prime number, and $\mathrm{wt}_M(\cdot)$ be an elementwise weight function. In Problem 2.2.1 and Problem 2.2.2, we present the generalized version of the decisional and the search variant of the syndrome decoding problem, respectively.

Problem 2.2.1 (Decisional Syndrome Decoding Problem, D-SDP). *Given $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, and $w \in \mathbb{N}$, determine if there exists $\mathbf{e} \in \mathbb{F}_q^n$ satisfying $\mathbf{s} = \mathbf{e}\mathbf{H}^T$ and $\mathrm{wt}_M(\mathbf{e}) = w$.*

Problem 2.2.2 (Syndrome Decoding Problem, SDP). *Given $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, and $w \in \mathbb{N}$, find a vector $\mathbf{e} \in \mathbb{F}_q^n$ satisfying $\mathbf{s} = \mathbf{e}\mathbf{H}^T$ and $\mathrm{wt}_M(\mathbf{e}) = w$.*

We refer to the problem as *generalized* since our problem definition can be seen as more general than the original problem definition. Namely, in the original version, the alphabet is binary, i.e. $q = 2$, and the underlying weight function is the Hamming weight (introduced in the previous chapter). We

thus refer to the original version as the *binary syndrome decoding problem in the Hamming weight* and denote it by 2-SDP_H. As this problem has already been thoroughly analyzed, it will a baseline for our analysis.

2.2.1 Worst case complexity

In their paper from 1978, Berlekamp, McEliece, and Van Tilborg showed that the 2-SDP_H is **NP**-hard. They did so by reducing the *three-dimensional matching problem* which is known to be **NP**-hard, to the 2-SDP_H. Knowing that the problem is also in **NP**, the three authors actually proved that the 2-SDP_H is, in fact, **NP**-complete. Moreover, for the weight functions we are focusing on in this thesis, namely, the Hamming and the Lee weights, it has been proven that the q-vary variants of D-SDP are **NP**-complete, too. A proof of the **NP**-completeness for the q-ary SDP in the Hamming weight can be found in [Bar94], while proof of the **NP**-completeness of the q-ary SDP in the Lee weight is given in [Weg+20]. Nonetheless, the generalized case we introduced has not been analyzed from the complexity theory point of view, and we leave it as an open question for some future work.

2.2.2 Average-case complexity

In the average-case analysis, we are interested in the complexity of a problem whose inputs are sampled from a particular distribution. In this thesis, we are interested in the difficulty of solving the generalized syndrome decoding problem whose underlying code is sampled from a uniform distribution and it is guaranteed that there exists a solution to the problem.

Average-case syndrome decoding problem   Let us define the maximum value of the finite field $\mathbb{F}_q$ as:

$$\Delta := \max_{\mathbf{x}\in\mathbb{F}_q} \mathrm{wt}_M(\mathbf{x}),$$

and let the surface area of a sphere of radius w in the same vector space be denoted as $\mathcal{S}_M(q, n, w)$ and defined as follows:

$$\mathcal{S}_M(q, n, w) := \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathrm{wt}_M(\mathbf{x}) = w\}.$$

The average-case syndrome decoding problem is then given via the following definition.

Problem 2.2.3 (Average-case Syndrome Decoding Problem, A-SDP)**.** *Let $w \in \mathbb{N}$ satisfy $w \leq n\Delta$, and let $\tilde{\mathbf{e}} \in \mathbb{F}_q^n$ be sampled uniformly at random from $\mathcal{S}_M(q, n, w)$. Let then $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ be sampled uniformly at random from $\mathbb{F}_q^{(n-k) \times n}$, and calculate $\mathbf{s} \in \mathbb{F}_q^{n-k}$ as $\mathbf{s} = \tilde{\mathbf{e}}\mathbf{H}^T$. Given $\mathbf{H}$, $\mathbf{s}$ and w, find a vector $\mathbf{e} \in \mathbb{F}_q^n$ satisfying $\mathbf{s} = \mathbf{e}\mathbf{H}^T$ and $\mathrm{wt}_M(\mathbf{e}) = w$.*

Permuted kernel problem   The permuted kernel problem is a computational problem introduced in [Sha89], where the author showed that the decisional version of the problem is **NP**-complete. In this sub-section, we will show that the average-case version of the syndrome decoding problem, A-SDP, is polynomial-time reducible to the average-case version of the permuted kernel problem, A-PKP.[1] Let us choose a vector $\mathbf{v} \in \mathbb{F}_q^n$. The average-case version of PKP is given as follows.

Problem 2.2.4 (Average-case Permuted Kernel Problem, A-PKP)**.** *Let $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ be sampled uniformly at random from $\mathbb{F}_q^{(n-k) \times n}$, $\bar{\pi} : [n] \to [n]$ be sampled uniformly at random from a set of all permutations of n elements, and calculate $\mathbf{s} \in \mathbb{F}_q^{n-k}$ as $\mathbf{s} = \bar{\pi}(\mathbf{v})\mathbf{H}^T$. Given $\mathbf{H}, \mathbf{s}, \mathbf{v}$, find a permutation $\pi : [n] \to [n]$ that satisfies $\mathbf{s} = \pi(\mathbf{v})\mathbf{H}^T$.*

We remark here that in the original version of PKP, the syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$ is the all-zero vector in $\mathbb{F}_q^{n-k}$, i.e. $\mathbf{s} = \mathbf{0}$, hence the name of the problem. In the version of PKP we present, the problem is generalized by replacing the all-zero syndrome with an arbitrary vector from $\mathbb{F}_q^{n-k}$. This generalization is introduced to highlight the similarity between A-SDP and A-PKP and does not affect the complexity of the PKP problem. We now present a reduction from SDP to PKP.

---

[1]This result will turn out to be rather important to our protocol design presented in Chapter 4.

Average-case reduction

Let $\mathbf{H} \in \mathbb{F}^{(n-k) \times n}, \mathbf{s} \in \mathbb{F}_q^{n-k}$, and $w \in \mathbb{N}$. Let then $\mathcal{R}_w$ be the set of restricted compositions, defined in Definition 2.1.2, where the restriction is given as $\sum_{i \in [q]} r_i \operatorname{wt}_m(i) = w$, and where $\operatorname{wt}_m : \mathbb{F}_q \to \mathbb{N}$ is a weight function. Finally, let $\tilde{\mathbf{r}} := (\tilde{r}_i)_{i \in [q]} \in \mathcal{R}_w$ such that $\binom{n}{\tilde{\mathbf{r}}} = \max_{\mathbf{r} \in \mathcal{R}_E} \binom{n}{\mathbf{r}}$ and $\tilde{\mathbf{v}} \in \mathbb{F}_q^n$ consists of $\tilde{r}_0$ zeros, $\tilde{r}_1$ ones, etc.

> **Proposition 2.2.1** ($SDP \preceq PKP$). *If there exists an algorithm B that solves an instance of* PKP *on the input* $(\mathbf{H}, \mathbf{s}, \tilde{\mathbf{v}})$ *in time* $O(poly(n \log_2 q))$, *then there exists an algorithm A that solves the instance of SDP on the input* $(\mathbf{H}, \mathbf{s}, w)$ *in time* $O(poly(n \log_2 q)$ *and succeeds with probability* $\Omega(1/poly(n \log_2 q))$ *when q is constant.*

Proof. For a fixed $q, n, k, w \in \mathbb{N}$ and a fixed elementwise weight function, $\operatorname{wt}_M(\cdot)$, consider a random instance of SDP. Namely, we sample $\mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k) \times n}$, $\bar{\mathbf{e}} \xleftarrow{\$} \mathbb{F}_q^n$ st. $\operatorname{wt}_M(\bar{\mathbf{e}}) = w$, and we calculated $\mathbf{s} = \mathbf{H}\bar{\mathbf{e}}$. Given $(\mathbf{H}, \mathbf{s})$, our goal is to find $\mathbf{e}$ st. $\operatorname{wt}_M(\mathbf{e}) = w$ and $\mathbf{e}\mathbf{H}^T = \mathbf{s}$.

Let us choose $\mathbf{r} \in R_w$. We then take a vector $\mathbf{v} \in C_\mathbf{r}$, and let A be an algorithm that runs in $\operatorname{poly}(n)$ and finds a solution to a random instance of PKP given on the input $(\mathbf{H}, \mathbf{s}, \mathbf{v})$. A solution to this problem is a vector $\mathbf{e} = \pi(\mathbf{v})$ st. $\mathbf{e}\mathbf{H}^T = \mathbf{s}$. Since $\operatorname{wt}_M(\mathbf{e}) = \operatorname{wt}_M(\mathbf{v}) = w$, we found a solution to our original syndrome decoding problem. Nevertheless, since $\mathbf{r}$ can be any element in R_w, we do not have a guarantee that there is a solution to the given PKP instance in the first place. To guarantee the solution is found, we repeat the algorithm for each $\mathbf{r} \in R_w$, as we know that if $\bar{\mathbf{r}} \in R_w$ is chosen st. $\bar{\mathbf{e}} \in C_{\bar{\mathbf{r}}}$, the a solution exists. We thus repeat the algorithm A $|R_w| \leq |R| = \binom{n+q-1}{q-1} = poly(n)$ times in the worst case. It implies that the time for finding a solution to an SDP instance, given access to an oracle solving a PKP instance, is at most polynomial and concludes the proof. $\square$

We remark here that Proposition 2.2.1 implies that, as long as q is constant, the PKP is essentially at least as hard as the SDP for elementwise weight functions.

2.3 Sphere surface areas and ball volumes

In this subsection, we derive a general formula for the sphere surface area and the ball volume, as well as the formula for the asymptotic values of these, that holds for an arbitrary choice of an elementwise weight function. This is considered to be one of the major contributions of this thesis as it allows us to analyze the syndrome decoding problem in a lot broader setting, both from the complexity theory and cryptanalytic perspective.

We start this section by recalling the well-known results on the surface areas (ball volumes) in the Hamming weight spaces. We then derive the corresponding formulas on the sphere surface area and the ball volume for the Lee weight, relying on the approach presented in [Ast84]. We finish the section by generalizing the approach for Lee weight to an arbitrary weight elementwise weight function and then re-derive the well-known result on the Hamming weight using our approach.

2.3.1 Hamming weight

The Hamming weight, depicted in Figure 2.1a as a unit-distance graph[2], is the most common choice of weight function, both in coding theory and in cryptography. As such, it is well studied from both perspectives so we will take it as a baseline for our research. Let us first recall its definition.

For a prime number q and an integer n, the Hamming weight of an element in $\mathbb{F}_q$, denoted as $\mathrm{wt}_h : \mathbb{F}_q \to \mathbb{N}$, and its elementwise version over $\mathbb{F}_q^n$, $\mathrm{wt}_H : \mathbb{F}_q^n \to \mathbb{N}$, are defined as:

$$\forall \mathbf{x} \in \mathbb{F}_q^n, \ \mathbf{x} = (x_i)_{i \in [n]}, \mathrm{wt}_h(x_i) := \begin{cases} 0, & \text{if } x_i = 0 \\ 1, & \text{otherwise} \end{cases}, \mathrm{wt}_H(\mathbf{x}) = \sum_{i \in [n]} \mathrm{wt}_h(x_i).$$

---

[2]In a unit distance graph, each vertex corresponds to a vector in $\mathbb{F}_q^n$ and each edge connects two vertices at the distance 1.
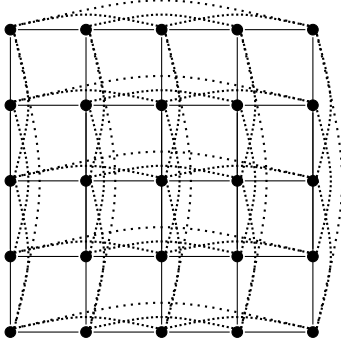
(a) Two-dimensional Hamming weight space for 5-ary alphabet
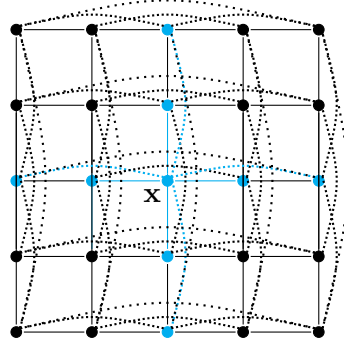
(b) Hamming ball (in cyan) of radius 1, centered at $\mathbf{x}$

Sphere surface area and ball volume

Let us recall that a sphere of radius $w \in \mathbb{N}, w \leq n$, in the vector space $\mathbb{F}_q^n$ endowed with the Hamming metric, denoted as $\mathcal{S}_H(q, n, w)$, is defined as:

$$\mathcal{S}_H(q, n, w) := \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathrm{wt}_H(\mathbf{x}) = w\}.$$

The corresponding sphere surface area, $\mathrm{surf}_H(\cdot)$, then counts the number of vectors having w non-zeros coordinates, and it is then given as:

$$\mathrm{surf}_H(q, n, w) := |\mathcal{S}_H(q, n, w)| = \binom{n}{w}(q - 1)^w.$$

Similarly, a Hamming ball of radius $w \in \mathbb{N}$, $w \leq n$, in the vector space $\mathbb{F}_q^n$ endowed with the Hamming metric, denoted as $\mathcal{B}_H(q, n, w)$ and depicted in Figure 2.1b, is defined as:

$$\mathcal{B}_H(q, n, w) := \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathrm{wt}_H(\mathbf{x}) \leq w\}.$$

The corresponding ball volume, $\mathrm{vol}_H(q, n, w)(\cdot))$, then counts the number of vectors having at most w non-zeros coordinates. Given that we observe a vector space over a discrete set, we have that:

$$\mathrm{vol}_H(q, n, w) := |\mathcal{B}_H(q, n, w)| = \sum_{i=0}^{w} \binom{n}{i}(q - 1)^i,$$

Asymptotic sphere surface area and ball volume

It is not hard to prove that for $w = \omega n$, $\omega \in [0, 1]$, the expression $\binom{n}{w}(q - 1)^w$ is upper-bounded by $q^{nH_q(\omega)}$ and lower bounded by $q^{nH_q(\omega)-o(n)}$ (see, for example, [Ber68]), where $H_q(\omega)$ is the *q-ary entropy function* given as:

$$H_q(\omega) = -\omega \log_q(\frac{\omega}{q-1}) - (1 - \omega) \log_q(1 - \omega).$$

As the bounds are rather tight, we obtain a close approximation of the asymptotic surface area of a Hamming sphere. A sphere of radius $w = \omega n$, $\omega \in [0, 1]$, in the vector space $\mathbb{F}_q^n$, thus has the following asymptotic surface area, $s_H(q, \omega)$:

$$s_H(q, \omega) = \lim_{n \to \infty} \frac{1}{n} \log_q \left(\operatorname{surf}_H(q, n, \omega n) \right) = H_q(\omega).$$

As the entropy function is concave, from its derivative, we determine that it reaches its maximum for $\omega = 1 - 1/q$. This further implies that the asymptotic surface area of a sphere also reaches its maximum for $1 - 1/q$, and decreases afterward. For the asymptotic value of the ball volume, defined as:

$$v_H(q, \omega) = \lim_{n \to \infty} \frac{1}{n} \log_q \operatorname{vol}_H(q, n, \omega n).$$

we then obtain:

$$
\begin{aligned}
v_H(q, \omega) &= \lim_{n \to \infty} \frac{1}{n} \log_q \sum_{i=0}^{w} \operatorname{surf}_H(q, n, i) \\
&= \max_{i=\{0,1,\dots,w\}} \lim_{n \to \infty} \frac{1}{n} \log_q \operatorname{surf}_H(q, n, i) \\
&= \begin{cases} s_H(q, \omega) = H_q(\omega), & \text{if } \omega \leq 1 - 1/q \\ 1, & \text{otherwise} \end{cases}.
\end{aligned}
$$

We highlight here that the two values, namely, the asymptotic sphere surface area and the asymptotic volume are identical for $\omega \leq 1 - 1/q$ and different for the rest of the interval. This difference is rather important for the cryptanalysis presented in the following chapter.

2.3.2 Lee weight

The Lee weight, depicted in Figure 2.2a by a unit-distance graph, is another weight function that is commonly encountered in coding theory and, since recently, more often used in cryptography. It was introduced in [Lee58] exactly for the purpose of decoding non-binary linear codes. Let us recall its definition.

For a prime number q and an integer n, the Lee weight over an element in $\mathbb{F}_q$, denoted as $\mathrm{wt}_l : \mathbb{F}_q \to \mathbb{N}$, and its elementwise version $\mathrm{wt}_L : \mathbb{F}_q^n \to \mathbb{N}$, are defined as follows:

$$\forall \mathbf{x} \in \mathbb{F}_q^n, \mathbf{x} = (x_i)_{i \in [n]}, \mathrm{wt}_l(x_i) := min(x_i, q - x_i), \mathrm{wt}_L(\mathbf{x}) := \sum_{i \in [n]} \mathrm{wt}_l(x_i).$$
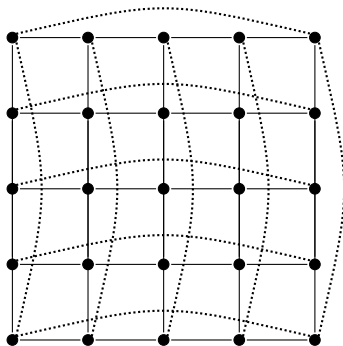
(a) Two-dimensional Lee weight space for 5-ary alphabet
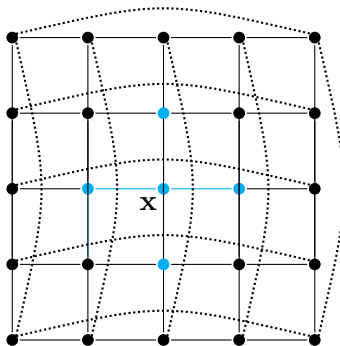
(b) Lee ball (in cyan) of radius 1, centered at $\mathbf{x}$

Sphere surface area and ball volume

We recall that a sphere of radius $w \in \mathbb{N}$, in the vector space $\mathbb{F}_q^n$ endowed with the Lee metric, denoted as $\mathcal{S}_L(q, n, w)$, is defined as:

$$\mathcal{S}_H(q, n, w) := \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathrm{wt}_L(\mathbf{x}) = w\},$$

while a ball of radius w in the same vector space, denoted as $\mathcal{B}_H(q, n, w)$ and depicted in Figure 2.2b, is defined as:

$$\mathcal{B}_H(q, n, w) := \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathrm{wt}_L(\mathbf{x}) \le w\}.$$

In contrast to the surface area of a Hamming sphere, the surface area of a Lee sphere cannot be presented via simple combinatorial formula. A more complex combinatorial expression can still be derived, as presented in [HW21], so it can be one approach for calculating the surface area of a Lee sphere. Another approach would be to combine the use of so-called generating functions and saddle-point techniques to calculate the volume, as presented in [GS91]. Equivalently, one can use multinomial coefficients and Lagrange multipliers as presented in [Ast84]. In this thesis, we rely on the last approach and use it to derive a more generalized method to calculate the sphere surface area (and, consequently the volume of a ball) that holds for an arbitrary elementwise weight function.

Proposition 2.3.1 (Surface area of a Lee sphere). *Let q be a prime number, $n, w \in \mathbb{N}$, where $w \leq \lfloor q/2 \rfloor n$. The surface area of a sphere of radius w in the vector space $\mathbb{F}_q^n$ endowed with the Lee weight, $\mathrm{surf}_L(q, n, w)$, is then calculated as:*

$$\mathrm{surf}_L(q, n, w) = \sum_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}}.$$

Proof. We just write

$$\mathrm{surf}_q(n, w) = |\{\mathbf{x} \in \mathbb{F}_q : \mathrm{wt}_L(\mathbf{x}) = w\}| = \sum_{\mathbf{r} \in R_w} |C_{\mathbf{r}}| = \sum_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}}.$$

$\square$

Given Proposition 2.3.1, we can calculate the volume of a Lee ball of radius $w \in \mathbb{N}, w \leq \lfloor q/2 \rfloor n$, in the vector space $\mathbb{F}_q^n$ as:

$$\mathrm{vol}_L(q, n, w) := |\mathcal{B}_L(q, n, w)| = \sum_{j=0}^{w} \sum_{\mathbf{r} \in \mathcal{R}_j} \binom{n}{\mathbf{r}}.$$

Asymptotic sphere surface area

To calculate the asymptotic surface area of a Lee sphere, we generalize the method proposed in [Ast84] and formalize its proof in Proposition 2.3.2.

Proposition 2.3.2 (Asymptotic surface area of a Lee sphere). *Let q be a prime number, and $n, w \in \mathbb{N}$, where $w = \lfloor \frac{q}{2} \omega n \rfloor$, $\omega \in [0, 1]$. The asymptotic surface area of a sphere of radius d in the vector space $\mathbb{F}_q^n$ endowed with the Lee weight, $s_L(q, \omega)$, is then calculated as:*

$$s_L(q, \omega) := \lim_{n \to +\infty} \max_{\mathbf{r} \in \mathcal{R}_w} \left(- \sum_{i=1}^{q} \frac{r_i}{n} \log_q \frac{r_i}{n} \right). \qquad (2.1)$$

Proof. Let us recall that, by Proposition 2.3.1, $\text{surf}_L(q, n, w) = \sum_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}}$. First notice that

$$\text{surf}_L(q, n, w) \leq |\mathcal{R}_w| \max_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}} \leq |\mathcal{R}| \max_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}} = \binom{n + q - 1}{q - 1} \max_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}}.$$

which means we can write

$$\max_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}} \leq \text{surf}_L(q, n, w) \leq \binom{n + q - 1}{q - 1} \max_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}}.$$

Following the same line of reasoning as in [Ast84], we assume that $\max_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}}$ is reached for $\tilde{\mathbf{r}} \in \mathcal{R}_w$ and observe that $\binom{n+q-1}{q-1} \leq \frac{(n+q-1)^{q-1}}{(q-1)!} \leq (n+q+1)^{q-1}$. We then calculate the asymptotic upper and lower bound given as:

$$\lim_{n \to +\infty} \frac{1}{n} \log_q \binom{n}{\tilde{\mathbf{r}}} \leq s_L(q, \omega) \leq \lim_{n \to +\infty} \frac{1}{n} \log_q \binom{n}{\tilde{\mathbf{r}}} + \lim_{n \to +\infty} \frac{q - 1}{n} \log_q(n+q-1).$$

As the last term approaches 0^+ as $n \to +\infty$, the asymptotic value of the sphere surface is $s_L(q, \omega) = \lim_{n \to +\infty} \frac{1}{n} \log_q \binom{n}{\tilde{\mathbf{r}}}$. Using Stirling's approximation, which states that for a large integer value, m, $\log m! = m \log m -$

$m \log_q(e) + o(m)$, we obtain the following:

$$
\frac{1}{n} \log_q \binom{n}{\tilde{\mathbf{r}}} = \frac{1}{n} \log_q \frac{n!}{\tilde{r}_0! \tilde{r}_1! \dots \tilde{r}_{q-1}!} = \frac{1}{n} \log_q n! - \frac{1}{n} \sum_{i \in [q]} \log_q \tilde{r}_i!
$$

$$
= \frac{1}{n} (n \log_q n - n \log_q(e)) - \frac{1}{n} \sum_{i \in [q]} \left(\tilde{r}_i \log_q \tilde{r}_i - \tilde{r}_i \log_q(e) \right) + o(1)
$$

$$
= \log_q n - \log_q(e) - \sum_{i \in [q]} \left(\frac{\tilde{r}_i}{n} \log_q \frac{\tilde{r}_i}{n} n - \frac{r_i}{n} \log_q(e) \right) + o(1)
$$

$$
= \log_q n - \log_q(e) - \sum_{i \in [q]} \frac{\tilde{r}_i}{n} \log_q \frac{\tilde{r}_i}{n} - (\log_q n - \log_q e) \sum_{i \in [q]} \frac{\tilde{r}_i}{n} + o(1)
$$

$$
= - \sum_{i \in [q]} \frac{\tilde{r}_i}{n} \log_q \frac{\tilde{r}_i}{n} + o(1).
$$

We thus obtain the expression that proves the claim, namely:

$$
s_L(q, \omega) = \lim_{n \to +\infty} \frac{1}{n} \log_q \binom{n}{\tilde{\mathbf{r}}} = \lim_{n \to +\infty} \left(- \sum_{i \in [q]} \frac{\tilde{r}_i}{n} \log_q \frac{\tilde{r}_i}{n} \right)
$$

$$
= \lim_{n \to +\infty} \max_{\mathbf{r} \in \mathcal{R}_w} \left(- \sum_{i \in [q]} \frac{r_i}{n} \log_q \frac{r_i}{n} \right).
$$

$\square$

We note here that the expression 2.1 is not a closed-form solution, but a problem of maximizing a concave function[3], or, equivalently, a problem of minimizing a convex function. The solution to this problem then yields the asymptotic surface area of a Lee sphere and, consequently, the asymptotic volume of a Lee ball. Calculating these values thus comes down to solving the following constrained convex optimization problem.

Problem 2.3.1. *Let* $\mathbf{p} \in \mathbb{R}_+^q$, $\mathbf{p} = (p_i)_{i \in [q]}$.

- *Maximize:* $- \sum_{i \in [q]} p_i \log_q p_i$,

---

[3]As each summand in the expression (3.1) is a concave function, their sum is also a concave function.

> - *Subject to:* $\sum_{i\in[q]} p_i = 1$, $\sum_{i\in[q]} p_i \operatorname{wt}_l(i) = \delta\lfloor q/2\rfloor$.

It can be easily verified that when replacing the optimization variable p_i from Problem 2.3.1 with r_i/n from (3.1), we recover the original formulation of the asymptotic surface area of a Lee sphere from Proposition 2.3.2. The solution of Problem 2.3.1 then indeed yields the asymptotic surface area of a Lee sphere. More specifically, if we denote by $\tilde{\mathbf{p}} = (\tilde{p}_1, ..., \tilde{p}_q)$ the solution[4] of Problem 2.3.1, the asymptotic value of the surface area of a sphere of radius $w = \lfloor \frac{q}{2}\omega n\rfloor$, $\omega \in [0, 1]$, is calculated as $s_L(w, \omega) = -\sum_{i\in[q]} \tilde{p}_i \log_q \tilde{p}_i$.

Solving convex optimization problem   To solve Problem 2.3.1, we can rely on the method of Lagrange multipliers, as suggested in [Ast84]. We thus first form the Lagrangian function, $\mathcal{L}(\mathbf{p}, \lambda_1, \lambda_2)$, corresponding to the constrained optimization problem:

$$\mathcal{L}(\mathbf{p}, \lambda_1, \lambda_2) = -\sum_{i\in[q]} p_i \log_q p_i - \lambda_1\Big(\sum_{i\in[q]} p_i - 1\Big) - \lambda_2\Big(\sum_{i\in[q]} p_i \operatorname{wt}_l(i) - \delta\lfloor q/2\rfloor\Big).$$

We then ask for the stationary point[5] of $\mathcal{L}(\mathbf{p}, \lambda_1, \lambda_2)$. Namely, we ask for values of $\mathbf{p}, \lambda_1, \lambda_2$ for which the gradient of $\mathcal{L}$, denoted as $\nabla\mathcal{L}$, evaluates to zero. Therefore, we obtain a system of equations containing all partial derivatives of $\mathcal{L}$ evaluating to zero. The solution of this system yields the solution to our constrained convex optimization problem. In particular, the

---

[4]As $\sum_{i\in[q]} p_i \log_q p_i$ is a convex function, the solution is unique.
[5]Again, given that the function is concave and we ask for its maximum, it is expected that the solution will be unique.

system corresponding to Problem 2.3.1 is given as follows:

$$\frac{\partial \mathcal{L}}{\partial p_0} = (-\log_q p_0 - 1/\ln(q)) - \lambda_1 = 0$$

$$\frac{\partial \mathcal{L}}{\partial p_1} = (-\log_q p_1 - 1/\ln(q)) - \lambda_1 - \lambda_2 \operatorname{wt}_l(1) = 0$$

$$\cdots$$

$$\frac{\partial \mathcal{L}}{\partial p_{q-1}} = (-\log_q p_{q-1} - 1/\ln(q)) - \lambda_1 - \lambda_2 \operatorname{wt}_l(q-1) = 0$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_1} = -\sum_{i \in [q]} p_i + 1 = 0$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_2} = -\sum_{i \in [q]} p_i \operatorname{wt}_l(i) + \delta \lfloor q/2 \rfloor = 0$$

Let us observe here that the number of variables in the system grows linearly with q. It is then expected that for a large enough q, solving the system manually would be a long and error-prone process. We thus normally rely on non-linear system solvers to find the solution for us. In this work, we use solvers from the software package called MOSEK [ApS19]. Though the underlying methodology of these solvers is equivalent to the method based on Lagrange multipliers, the input to the solvers is not in the form of a system of equations. It rather resembles the problem's original formulation, with minor adjustments to the framework. In our case, the problem is reformulated as follows.

Problem 2.3.2. *Let* $\mathbf{p} \in \mathbb{R}_+^q$, $\mathbf{p} = (p_i)_{i \in [q]}$, *and* $\mathbf{t} \in \mathbb{R}_+^q$, $\mathbf{t} = (t_i)_{i \in [q]}$.

- *Maximize:* $\sum_{i \in [q]} t_i$,

- *Subject to:* $\sum_{i \in [q]} p_i = 1$, $\sum_{i \in [q]} p_i \operatorname{wt}_l(i) = \delta \lfloor q/2 \rfloor$, $(1, \mathbf{p}, \mathbf{t}) \in K_{exp}$.

We note here that the constraint $(1, \mathbf{p}, \mathbf{t}) \in K_{exp}$ means that $t_i \leq -p_i \log_q p_i$, for each $i \in [q]$.[6] It can be easily verified that Problem 2.3.1 and Problem 2.3.2

---

[6]The notation K_{exp} comes from the MOSEK optimizer[ApS19] and represents the exponential convex cone.

are equivalent, hence, finding a solution to either of the two yields the asymptotic value of the sphere surface area. Our numerical results on the asymptotic values of the surface areas of Lee spheres (equivalently, the asymptotic values of the volumes of Lee balls) were obtained using this approach.

Asymptotic ball volume

As in the case of Hamming metric, we see that $s_L(\cdot)$ is a concave function, so for a fixed q, its maximum is reached for some $\omega_{\max} \in [0,1]$. To find $\omega_{\max}$, we can first solve Problem 2.3.1 to express $s_L(\cdot)$ as a function of $\omega \in [0,1]$, and then calculated the value of $\omega_{\max}$ from the derivative of $s_L(q,\omega)$, as we did in the case of Hamming metric. Alternatively, we can first solve Problem 2.3.1 without the last constraint on the weight[7], namely, without requiring that $\sum_{i \in [q]} p_i \, \mathrm{wt}_l(i) = \omega \lfloor q/2 \rfloor$, and then calculate for which $\omega_{\max} \in [0,1]$ the second constraint is also satisfied. We thus basically reverse the ordering of the maximizations. Therefore, using the method of Lagrange multipliers, we can first find $\{p_i\}_{i \in [q]}$ for which the maximum of $-\sum_{i \in [q]} p_i \log_q p_i$ is reached and $\sum_{i \in [q]} p_i = 1$ is satisfied. It turns out that if $p_i = 1/q, \forall i \in [q]$, the maximum is reached, so we calculate $\omega_{\max}$ as:

$$
\omega_{\max} = \sum_{i \in [q]} \frac{p_i \, \mathrm{wt}_l(i)}{\lfloor q/2 \rfloor} = \frac{1}{q \lfloor q/2 \rfloor} \sum_{i \in [q]} wt_l(i) = \begin{cases} \frac{1}{2}, & \text{if } q \text{ is even} \\ \frac{q+1}{2q}, & \text{else} \end{cases} . \quad (2.2)
$$

Given the solution to Problem 2.3.1, we can then calculate the asymptotic volume of a Lee ball of radius $w = \lfloor \frac{q}{2} \omega n \rfloor$, $\omega \in [0,1]$, in the vector space $\mathbb{F}_q^n$ as:

$$
v_L(q,\omega) = \lim_{n \to \infty} \frac{1}{n} \log_q \mathrm{vol}_L(q,n,\omega n) = \lim_{n \to \infty} \frac{1}{n} \log_q \sum_{i=0}^{w} \mathrm{surf}_L(q,n,i)
$$

$$
\approx \max_{i=\{0,1,\ldots,w\}} \lim_{n \to \infty} \frac{1}{n} \log_q \mathrm{surf}_q(n,i) = \begin{cases} s_L(q,\omega), & \text{if } \omega \le \omega_{max} \\ 1, & \text{else} \end{cases}
$$

$$
= \begin{cases} \lim_{n \to +\infty} \max_{\mathbf{r} \in \mathcal{R}_w} \left(-\sum_{i \in [q]} \frac{r_i}{n} \log_q \frac{r_i}{n} \right), & \text{if } \omega \le \omega_{max} \\ 1, & \text{else} \end{cases} ,
$$

---

[7]Notice here that, in fact, the maximization is independent of the weight function.

where $\omega_{\max}$ is given in (3.2), and the maximum of 1 is reached when the ball covers the whole vector space $\mathbb{F}_q^n$, i.e. $\mathrm{vol}_q(n,d) = q^n$.

2.3.3 Arbitrary elementwise weight

Proposition 2.3.1 and Proposition 2.3.2, in fact, can be generalized to any elementwise weight function. Intuitively, we can see this generalization as replacing the Lee weight constraint in the definition of the set of restricted compositions with a weight constraint corresponding to some other weight function, for example, the Hamming weight. We thus obtain the following proposition.

Sphere surface area   Let q be a prime number, $n \in \mathbb{N}$, and $\mathbb{F}_q^n$ be a vector space endowed with an elementwise weight function, $\mathrm{wt}_M(\cdot)$. Let $\mathrm{wt}_m(\cdot)$ be the corresponding weight over an element in $\mathbb{F}_q$, and $\Delta \in \mathbb{F}_q$ be defined as $\Delta := \max_{x \in \mathbb{F}_q}\{\mathrm{wt}_m(x)\}$. Let $\omega \in [0,1]$ and $d = \lfloor \Delta \omega n \rfloor$.

Proposition 2.3.3 ((Asymptotic) Sphere surface area). *The surface area of a sphere of radius d in the vector space $\mathbb{F}_q^n$, $\mathrm{surf}_M(q,n,d)$, and its asymptotic value, $s_M(q,\omega)$, are calculated as:*

$$\mathrm{surf}_M(q,n,d) = \sum_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}}, \quad s_M(q,\delta) := \lim_{n \to +\infty} \max_{\mathbf{r} \in \mathcal{R}_w} \left(-\sum_{i=1}^{q} \frac{r_i}{n} \log_q \frac{r_i}{n} \right).$$

To prove this proposition we can follow the same line of reasoning as in the proof of Proposition 2.3.1 and Proposition 2.3.2, so we will not repeat it here.

From Proposition 2.3.3, we have that the asymptotic surface area of a sphere in a vector space endowed with an arbitrary elementwise weight function comes down to solving the following convex optimization problem.

Problem 2.3.3. *Let q be a prime number, let $\mathbf{p} \in \mathbb{R}_+^q$, $\mathbf{p} = (p_i)_{i \in [q]}$, and let $\mathrm{wt}_m : \mathbb{F}_q \to \mathbb{N}$, $\Delta := \max_{x \in \mathbb{F}_q} \mathrm{wt}_m(x)$.*

- *Maximize:* $-\sum_{i \in [q]} p_i \log_q p_i,$

- *Subject to:* $\quad \sum_{i \in [q]} p_i = 1, \quad \sum_{i \in [q]} p_i \, \mathrm{wt}_m(i) = \omega \Delta,$

As in the case of the Lee weight, we can be easily verified that once we replace the optimization variable p_i from Problem 2.3.3 with r_i/n from (3.2), we recover the original formulation of the asymptotic surface area of a sphere from Proposition 2.3.3. Again, if we denote by $\tilde{\mathbf{p}} = (\tilde{p}_1, ..., \tilde{p}_q)$ the solution of Problem 2.3.3, the asymptotic value of the surface area of a sphere of radius $w = \lfloor \Delta n \omega \rfloor$, $\omega \in [0, 1]$, is calculated as $s_M(q, \omega) = -\sum_{i \in [q]} \tilde{p}_i \log_q \tilde{p}_i$.

Asymptotic sphere surface area as entropy

> **Definition 2.3.1** (Entropy)**.** *Let X be a discrete random variable that takes values in the alphabet $\mathcal{A}$, where $|\mathcal{A}| = q$, that is distributed according to $p : \mathcal{A} \to [0, 1]$. The entropy of X is then given as $H(X) = -\sum_{x \in \mathcal{A}} p(x) \log_q p(x)$.*

Let now X be a random variable that takes values in $\mathbb{F}_q$ and is distributed according to $p : \mathbb{F}_q \to [0, 1]$, and let $p(\cdot)$ be defined as $\forall x_i \in \mathbb{F}_q$, $p(x_i) := \tilde{p}_i$, for $i \in [q]$, where values $\{\tilde{p}_i\}_{i \in [q]}$ are obtained as the solution to Problem 2.3.3. The asymptotic sphere surface area is then given as:

$$s_M(q, \delta) = -\sum_{i \in [q]} \tilde{p}_i \log_q \tilde{p}_i = -\sum_{x_i \in \mathbb{F}_q} p(x_i) \log_q p(x_i) = H(X).$$

We observe here that the q-ary entropy function, defined in 2.3.1, is just a special case of the entropy where a random variable X_H takes elements from $\mathbb{F}_q$, and where $p(0) = 1 - \omega$, and $\forall x_i \in \mathbb{F}_q \setminus \{0\}$, $p(x_i) = \omega/(q-1)$. The asymptotic surface area of the Hamming sphere of radius $\lfloor n\omega \rfloor$, $\omega \in [0, 1]$, in the vector space $\mathbb{F}_q^n$, thus corresponds to the entropy of the random variable X_H. In the case of the Lee weight, we can also form a corresponding random variable X_L taking elements from $\mathbb{F}_q$ and satisfying $\forall x_i \in \mathbb{F}_q$, $p(x_i) = \tilde{p}_i$, where $\tilde{p}_i$ are solutions to Problem 2.3.3. Nevertheless, we do not have a closed-form solution to express the dependence of $p(x_i)$ from ω.

Ball volume

Given Proposition 2.3.3, we can calculate the volume of a ball of radius $w \in \mathbb{N}, w \leq \Delta n\omega$, where $\Delta := \max_{x \in \mathbb{F}_q} \mathrm{wt}_m(x)$, and $\omega \in [0,1]$, in the vector space $\mathbb{F}_q^n$ as:

$$\mathrm{vol}_M(q, n, d) := |\mathcal{B}_M(q, n, d)| = \sum_{j=0}^{w} \sum_{\mathbf{r} \in \mathcal{R}_j} \binom{n}{\mathbf{r}}.$$

As in the case of the Lee weight, we can show that the asymptotic value of the sphere surface area, $s_M(\cdot)$, reaches its maximum for $\omega_{\max} \in [0,1]$ given as $\omega_{\max} = \frac{\sum_{i \in [q]} \mathrm{wt}_m(i)}{q\Delta}$. We thus obtain the following expression for calculating the asymptotic volume of a ball of radius w:

$$\begin{aligned}
v_M(q, \omega) &= \lim_{n \to \infty} \frac{1}{n} \log_q \mathrm{vol}_M(q, n, \omega n) \\
&= \begin{cases} \lim_{n \to +\infty} \max_{\mathbf{r} \in \mathcal{R}_w} \left(-\sum_{i \in [q]} \frac{r_i}{n} \log_q \frac{r_i}{n} \right), & \text{if } \omega \leq \omega_{\max} \\ 1, & \text{else} \end{cases}.
\end{aligned}$$

2.3.4 Hamming space

In this subsection, we would like to show that, using results from Proposition 2.3.3, we can re-prove the well-known result on the (asymptotic) surface area of a sphere in the Hamming metric space. We thus first define the set of restricted compositions corresponding to the Hamming metric, namely:

$$\mathcal{R}_w := \{\mathbf{r} = (r_i)_{i \in [q]} \in \mathbb{N}_0^q \mid \sum_{i \in q} r_i = n, \sum_{i \in [q] \setminus \{0\}} r_i = w\},$$

where $n \in \mathbb{N}$, q is a prime number, and $w \in \mathbb{N}$ satisfies $w = \lfloor \omega n \rfloor$, $\omega \in [0,1]$. We then observe that the two conditions in the definition of $\mathcal{R}_w$ are equivalent to the following two conditions:

$$r_0 = n - w, \quad r_1 + \cdots + r_{q-1} = w.$$

Surface area We thus obtain the surface area of a sphere in the Hamming space:

$$
\operatorname{surf}_q(n, w) = \sum_{\mathbf{r} \in \mathcal{R}_w} \binom{n}{\mathbf{r}} = \sum_{r_1 + \cdots + r_{q-1} = w} \binom{n}{n - w, r_1, \ldots, r_{q-1}}
$$

$$
= \sum_{r_1 + \cdots + r_{q-1} = d} \frac{n!}{(n - w)!\, r_1! \ldots r_{q-1}!} \cdot \frac{w!}{w!}
$$

$$
= \frac{n!}{(n - w)!w!} \sum_{r_1 + \cdots + r_{q-1} = w} \frac{w!}{r_1! \ldots r_{q-1}!}
$$

$$
= \binom{n}{w} \sum_{r_1 + \cdots + r_{q-1} = w} \binom{d}{r_1, \ldots, r_{q-1}} = \binom{n}{w}(q - 1)^w,
$$

where $\sum_{r_1 + \cdots + r_{q-1} = w} \frac{w!}{r_1! \ldots r_{q-1}!} = (q - 1)^d$ is given by the *multinomial theorem* and it is known as the sum of multinomial coefficients (see, for example, [Ber10]).

Asymptotic sphere surface area To calculate the asymptotic sphere surface area, for the Hamming weight, we observe the following constrained convex optimization problem.

Problem 2.3.4. *Let $\mathbf{p} \in \mathbb{R}_+^q$, $\mathbf{p} = (p_i)_{i \in [q]}$, $\omega \in [0, 1]$.*

- *Maximize:* $-\sum_{i \in [q]} p_i \log_q p_i,$

- *Subject to:* $\sum_{i \in [q]} p_i = 1, \quad \sum_{i \in [q]} p_i \operatorname{wt}_h(i) = \omega.$

We then derive the Lagrangian function, $\mathcal{L}(\mathbf{p}, \lambda_1, \lambda_2)$, corresponding to Problem 2.3.4:

$$
\mathcal{L}(\mathbf{p}, \lambda_1, \lambda_2) = -\sum_{i \in [q]} p_i \log_q p_i - \lambda_1 \Big(\sum_{i \in [q]} p_i - 1\Big) - \lambda_2 \Big(\sum_{i \in [q]} p_i \operatorname{wt}_h(i) - \omega\Big).
$$

to obtain the following system of equations:

$$\frac{\partial \mathcal{L}}{\partial p_0} = (-\log_q p_0 - 1/\ln(q)) - \lambda_1 = 0 \tag{2.3}$$

$$\frac{\partial \mathcal{L}}{\partial p_1} = (-\log_q p_1 - 1/\ln(q)) - \lambda_1 - \lambda_2 = 0 \tag{2.4}$$

$$\cdots \tag{2.5}$$

$$\frac{\partial \mathcal{L}}{\partial p_{q-1}} = (-\log_q p_{q-1} - 1/\ln(q)) - \lambda_1 - \lambda_2 = 0 \tag{2.6}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_1} = -\sum_{i \in [q]} p_i + 1 = 0 \tag{2.7}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_2} = -\sum_{i \in [q] \setminus \{0\}} p_i + \omega = 0 \tag{2.8}$$

We solve the system as follows:

$$(2.8) \Rightarrow \sum_{i \in [q] \setminus \{0\}} p_i = \omega$$

$$(2.7) \Rightarrow p_0 + \sum_{i \in [q] \setminus \{0\}} p_i - 1 = 0 \quad \Rightarrow \quad p_0 = 1 - \omega$$

$$(2.4) - (2.6) \Rightarrow \log_q \frac{p_1}{p_2} = \log_q \frac{p_2}{p_3} = \cdots = \log_q \frac{p_{q-1}}{p_{q-2}} = 0$$

$$\Rightarrow p_1 = p_2 = \cdots = p_{q-1}$$

$$(2.8), (2.3) - (2.7) \Rightarrow p_1 = p_2 = \cdots = p_{q-1} = \omega/(q-1).$$

Finally, we obtain the asymptotic surface area of a Hamming ball of radius $w = \lfloor \omega n \rfloor$, $\omega \in [0, 1]$, given as:

$$s_H(q, \omega) = -\sum_{i \in [q]} p_i \log_q p_i = -(1-\omega) \log_q (1-\omega) - \omega \log_q \frac{\omega}{q-1} = H_q(\omega).$$

CHAPTER 3

INFORMATION SET DECODING

The information set decoding framework is known as the set of the most efficient algorithms attacking the syndrome decoding problem in the wide range of parameters, namely, for values of $R > 0.3$.[1] In this chapter, we use this framework to analyze the complexity of the generalized version of the syndrome decoding problem introduced in the previous chapter. These results appear in [CDE21].

---

[1]For $R < 0.3$, recent improvements in the field showed that the statistical decoding [Car+22] is more efficient in solving the syndrome decoding problem.

3.1 Information set decoding preliminaries

The *information set decoding* (ISD) is a common name for the set of algorithms that rely on the linear algebra properties of linear codes to gain speed-up in solving the syndrome decoding problem. Eugene Prange introduced the original algorithm in [Pra62] and, over the years, it was simplified, generalized, and improved by the work of many. The framework presented in Definition 3.1.1, introduced by Finiazs and Sendrier in [FS09], encompasses different ISD variants used in this thesis. The framework was originally designed for solving 2-SDP_H and then further generalized to an arbitrary prime alphabet by Peters in [Pet10]. For the purpose of this thesis, we generalize it to an arbitrary elementwise weight function $\mathrm{wt}_M(\cdot)$ (along with an arbitrary alphabet size).

Definition 3.1.1 (Information set decoding framework)**.** *The information set decoding framework comprises the following four steps:*

1. *permutation step: The algorithm samples a permutation matrix* $\mathbf{P} \in \mathbb{F}_2^{n \times n}$ *as* $\mathbf{P} \xleftarrow{\$} \Pi$.

2. *decomposition step: For a fixed* $l \in \mathbb{N}$, *the algorithm performs the Gaussian elimination on the first* $n - k - l$ *columns of the permuted parity check matrix* $\mathbf{HP}$. *The goal is to obtain the decomposition,* $\bar{\mathbf{H}} \in \mathbb{F}_q^{(n-k) \times n}$, *given as:*

$$\bar{\mathbf{H}} = \begin{pmatrix} \mathbf{I} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} = \mathbf{UHP},$$

such that $\mathbf{I}$ *is the identity matrix of rank* $n - k - l$, $\mathbf{0}$ *is a matrix of all zeros in* $\mathbb{F}_q^{l \times (n-k-l)}$, $\mathbf{H}_1 \in \mathbb{F}_q^{(n-k-l) \times (k+l)}$, *and* $\mathbf{H}_2 \in \mathbb{F}_q^{l \times (k+l)}$. *If* $\mathbf{I}$ *is of rank smaller than* $n - k - l$, *the algorithm returns to the first step. Otherwise, it decomposes the syndrome* $\mathbf{s}$ *as follows:*

$$\bar{\mathbf{s}} = \mathbf{Us} = \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{pmatrix}$$

---

where $\bar{\mathbf{s}}$ is a decomposition of the syndrome, $\mathbf{U} \in \mathbb{F}_q^{(n-k)\times(n-k)}$ corresponds to elementary row operations performed during Gaussian elimination, and $\mathbf{s}_1 \in \mathbb{F}_q^{n-k-l}$, $\mathbf{s}_2 \in \mathbb{F}_q^l$.

3. *mSDP step:* For a fixed $d \in \mathbb{N}$, $d \leq w$, the algorithm finds multiple solutions to the syndrome decoding sub-instance given on the inputs $(\mathbf{H}_2, \mathbf{s}_2, d)$.

4. *test step:* For each $\mathbf{e}_2$ found as a solution to the syndrome decoding sub-instance, the algorithm calculates $\mathbf{e}_1 = \mathbf{e}_2 \mathbf{H}_1^T - \mathbf{s}_1$ and checks if $\mathrm{wt}_M \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{0} \end{pmatrix} = w - d$. If no $\mathbf{e}_1$ is of weight $w - d$, the algorithm returns to the first step. Otherwise, it returns:

$$\mathbf{e} = \mathbf{P}\bar{\mathbf{e}} = \mathbf{P}\begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix},$$

where $\bar{\mathbf{e}} = \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix}$ is a decomposition of a permuted error vector.

Let us first verify that the algorithms in this framework give a correct solution to the syndrome decoding problem. We can then explain the intuition behind different steps and estimate the running time of different algorithms within the framework.

Correctness of the ISD framework. Let us assume that after the second step, an algorithm in the ISD framework succeeds in obtaining the desired decomposition of the permuted parity check matrix $\mathbf{HP}$ (which indeed happens with some constant probability). We then obtain a new instance of the syndrome decoding problem given on the inputs $(\bar{\mathbf{H}}, \bar{\mathbf{s}}, w)$ and we would like to show that finding a solution to this instance yields a solution to the original instance, given on the input $(\mathbf{H}, \mathbf{s}, w)$.

From the problem definition, we know that $\bar{\mathbf{e}} \in \mathbb{F}_q^n$ of weight w, satisfying $\bar{\mathbf{s}} = \bar{\mathbf{H}}\bar{\mathbf{e}}$ is a solution to the $(\bar{\mathbf{H}}, \bar{\mathbf{s}}, w)$ instance. Equivalently, a solution $\bar{\mathbf{e}} =$

$\begin{pmatrix} \mathbf{e}_1, \\ \mathbf{e}_2 \end{pmatrix} \in \mathbb{F}_q^{n-k-l} \times \mathbb{F}_q^{k+l}$, satisfies:

$$\begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} \begin{pmatrix} \mathbf{e}_1, \\ \mathbf{e}_2 \end{pmatrix}, \quad \mathrm{wt}_M(\bar{\mathbf{e}}) = w.$$

Given that the weight function is elementwise, we have that $\mathrm{wt}_M \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix} = \mathrm{wt}_M \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{0} \end{pmatrix} + \mathrm{wt}_M \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_2 \end{pmatrix} = w$. With a slight abuse of notation, we then simplify the previous expression to obtain $\mathrm{wt}(\mathbf{e}_1) + \mathrm{wt}_M(\mathbf{e_2}) = w$. Finding a solution $\bar{\mathbf{e}}$ then comes down to solving the following system of linear equations:

$$\mathbf{e}_1 + \mathbf{H}_1 \mathbf{e}_2 = \mathbf{s}_1 \tag{3.1}$$

$$\mathbf{H}_2 \mathbf{e}_2 = \mathbf{s}_2, \tag{3.2}$$

with an additional constraint given as $\mathrm{wt}_M(\mathbf{e}_1) + \mathrm{wt}_M(\mathbf{e_2}) = w$.

In step 3, the algorithm then finds vectors $\mathbf{e}_2 \in \mathbb{F}_q^{k+l}$ satisfying 3.2 and $\mathrm{wt}_M(\mathbf{e}_2) = d$, where $d \in \mathbb{N}$. From 3.1, we know that $\mathbf{e}_1$ can be calculated as $\mathbf{e}_1 = \mathbf{s}_1 - \mathbf{H}_1 \mathbf{e}_2$. If now $\mathrm{wt}_M(\mathbf{e}_1) = w - d$, we have that $\mathrm{wt}_M(\mathbf{e}_1) + \mathrm{wt}_M(\mathbf{e}_2) = w$. In step 4, the algorithm thus calculates $\mathbf{e}_1$ and checks if its weight is $w - d$. If that is the case, the algorithm obtains $\bar{\mathbf{e}}$ that satisfies $\bar{\mathbf{s}} = \bar{\mathbf{H}}\bar{\mathbf{e}}$ and $\mathrm{wt}_M(\bar{\mathbf{e}}) = w$ and yields a solution to a syndrome decoding instance on the inputs $(\bar{\mathbf{H}}, \bar{\mathbf{e}}, w)$. Consequently, $\mathbf{e} = \mathbf{P}\bar{\mathbf{e}}$ is of weight w and satisfies

$$\mathbf{He} = (\mathbf{U}^{-1}\bar{\mathbf{H}}\mathbf{P}^{-1})(\mathbf{P}\bar{\mathbf{e}}) = \mathbf{U}^{-1}\bar{\mathbf{H}}\bar{\mathbf{e}} = \mathbf{U}^{-1}\bar{\mathbf{s}} = \mathbf{s},$$

so $\mathbf{e}$ is a solution to the original instance of the syndrome decoding problem given on the inputs $(\mathbf{H}, \mathbf{s}, w)$. $\qquad\square$

Intuition behind the framework An intuition behind the different steps of the framework can be presented as follows.

- *permutation*: The permutation step plays the role of the *guessing* step described in the original ISD algorithm. The original algorithm *guesses*

the information set, i.e. the set of coordinates of a codeword that were not compromised by errors during transmission or, equivalently, the set of zero-element coordinates of the error vector. As the permutation $\mathbf{P}$ is sampled uniformly at random, the last $k + l$ positions of the permuted error vector $\bar{\mathbf{e}} = \mathbf{P}^{-1}\mathbf{e}$ correspond to randomly sampled positions of the error vector $\mathbf{e}$ whose overall weight is d, where $d \leq w$. Equivalently, the algorithm could have sampled uniformly at random a set $\mathcal{G} \in [n]$ of $k + l$ elements corresponding to the coordinates of $k + l$ positions of $\mathbf{e}$.

- *decomposition*: Given $\mathcal{G} \subseteq [n]$ and $d \in \mathbb{N}$, the original algorithm aims to obtain the parity check matrix in the standard form and, consequently, forms a sub-instance of the syndrome decoding problem. Solutions of this sub-instance are of weight d and satisfy the initial constraint $\mathbf{s} = \mathbf{H}\mathbf{e}$ for the set of positions given by $\mathcal{G}$. These solutions thus potentially yield a solution to the original problem. Namely, if the initial constraints on the $[n] \setminus \mathcal{G}$ positions are also satisfied, i.e. if $\mathbf{H}\mathbf{e} = \mathbf{s}$ at the rest of positions and $\mathrm{wt}_M((e_i)_{i \in [n] \setminus \mathcal{G}}) = w - d$, a solution of the original problem is obtained from the solution of the sub-instance with a polynomial overhead.

- *mSDP*: In this step, the algorithm finds multiple solutions to the sub-instance from the previous step.[2]

- *test*: Finally, the algorithm checks if any of the solutions found in the previous step yields a solution to the original problem. It does so by calculating the error vector at positions $[n] \setminus \mathcal{G}$ and checking if its weight is $w - d$. If not, the algorithm returns to the *guess* step.

---

[2]Notice here that, in contrast to the original problem where the goal is to find any solution, the goal here is to find as much as possible solutions.

3.2   Probability of finding a solution

It is not hard to see that the probability that the algorithm finds a solution at the end of an iteration of the above-presented framework depends on the probability that the initial guess was correct, as well as on the probability the decomposition was successful. Apart from that, the probability of finding a solution in one iteration depends also on the number of solutions that exist for the original instance, as well as on the number of solutions that are found in the third step. Taking all of this into account, we derive the expression for calculating the probability of finding a solution to the generalized syndrome decoding problem in one iteration of the ISD framework. We present it in the following lemma.

Lemma 3.2.1. *Let* $\operatorname{surf}_M(q, n, w)$ *and* $\operatorname{surf}_M(q, n - k - l, w - d)$, *respectively, be a surface area of a sphere of radius* w *in a vector space* $\mathbb{F}_q^n$ *and a sphere of radius* $w - d$ *in a vector space* $\mathbb{F}_q^{n-k-l}$. *Let then* $|\mathcal{L}|$ *be the expected number of solutions found in the* $mSDP$ *step of the* ISD *framework. The probability that a solution is found after one iteration of an algorithm within the framework,* $p \in [0, 1]$, *is then given as:*

$$p = \min \left(1, \ \frac{\operatorname{surf}_M(q, n - k - l, w - d)}{\max \left(q^{n-k}, \operatorname{surf}_M(q, n, w) \right) q^{-l}} |\mathcal{L}| \right).$$

Proof. Let us assume that the algorithm succeeds in decomposing the permuted parity-check matrix (which happens with some constant probability). After the decomposition step, we thus have:

$$\bar{\mathbf{H}} = \mathbf{UHP} = \begin{pmatrix} \mathbf{I} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix}, \quad \bar{\mathbf{s}} = \mathbf{Us} = \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{pmatrix},$$

that yield a potential solution, $\bar{\mathbf{e}} = \mathbf{P}^{-1}\mathbf{e} = \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix}$. We will thus refer to the instance $(\bar{\mathbf{H}}, \bar{\mathbf{s}}, w)$ as the *original A-SDP* as it is equivalent to a truly original instance given on the input $(\mathbf{H}, \mathbf{s}, w)$.

Let then $\bar{\mathcal{E}}$ be the set of all the solutions to the original A-SDP instance, defined as:

$$\bar{\mathcal{E}} := \{\bar{\mathbf{e}} \in \mathbb{F}_q^n \mid \bar{\mathbf{e}}\bar{\mathbf{H}}^T = \bar{\mathbf{s}}, \mathrm{wt}_M(\bar{\mathbf{e}}) = w\}.$$

By definition of A-SDP, we know there exists at least one solution in $\bar{\mathcal{E}}$, so $\bar{\mathcal{E}} \neq \emptyset$ and thus $|\bar{\mathcal{E}}| \neq 0$. The expected size of the set is then calculated as the expected size of the coset corresponding to the syndrome $\bar{\mathbf{s}}$ with the ball of radius w:

$$|\bar{\mathcal{E}}| = \max\left(1, \frac{\mathrm{surf}_M(q, n, w)}{q^{n-k}}\right) = \frac{\max\left(q^{n-k}, \mathrm{surf}_M(q, n, w)\right)}{q^{n-k}}.$$

Let then $\mathcal{E}_{12}$ be a set of all error vectors with a weight distribution according to the initial guess. We define it as follows:

$$\mathcal{E}_{12} := \{(\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_q^{n-k-l} \times \mathbb{F}_q^{k+l} \mid \mathrm{wt}_M(\mathbf{e}_1, \mathbf{0}) = w - d, \mathrm{wt}_M(\mathbf{e}_2, \mathbf{0}) = d\},$$

The probability that the initial guess was correct is then given as:

$$p_{\mathrm{init}} = Pr[(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{E}_{12} \mid \mathrm{wt}(\mathbf{e}_1, \mathbf{e}_2) = w]$$
$$= \frac{\mathrm{surf}_M(q, n - k - l, w - d)\,\mathrm{surf}_M(q, k + l, d)}{\mathrm{surf}_M(q, n, w)}.$$

Let now $\mathcal{E}_2$ be the set of all solutions to the SDP sub-instance, defined as follows:

$$\mathcal{E}_2 := \{\mathbf{e}_2 \in \mathbb{F}_q^{k+l} \mid \mathbf{e}_2\mathbf{H}_2^T = \mathbf{s}_2, \mathrm{wt}_M(\mathbf{e}_2) = d\}.$$

Its expected size is given as[3]:

$$|\mathcal{E}_2| = \frac{\mathrm{surf}_M(q, k + l, d)}{q^l}.$$

For a reasonable choice of parameters l and d, we expect that $|\mathcal{E}_2| \neq 0$ with high probability, which implies that there exists a solution to the SDP sub-instance. Given a solution to the sub-instance, $\mathbf{e}_2 \in \mathcal{E}_2$, the probability that

---

[3]Note here that, since the problem is actually an instance of a general SDP, and not an instance of A-SDP, there is no promise that a solution exists.

it yields a solution to the original A-SDP instance is calculated as:

$$p_1 = Pr[(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{E}_{12} \mid \mathrm{wt}(\bar{\mathbf{e}}) = w, \mathbf{e}_2 \in \mathcal{E}_2] = \frac{p_{\mathrm{init}}}{|\mathcal{E}_2|}$$

$$= \frac{\mathrm{surf}_M(q, n-k-l, w-d)\,\mathrm{surf}_M(q, k+l, d)}{\mathrm{surf}_M(q, n, w)} \cdot \frac{q^l}{\mathrm{surf}_M(q, k+l, d)}$$

$$= \frac{\mathrm{surf}_M(q, n-k-l, w-d)q^l}{\mathrm{surf}_M(q, n, w)}.$$

Let us now denote by $\mathcal{L}$ the set of all solutions to the SDP sub-instance found in the third step[4], and let $|\mathcal{L}|$ be its expected size. The probability that any of these solutions yield a solution to the original A-SDP instance is then given as[5]:

$$p_{\mathcal{L}} = \frac{\mathrm{surf}_M(q, n-k-l, w-d)q^l}{\mathrm{surf}_M(q, n, w)}|\mathcal{L}|.$$

Finally, the probability of finding any solution to the original A-SDP instance, given any solution found in the $mSDP$ step is calculated as:

$$p = 1 - (1 - p_{\mathcal{L}})^{|\bar{\mathcal{E}}|} \approx \min\left(1,\ |\mathcal{E}|p_{\mathcal{L}}\right)$$

$$= \min\left(1,\ \ \max\left(1, \frac{\mathrm{surf}_M(q, n, w)}{q^{n-k}}\right) \cdot \frac{\mathrm{surf}_M(q, n-k-l, w-d)q^l}{\mathrm{surf}_M(q, n, w)}|\mathcal{L}|\right)$$

$$= \min\left(1,\ \frac{\mathrm{surf}_M(q, n-k-l, w-d)}{\min\left(q^{n-k}, \mathrm{surf}_M(q, q, n, w)\right)q^{-l}}|\mathcal{L}|\right).$$

$\square$

This probability determines the expected number of iterations before the solution is reached with a high probability. Different ISD algorithms thus use different strategies to increase this probability while maintaining or even decreasing the cost of one iteration of the algorithm. After the analysis of the running time of these algorithms, it will become apparent that these two values cannot be minimized simultaneously. Naturally, finding the best trade-off between the two is the major goal of algorithm designers working on this framework.

[4]Notice here that $\mathcal{L} \subseteq \mathcal{E}_2$, i.e. it is a sub-set of all solutions to the SDP sub-instance found by the algorithm.

[5]This follows from the linearity of expectations.

3.3 Classical ISD algorithms

We now present four classical ISD algorithms used in this thesis to determine the average-case complexity of the syndrome decoding problem. The choice of these four is driven by the overall efficiency of the algorithms, as well as by the relative simplicity of the analysis of their running time. Along with the descriptions of the algorithms, we show the running time of each algorithm.

In our analysis, we focus on the evolution of the original algorithm, presented in [Pra62], to the version that we refer to as *Wagner's algorithm*. More specifically, we present the latter three ISD algorithms as iterative refinements of the original algorithm. Each of these refinements decreased the expected running time by providing better trade-offs between the cost of one iteration of the algorithm and the probability of finding a solution in one iterations. We then suggest an alternative view that suggests that all four algorithms we presented can be seen as one algorithm with different values of parameters.

General setting   In the classical setting, the running time $t_C : \mathbb{N}^3 \to \mathbb{N}$, given as a function of the code length, n, the code dimension, k, and the weight w, can be calculated by adding up the running time of each step of the framework. If we denote by $\kappa \in \mathbb{N}$ the overall cost of one iteration, and by $\kappa_i \in \mathbb{N}$, $i \in \{1, 2, 3, 4\}$ the cost of each step, we thus obtain:

$$t_C(n, k, w) = \frac{\kappa}{p} = \frac{\kappa_1 + \kappa_2 + \kappa_3 + \kappa_4}{p},$$

where p is the probability of finding a solution in one iteration of the algorithm. Finding the best trade-off thus comes down to increasing the value p and either maintaining or decreasing the values of κ_i.

3.3.1 Prange's algorithm

Prange's algorithm, introduced in [Pra62], is considered to be the first ISD algorithm. In its original version, it solves A-SDP$_H$ over a binary alphabet and the Hamming weight. In fact, it can be seen as a special case of the previously introduced framework where both $l = 0$ and $d = 0$. In Algorithm 1, we

present an adapted version of the original algorithm that now solves A-SDP over the finite field of size q and arbitrary elementwise function $\mathrm{wt}_M(\cdot)$. The running time of each step of the algorithm is given along with its description, and the overall running time is given in Proposition 3.3.1.

Algorithm 1 Prange's algorithm (PRANGE)	*Running time*
Input: $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, $w \in \mathbb{N}$	
Output: $\mathbf{e} \in \mathcal{S}_M(q, n, w)$ s.t. $\mathbf{He} = \mathbf{s}$	
1: **loop**	
2: $\quad \mathbf{P} \xleftarrow{\$} \Pi$	$\triangleright O(n \log_2 n)$
3: $\quad \mathbf{UHP} = \begin{pmatrix} \mathbf{I} & \mathbf{H}_1 \end{pmatrix} \leftarrow \mathrm{GE}(\mathbf{HP},\ n-k)$	$\triangleright O(n(n-k)^2)$
4: $\quad$ **if** $\mathrm{rank}(\mathbf{I}) < n-k$ **then**	$\triangleright O(n-k)$
5: $\quad\quad$ **goto** 2:	$\triangleright O(1)$
6: $\quad$ **else**	$\triangleright O(1)$
7: $\quad\quad \mathbf{s}_1 \leftarrow \mathbf{Us}$	$\triangleright O((n-k)^2)$
8: $\quad$ **end if**	
9: $\quad \mathbf{e}_1 \leftarrow \mathbf{s}_1$	$\triangleright O(n-k)$
10: $\quad$ **if** $\mathrm{wt}_M(\mathbf{e}_1) = w$ **then**	$\triangleright O(n-k)$
11: $\quad\quad$ **return** $\mathbf{e} \leftarrow \mathbf{P}\begin{pmatrix} \mathbf{e}_1 \\ \mathbf{0} \end{pmatrix}$	$\triangleright O(n \log_2 n)$
12: $\quad$ **end if**	
13: **end loop**	

Proposition 3.3.1 (Running time of Prange's algorithm). *The running time of Prange's algorithm, $t_C^P : \mathbb{N}^3 \to \mathbb{N}$, is given as:*

$$t_C^P(n, k, w) = O\left(\frac{n(n-k)^2}{p}\right),$$

where

$$p = \min\left(1,\ \frac{\mathrm{surf}_M(q, n-k, w)}{\min\left(q^{n-k},\ \mathrm{surf}_M(q, n, w)\right)}\right).$$

Proof. The running time of the algorithm is given as:

$$t_C^P = \frac{\kappa}{p} = \frac{\kappa_1 + \kappa_2 + \kappa_4}{p} = O\left(\frac{n\log_2 n + n(n-k) + n\log_2 n}{p}\right),$$

where p is obtained from Lemma 3.2.1. For values of parameters relevant in practice, we have $\log_2 n < (n-k)$. The running time of one iteration of the algorithm is thus dominated by the cost of Gaussian elimination performed in step 2 of the framework, i.e. in line 3 of the above-described algorithm. It gives the upper bound of the cost of one iteration, $\kappa = O(n(n-k)^2)$, and thus proves the proposition. □

The cost of one iteration of Prange's algorithm is then as small as we could hope for[6], but the probability of obtaining a solution in one iteration is rather small, too. In particular, for the parameter values relevant to cryptography, this probability is exponentially small. We thus look for a better trade-off between the iteration cost and the probability of finding a solution in one iteration.

3.3.2   Lee-Brickel's algorithm

To improve the trade-off given by Prange's algorithm, Lee and Brickel in [LB88] suggested an algorithm for which $l = 0$ but the choice of d is more flexible. As in the case of Prange's algorithm, and the rest of the algorithms described in this section, the original algorithm was designed for solving an instance of A-SDP over a binary field and the Hamming weight. In Algorithm 2, we present a version of Lee-Brickel's algorithm designed for solving A-SDP over the finite field of size q and arbitrary elementwise function $\text{wt}_M(\cdot)$. Along with the algorithm description, we present the cost of each step.

Algorithm 2 Lee-Brickel's algorithm (LEE_BRICKEL) *Running time*

Input: $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_q^{n-k}, w, d \in \mathbb{N}$
Output: $\mathbf{e} \in \mathcal{S}_M(q, n, w)$ s.t. $\mathbf{He} = \mathbf{s}$

---

[6]As $R = k/n$ is a constant, the cost is $O(n(n - Rn))$, which is essentially polynomial in $n\log_2 q$.

1: **loop**
2:     $\mathbf{P} \xleftarrow{\$} \Pi$ ▷ $O(n \log_2 n)$
3:     $\mathbf{UHP} = \begin{pmatrix} \mathbf{I} & \mathbf{H}_1 \end{pmatrix} \leftarrow \text{GE}(\mathbf{HP}, \ n-k)$ ▷ $O(n(n-k)^2)$
4:     **if** $\text{rank}(\mathbf{I}) < n - k$ **then**   ▷ $O(n-k)$
5:         **goto** 2:   ▷ $O(1)$
6:     **else**   ▷ $O(1)$
7:         $\mathbf{s}_1 \leftarrow \mathbf{Us}$ ▷ $O((n-k)^2)$
8:     **end if**
9:     **for all** $\mathbf{e}_2 \in \mathcal{S}_M(q,k,d)$ **do**   ▷ $O(1)$
10:         $\mathbf{e}_1 \leftarrow \mathbf{e}_2\mathbf{H}_1^T - \mathbf{s}_1$ ▷ $O((n-k)k)$
11:         **if** $\text{wt}_M(\mathbf{e}_1) = w - d$ **then**   ▷ $O(n-k)$
12:             **return** $\mathbf{e} \leftarrow (\mathbf{e}_1, \ \mathbf{e}_2)\mathbf{P}^{-1}$ ▷ $O(n \log_2 n)$
13:         **end if**
14:     **end for**
15: **end loop**

If we now compare Prange's and Lee-Brickel's algorithms, we see that they differ only in the fourth step of the framework, i.e. lines 9-14 of Lee-Brickel's algorithm. While Prange's algorithm implicitly checks only $\mathbf{e}_2 = \mathbf{0}$, Lee-Brickel's algorithm checks all $\mathbf{e}_2 \in \mathcal{S}_M(q,k,d)$, i.e. it checks $|\mathcal{S}_M(q,k,d)| = \text{surf}_M(q,k,d)$ many vectors that potentially yield a solution to the original problem. This change implies the running time expressed via Proposition 3.3.2.

> **Proposition 3.3.2** (Running time of Lee-Brickel's algorithm)**.** *For a fixed $d \in \mathbb{N}$, the running time of Lee-Brickel's algorithm, $t_C^{LB} : \mathbb{N}^3 \to \mathbb{N}$, is given as:*
>
> $$t_C^{LB}(n,k,w) = O\left(\frac{n(n-k)^2 + \text{surf}_M(q,k,d)(n-k)k}{p} \right),$$

where

$$p = \min\left(1, \ \frac{\operatorname{surf}_M(q, n - k, w - d)}{\min\left(q^{n-k}, \ \operatorname{surf}_M(q, n, w)\right)} \operatorname{surf}_M(q, k, d)\right).$$

Proof. Similarly to the case of Prange's algorithm, the running time of Lee-Brickel's algorithm is given as:

$$
\begin{aligned}
t_C^{LB} &= \frac{\kappa_1 + \kappa_2 + \kappa_4}{p} \\
&= O\left(\frac{n \log_2 n + n(n-k) + \operatorname{surf}_M(q, k, d)(n \log_2 n + (n-k)k)}{p}\right),
\end{aligned}
$$

where p is obtained from Lemma 3.2.1. We observe, however, that $n \log_2$ is significantly smaller than $(n - k)k$ for the parameters of interest, implying that the overall cost of one iteration is $\kappa = O\big(n(n-k) + \operatorname{surf}_M(q, k, d)(n - k)k\big)$, which concludes the proof. $\square$

From Proposition 3.3.2, we see that the running time of one iteration of Lee-Brickel's algorithm is longer than the running time of one iteration of Prange's algorithm. The improvement brought by Lee-Brickel's algorithm comes from the increased probability of finding a solution in one iteration of the algorithm. It is worth noticing, however, that the improvement comes only with a proper choice of d and that, even with the optimal choice of d, it is still relatively small (at most polynomial in $n \log_2 q$). The major benefit of Lee-Brickel's approach, in fact, is the generalization introduced when d is added as an additional parameter, which provided a new direction for the development of the ISD algorithms.

3.3.3   Stern's/Dumer's algorithm

Stern/Dumer's algorithm, firstly presented in [Ste88] and then slightly improved in [Dum91], advances Lee-Brickel's algorithm by introducing the third step of the framework (line 9 of the algorithm). In this step, the algorithm searches for solutions to the sub-instance of SDP that is given on the input $(\mathbf{H}_2, \mathbf{s}_2, d)$. In Algorithm 3, we present a version of Stern's/Dumers's

algorithm that solves A-SDP over the finite field of size q and arbitrary elementwise function $\mathrm{wt}_M(\cdot)$. In Algorithm 4 and Algorithm 5, we present a sub-routine used to provide a list of solutions to the syndrome decoding sub-instance. Along with the algorithms' descriptions, we present each step's running time.

Algorithm 3 Stern/Dumer's algorithm (STERN_DUMER) *Running time*

Input: $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, $w \in \mathbb{N}$, $l, d \in \mathbb{N}$
Output: $\mathbf{e} \in \mathcal{S}_M(q, n, w)$ s.t. $\mathbf{He} = \mathbf{s}$

1: **loop**
2:    $\mathbf{P} \xleftarrow{\$} \Pi$ $\triangleright O(n \log_2 n)$
3:    $\mathbf{UHP} = \begin{pmatrix} \mathbf{I} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} \leftarrow \mathrm{GE}(\mathbf{HP}, n-k-l)$ $\triangleright O(n(n-k-l)^2)$
4:    **if** $\mathrm{rank}(\mathbf{I}) < n - k - l$ **then** $\triangleright O(n-k)$
5:        **goto** 2: $\triangleright O(1)$
6:    **else** $\triangleright O(1)$
7:        $\begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{pmatrix} \leftarrow \mathbf{Us}$ $\triangleright O((n-k)^2)$
8:    **end if**
9:    $\mathcal{L} \leftarrow \text{STERN\_DUMER\_SUB}(\mathbf{H}_2, \mathbf{s}_2, d)$ $\triangleright t_C^{SD\_sub}$
10:   **for all** $\mathbf{e}_2 \in \mathcal{L}$ **do** $\triangleright O(1)$
11:       $\mathbf{e}_1 \leftarrow \mathbf{e}_2 \mathbf{H}_1^T - \mathbf{s}_1$ $\triangleright O((n-k-l)(k+l))$
12:       **if** $\mathrm{wt}_M(\mathbf{e}_1) = w - d$ **then** $\triangleright O(n-k-l)$
13:           **return** $\mathbf{e} \leftarrow \mathbf{P} \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix}$ $\triangleright O(n \log_2 n)$
14:       **end if**
15:   **end for**
16: **end loop**

The first step of the general framework (line 2 of the Algorithm 3), namely, the permutation step is the same for all ISD algorithms. The difference in the second step is that, in the case of Lee-Brickel's algorithm, the Gaussian elimination is performed on the first $n - k$ columns of the permuted parity-check matrix $\mathbf{HP}$, while in the case of Stern/Dumer's algorithms, it is performed on only first $n-k-l$ columns of the permuted parity-check matrix. Furthermore, Lee-Brickel's algorithm does not have the third step (as $l = 0$, there is no sub-

Algorithm 4 Subroutine (STERN_DUMER_SUB)	*Running time*

Input: $\mathbf{H}_2 \in \mathbb{F}_q^{l \times (k+l)}$, $\mathbf{s}_2 \in \mathbb{F}_q^l$, $d \in \mathbb{N}$
Output: $\mathcal{L} \subseteq \mathbb{F}_q^{k+l}$
 1: **for all** $\mathbf{e}' \in \mathcal{S}_q^M(\frac{k+l}{2}, \frac{d}{2})$ **do**
 2:     $\mathcal{L}_0.\text{INSERT}\big((\mathbf{e}', \mathbf{0})\mathbf{H}_2^T\big)$ $\triangleright\ O((k+l)l)$
 3:     $\mathcal{L}_1.\text{INSERT}\big((\mathbf{0}, \mathbf{e}')\mathbf{H}_2^T - \mathbf{s}_2\big)$ $\triangleright\ O((k+l)l)$
 4: **end for**
 5: **return** $\mathcal{L} \leftarrow \text{MERGE-JOIN}(\mathcal{L}_0, \mathcal{L}_1, l)$ $\triangleright\ t_C^{\text{mj}}$

instance of SDP), and, in the fourth step, it checks if any of the error vectors $\mathbf{e}_2 \in \mathcal{S}_M(q, k+l, d)$ yields a solution to the original instance. Stern's/Dumer's algorithm, on the other hand, uses a sub-routine that first creates two lists of partial solutions to the sub-instance, according to the Shamir-Schroeppel approach [SS79], and then merges these two lists using MERGE-JOIN routine 5. The algorithm thus creates a list of solutions to the sub-instance (line 9) and checks if any of these yields a solution to the original problem (lines 10-15). The running time of the algorithm is given via Proposition 3.3.3.

Proposition 3.3.3 (Running time of Stern/Dumer's algorithm)**.** *For fixed* $l, d \in \mathbb{N}$, *the running time of Stern/Dumer's algorithm,* $t_C^{SD} \in \mathbb{N}$, *is given as:*

$$t_C^{SD}(n, k, w) = O\left(\frac{n(n-k-l)^2 + t_C^{SD\_sub} + |\mathcal{L}| \cdot (n-k-l)(k+l)}{p} \right),$$
(3.3)

where

$$|\mathcal{L}_0| = \text{surf}_q^M\left(\frac{k+l}{2}, \frac{d}{2} \right), \quad |\mathcal{L}| = \frac{\text{surf}_M(q, \frac{k+l}{2}, \frac{d}{2})^2}{q^l},$$
(3.4)

$$t_C^{SD\_sub} = O\left(|\mathcal{L}_0| \cdot (k+l)l + t^{\text{mj}} \right),$$
(3.5)

$$t^{\text{mj}} = O\left(\max\left(|\mathcal{L}_0|, |\mathcal{L}| \right) \cdot l \right).$$
(3.6)

Algorithm 5 Merge-join (MERGE-JOIN)	*Running time*
Input: $\mathcal{L}_0 \subseteq \mathbb{F}_q^n$, $\mathcal{L}_1 \subseteq \mathbb{F}_q^n$, $l \in \mathbb{N}$	
Output: $\mathcal{L} = \mathcal{L}_0 \bowtie_l \mathcal{L}_1 \subseteq \mathbb{F}_q^n$	
1: $\mathcal{L}_0.\text{SORT}()$, $\mathcal{L}_1.\text{SORT}()$	$\triangleright O\big(n(|\mathcal{L}_0| + |\mathcal{L}_1|)\big)$
2: $i \leftarrow 0$, $j \leftarrow |\mathcal{L}_1| - 1$	$\triangleright O(1)$
3: **while** $i < |\mathcal{L}_0|, j \geq 0$ **do**	$\triangleright O(1)$
4: **if** $\mathcal{L}_0[i] < \mathcal{L}_1[j]$ **then**	$\triangleright O(1)$
5: $i \leftarrow i + 1$	$\triangleright O(1)$
6: **else**	
7: **if** $\mathcal{L}_0[i] > \mathcal{L}_1[j]$ **then**	$\triangleright O(1)$
8: $j \leftarrow j - 1$	$\triangleright O(1)$
9: **else**	
10: $i_0, i_1 \leftarrow i$, $j_0, j_1 \leftarrow j$	$\triangleright O(1)$
11: **while** $i_1 < |\mathcal{L}_0|, \mathcal{L}_0[i_0] = \mathcal{L}_0[i_1]$ **do**	$\triangleright O(n)$
12: $i_1 \leftarrow i_1 + 1$	$\triangleright O(1)$
13: **end while**	
14: **while** $j_1 \geq 0, \mathcal{L}_1[j_0] = \mathcal{L}_1[j_1]$ **do**	$\triangleright O(n)$
15: $j_1 \leftarrow j_1 - 1$	$\triangleright O(1)$
16: **end while**	
17: **for** $i \in [i_0, i_1]$ **do**	$\triangleright O(1)$
18: **for** $j \in [j_1 + 1, j_0 + 1]$ **do**	$\triangleright O(1)$
19: $\mathcal{L}.\text{INSERT}\big(\mathcal{L}_0[i] + \mathcal{L}_1[j]\big)$	$\triangleright O(n)$
20: **end for**	
21: **end for**	
22: $i \leftarrow i_1, j \leftarrow j_1$	$\triangleright O(1)$
23: **end if**	
24: **end if**	
25: **end while**	

and

$$p = \min \left(1, \ \frac{\operatorname{surf}_M(q, n-k-l, w-d) \operatorname{surf}_M(q, \frac{k+l}{2}, \frac{d}{2})^2}{\min \left(q^{n-k}, \ \operatorname{surf}_M(q, n, w) \right)} \right). \quad (3.7)$$

Proof. Let us first observe that the running time of the first step of the framework, given as $\kappa_1 = O(n \log_2 n)$, is negligible in comparison to the running time of other steps. Therefore, the cost of one iteration comes down to:

$$t_C^W = \frac{\kappa_2 + \kappa_3 + \kappa_4}{p} = O\left(\frac{n(n-k-l)^2 + t_C^{SD\_sub} + |\mathcal{L}|(n-k-l)(k+l)}{p} \right),$$

where p is given by Lemma 3.2.1. Moreover, we know that MERGE-JOIN routine returns a list $\mathcal{L}$ with $|\mathcal{L}| = \frac{\operatorname{surf}_M(q, \frac{k+l}{2}, \frac{d}{2})^2}{q^l}$ elements on average. From Lemma 3.2.1, we then obtain the expression 3.8 that calculates the probability of finding a solution in one iteration of the algorithm. The running time of MERGE-JOIN, t_C^{mj}, is calculated as $O\left(\max(\operatorname{surf}_M(q, \frac{k+l}{2}, \frac{d}{2}), |\mathcal{L}|)l \right)$, which gives the expression 3.7. Given 3.7, we then calculate the running time of the Stern/Dumer's subroutine as

$$O\left(\operatorname{surf}_M(q, \frac{k+l}{2}, \frac{d}{2})(k+l)l + \max(\operatorname{surf}_M(q, \frac{k+l}{2}), \frac{d}{2}), |\mathcal{L}|)l \right),$$

as expressed by 3.6. Finally, given 3.5 and knowing the expected $|\mathcal{L}|$, the running time of the Stern/Dumer's algorithm is given by expression 3.4. $\quad \square$

Let us now compare the efficiency of Lee-Brickel's and Stern/Dumer's algorithms. We observe that the purpose of introducing the third step into Stern/Dumer's algorithm is to reduce the running time of its fourth step. Let us recall that Lee-Brickel's algorithm skips the third step and that, in the fourth step, the algorithm checks if any $\mathbf{e}_2 \in \mathcal{S}_M(q, k+l, d)$ yields a solution to the original problem. Stern/Dumer's algorithm, on the other hand, in step 3, finds solutions to the sub-instance of SDP and then, in step 4, checks if any of these yields a solution to the original problem. Moreover, in step 3, Stern/Dumer's algorithm obtains only solutions to the sub-instance with a particular property, i.e. only those that have weight equally distributed between two parts of the error vector, i.e. $d/2$ each. Consequently, in step 4, Stern/Dumer's algorithm checks only the subset of vectors $\mathbf{e}_2$ from Lee-Brickel's algorithm, so the number of checks is decreased.

The question now is whether this decrease implies a shorter overall running time of Stern/Dumer's algorithm in comparison to Lee-Brickel's. To answer this question, we need to consider the running time of step 3, as well as the probability of finding a solution in step 4.

After careful analysis of the trade-offs, it becomes apparent that the improvement of Stern/Dumer's algorithm over Lee-Brickel's algorithm crucially depends on the choice of parameters l and d. A particularly interesting choice of parameter l would be to take the value that makes the approximation $q^l \approx \mathrm{surf}_M(q, \frac{k+l}{2}, \frac{d}{2})$ as tight as possible. The running time of STERN_DUMER algorithm is then calculated as $t_C^{SD} = O\left(\frac{n(n-k-l)^2 + t_C^{SD\_sub} + q^l(n-k-l)(k+l)}{p} \right)$, where $t_C^{SD\_sub} = O\left(q^l(k+l)l\right)$. We observe that this choice implies that the expected number of solutions the sub-routine returns is $|\mathcal{L}| = q^l$, which further implies that each solution to the sub-instance is found in time $O((k+l)l)$. The running time of the third step of the framework is thus minimized.

3.3.4 Wagner's algorithm

The algorithm is created by combining Wagner's k-*tree* algorithm, introduced in [SS81], with Shamir's and Schroeppel's approach [SS79]. It generalizes the ideas of Stern and Dumer by introducing the k-tree algorithm in step 3 of the framework (line 9 of the algorithm). We will thus refer to it as *Wagner's algorithm*. In Algorithm 6, we present its version that solves A-SDP over the finite field of size q and arbitrary elementwise function $\mathrm{wt}_M(\cdot)$. In Algorithm 7, we present a subroutine that uses the k-tree algorithm to create a list of solutions to the syndrome decoding sub-instance. Along with the algorithms' descriptions, we present the running time of each step.

Algorithm 6 Wagner's algorithm (WAGNER)   *Running time*

Input: $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, $w \in \mathbb{N}$, $l, d, a \in \mathbb{N}$
Output: $\mathbf{e} \in \mathcal{S}_M(q, n, w)$ s.t. $\mathbf{He} = \mathbf{s}$

  1: **loop**

  2:     $\mathbf{P} \xleftarrow{\$} \Pi$ $\triangleright O(n \log_2 n)$

  3:     $\mathbf{UHP} = \begin{pmatrix} \mathbf{I} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} \leftarrow \mathrm{GE}(\mathbf{HP}, \ n-k-l)$ $\triangleright O(n(n-k-l)^2)$

4:	**if** $\mathrm{rank}(\mathbf{I}) < n - k - l$ **then**	$\triangleright\, O(n-k-l)$
5:	**goto** 2:	$\triangleright\, O(1)$
6:	**else**	$\triangleright\, O(1)$
7:	$\begin{pmatrix}\mathbf{s}_1 \\ \mathbf{s}_2\end{pmatrix} \leftarrow \mathbf{U}\mathbf{s}$	$\triangleright\, O((n-k)^2)$
8:	**end if**	
9:	$\mathcal{L} \leftarrow$ WAGNER_SUB$(\mathbf{H}_2,\ \mathbf{s}_2,\ d,\ a)$	$\triangleright\, t_C^{W\_sub}$
10:	**for all** $\mathbf{e}_2 \in \mathcal{L}$ **do**	$\triangleright\, O(1)$
11:	$\mathbf{e}_1 \leftarrow \mathbf{e}_2\mathbf{H}_1^T - \mathbf{s}_1$	$\triangleright\, O((n-k-l)(k+l))$
12:	**if** $\mathrm{wt}_M(\mathbf{e}_1) = w - d$ **then**	$\triangleright\, O(n-k-l)$
13:	**return** $\mathbf{e} \leftarrow \mathbf{P}\begin{pmatrix}\mathbf{e}_1 \\ \mathbf{e}_2\end{pmatrix}$	$\triangleright\, O(n\log_2 n)$
14:	**end if**	
15:	**end for**	
16:	**end loop**	

Algorithm 7 Subroutine (WAGNER_SUB)   *Running time*

Input: $\mathbf{H}_2 \in \mathbb{F}_q^{l\times(k+l)},\ \mathbf{s}_2 \in \mathbb{F}_q^l,\ d,a \in \mathbb{N}$
Output: $\mathcal{L} \subseteq \mathbb{F}_q^{k+l}$

1:	**for all** $\mathbf{e}' \in \mathcal{S}_q^M\big(\frac{k+l}{2^a}, \frac{d}{2^a}\big)$ **do**	$\triangleright\, O(1)$
2:	$\mathcal{L}_{(0,0)}.\text{INSERT}\big((\mathbf{e}',\mathbf{0},\mathbf{0},\ldots,\mathbf{0})\mathbf{H}_2^T\big)$	$\triangleright\, O((k+l)l)$
3:	$\mathcal{L}_{(0,1)}.\text{INSERT}\big((\mathbf{0},\mathbf{e}',\mathbf{0},\ldots,\mathbf{0})\mathbf{H}_2^T\big)$	$\triangleright\, O((k+l)l)$
4:	$\ldots$	
5:	$\mathcal{L}_{(0,2^a-2)}.\text{INSERT}\big(\mathbf{0},\mathbf{0},\ldots,\mathbf{e}',\mathbf{0})\mathbf{H}_2^T\big)$	$\triangleright\, O((k+l)l)$
6:	$\mathcal{L}_{(0,2^a-1)}.\text{INSERT}\big((\mathbf{0},\mathbf{0},\ldots,\mathbf{0},\mathbf{e}')\mathbf{H}_2^T - \mathbf{s}_2\big)$	$\triangleright\, O((k+l)l)$
7:	**end for**	
8:	**for all** $i \in [a]$ **do**	
9:	**for all** $j \in [2^{a-i-1}]$ **do**	
10:	$\mathcal{L}_{(i+1,j)} \leftarrow$ MERGE-JOIN$\big(\mathcal{L}_{(i,2j)},\ \mathcal{L}_{(i,2j+1)},\ (i+1)\frac{l}{a}\big)$	
11:	**end for**	
12:	**end for**	$\triangleright\, t_{\mathrm{mj}}$
13:	**return** $\mathcal{L}_{a,0}$	$\triangleright\, O(1)$

As already mentioned, Wagner's algorithm can be viewed as a generalization of Stern/Dumer's algorithm in which the first, second, and fourth steps of the framework are the same, while the third step (line 9 of the algorithm) is modified. In the case of Wagner's algorithm, in line 9, the algorithm calls a sub-routine that creates 2^a lists of partial solutions, instead of creating only 2 as in the case of Stern/Dumer's algorithm. In both cases, however, the lists are formed using the Shamir-Schroeppel approach and then merged using MERGE-JOIN routine. While Stern/Dumer's algorithm uses the joining routine at only one level, Wagner's algorithm uses the k-tree algorithm which applies MERGE-JOIN routine on multiple levels. An alternative view is then that Stern/Dumer's algorithm represents only a special case of Wagner's algorithm for which $a = 1$. This can also be seen from the running time of Wagner's algorithm given by Proposition 3.3.4. Namely, if we let $a = 1$ in the proposition, i.e. we fix the number of levels in Wagner's algorithm to 1, we recover the running time of Stern/Dumer's algorithm.

Proposition 3.3.4 (Running time of Wagner's algorithm). *For fixed* $l, d, a \in \mathbb{N}$, *the running time of Wagner's algorithm,* $t_C^W : \mathbb{N}^3 \to \mathbb{N}$, *is given as:*

$$t_C^W(n, k, w) = O\left(\frac{n(n - k - l)^2 + t_C^{W\_sub} + |\mathcal{L}|\,(n - k - l)(k + l)}{p}\right),$$
(3.8)

where

$$|\mathcal{L}_0| = surf_q^M\left(\frac{k + l}{2^a}, \frac{d}{2^a}\right), \quad |\mathcal{L}| = \frac{\mathrm{surf}_M(q, \frac{k+l}{2^a}, \frac{d}{2^a})^{2^a}}{q^{(2^a - 1)\frac{l}{a}}},$$
(3.9)

$$t_C^{W\_sub} = O\left(|\mathcal{L}_0|\,(k + l)l + t^{\mathrm{mj}}\right),$$
(3.10)

$$t^{\mathrm{mj}} = O\left(\max\left(\mathcal{L}_0, \mathcal{L}\right)l\right).$$
(3.11)

and

$$p = \min\left(1, \frac{\mathrm{surf}_M(q, n - k - l, w - d)\,\mathrm{surf}_M(q, \frac{k+l}{2^a}, \frac{d}{2^a})^{2^a}}{\min\left(q^{n-k}, \mathrm{surf}_M(q, n, w)\right)q^{\frac{(2^a - a - 1)l}{a}}}\right).$$
(3.12)

Proof. As in the case of Stern/Dumer's algorithm, the running time of the

first step of the framework is negligible in comparison to the running time of other steps, so the cost of one iteration comes down to:

$$t_C^W = \frac{\kappa_2 + \kappa_3 + \kappa_4}{p} = O\left(\frac{n(n-k-l)^2 + t_C^{W\text{-}sub} + |\mathcal{L}|(n-k-l)(k+l)}{p}\right),$$

where p is given by Lemma 3.2.1. Since MERGE-JOIN routine, in combination with the k-tree algorithm, returns a list $\mathcal{L}$ with $|\mathcal{L}| = \frac{\text{surf}_M(q, \frac{k+l^a}{2^a}, \frac{d}{2^a})^{2^a}}{q^{(2^a-1)\frac{l}{a}}}$ elements on average, from Lemma 3.2.1, we then obtain the expression 3.12 that calculates the probability of finding a solution in one iteration of the algorithm. As the running time of MERGE-JOIN, t_C^{mj}, is calculated as:

$$O\big(\max(\text{surf}_M(q, \frac{k+l}{2^a}, \frac{d}{2^a}), |\mathcal{L}|)l\big),$$

we obtain the expression 3.11. Given 3.11, we then calculate the running time of Wagner's subroutine as:

$$O\big(\text{surf}_M(q, \frac{k+l}{2^a}), \frac{d}{2^a})|(k+l)l + \max(\text{surf}_M(q, \frac{k+l}{2^a}), \frac{d}{2^a}), |\mathcal{L}|)l\big),$$

as expressed by 3.10. Finally, given 3.10 and knowing the expected $|\mathcal{L}|$, the running time of WAGNER algorithm is given by expression 3.9. $\qquad\square$

As in the case of Stern's/Dumer's algorithm, one strategy for minimizing the running time of the algorithm would be to set the parameters l, d, and a so that $\text{surf}_M(q, \frac{k+l}{2^a}, \frac{d}{2^a})^{2^a} \approx q^{(2^a-1)\frac{l}{a}}$. We thus minimize the running time of the third step of the framework, in which the algorithm finds the solutions to the syndrome decoding sub-instance, and thus obtain close to optimal (if not optimal) Wagner's algorithm.

Wagner's algorithm as a generalization

From the above-presented descriptions of classical ISD algorithms, we can conclude that all four algorithms can be observed as different versions of Wagner's algorithm. To see that, let us recall that Stern's/Dumer's algorithm can be observed as a special case of Wagner's algorithm for which $a = 1$ and values of l, d can be optimized. Similarly, Lee-Brickel's algorithm can be

observed as a bit more restrictive version of Wagner's algorithm in which a and l are fixed, i.e. $a = 1, l = 0$, and d can be optimized. Finally, Prange's algorithm can be observed as the most restrictive version for which all three parameters are fixed, namely, $a = 1, l = 0$, and $d = 0$. We will use this observation to simplify the analysis of the asymptotic running time of the four algorithms.

3.4 Quantum ISD algorithms

What we call the *quantum algorithms* are, in fact, hybrid classical-quantum algorithms that combine the above-presented classical ISD framework with quantum algorithms such as Grover's search, amplitude amplification, or quantum walks, described in Chapter 1. In the first paper suggesting that this approach was successful, namely, in[Ber10], the author combined Prange's algorithm with Grover's search to create a hybrid classical-quantum algorithm that solves binary A-SDP over the Hamming weight. An almost quadratic speed-up of the approach over (classical) Prange's algorithm encouraged further development of quantum ISD algorithms. Significant advances were made in [KT17] and [Kir18] afterward, where the authors combined more advanced classical ISD algorithms, such as [Bec+12], [MMT11], with Grover's search and *quantum walks*. In our work, we combine Wagner's algorithm with amplitude amplification and Grover's search. We thus obtain a hybrid classical-quantum algorithm that solves A-SDP over q-ary alphabet and elementwise weight function $\mathrm{wt}_M(\cdot)$. We refer to it as the *quantum Wagner's* algorithm.

Our approach To gain a speed up over classical algorithms, we modify the mSDP step of the framework. Namely, instead of returning a list of solutions to the syndrome decoding sub-instance, the mSDP step now returns a description of a function $f : m \to \mathbb{F}_q^{k+l}, m \geq \mathrm{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})$. On input $i \in \left[\mathrm{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1}) \right]$, $f(\cdot)$ outputs what would be the i-th element of the list returned in the mSDP of classical Wagner's algorithm. The promise is that if $f(\cdot)$ is evaluated on every input in $\left[\mathrm{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1}) \right]$, it returns all the solutions that would be in the list of solutions returned by the mSDP step of classical Wagner's algorithm. To be more specific, let us take $n, k, m \in \mathbb{N}$, let q be a prime number, and let $\mathrm{wt}_M(\cdot)$ be an elementwise weight function. We then introduce the following version of the syndrome decoding problem.

Problem 3.4.1 (Multi-solution Syndrome Decoding Problem, M-SDP).
Let $m \in \mathbb{N}$ satisfy $m \geq \mathrm{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})$. Given $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, and $w \in \mathbb{N}$, return a description of a function $f : [m] \to \mathbb{F}_q^{k+l}$ defined
$$as \; f(i) = \begin{cases} \mathcal{L}[i], \text{ if } i < |\mathcal{L}| \\ \mathbf{0}, \text{ otherwise} \end{cases} , where \, \mathcal{E} := \{\mathbf{e} \in \mathbb{F}_q^n \mid \mathbf{s} = \mathbf{e}\mathbf{H}^T, \mathrm{wt}_M(\mathbf{e}) = $$
$w\}$ and $\mathcal{L} \subseteq \mathcal{E}$.

In the hybrid classical-quantum setting, in the mSDP step, the algorithm thus solves an instance of M-SDP. The algorithm, along with the running time of each step, is presented in Algorithm 8. The sub-routines that perform the third step of the general framework are given in Algorithm 9 and 10, along with the running time of each of its steps.

Algorithm 8 Wagner's algorithm (QUANTUM_WAGNER) *Running time*

Input: $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, $w \in \mathbb{N}$, $l, d, a \in \mathbb{N}$
Output: $\mathbf{e} \in \mathbb{F}_q^n$ s.t. $\mathbf{He} = \mathbf{s}$ and $\mathrm{wt}_M(\mathbf{e}) = w$
1: **for all** $\mathbf{e}' \in \mathcal{S}_q^M(\frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})$ **do**
2:    $\mathcal{E}'.\mathrm{INSERT}(\mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}, \mathbf{e}')$ $\triangleright O(k+l)$
3: **end for**
4: **loop**
5:    $\mathbf{P} \xleftarrow{\$} \Pi$ $\triangleright O(n \log_2 n)$
6:    $\mathbf{UHP} = \begin{pmatrix} \mathbf{I} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} \leftarrow \mathrm{GE}(\mathbf{HP}, n-k-l)$ $\triangleright O(n(n-k-l)^2)$
7:    **if** $\mathrm{rank}(\mathbf{I}) < n-k-l$ **then** $\triangleright O(n-k-l)$
8:       **goto** 2: $\triangleright O(1)$
9:    **else** $\triangleright O(1)$
10:    $\begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{pmatrix} \leftarrow \mathbf{Us}$ $\triangleright O((n-k)^2)$
11:    **end if**
12:    $f(\cdot) \leftarrow \mathrm{QUANTUM\_WAGNER\_SUB}(\mathbf{H}_2, \mathbf{s}_2, d, a, \mathcal{E}')$ $\triangleright t_Q^{W\_sub}$

13:      **function** $g\big(\, i \in \big[\, \mathrm{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})\big]\big)$

14:          $\mathbf{e}_2 \leftarrow f(i)$ $\triangleright\ O(k+l)$

15:          $\mathbf{e}_1 \leftarrow \mathbf{e}_2 \mathbf{H}_1^T - \mathbf{s}_1$ $\triangleright\ O((n-k-l)(k+l))$

16:          **if** $\mathrm{wt}_M(\mathbf{e}_1) = w - d$ **then** $\triangleright\ O(n-k-l)$

17:              **return** 1 $\triangleright\ O(1)$

18:          **else** $\triangleright\ O(1)$

19:              **return** 0 $\triangleright\ O(1)$

20:          **end if**

21:      **end function**

22:      $i \leftarrow \textsc{grover}(g(\cdot))$ $\triangleright\ O\!\left(\sqrt{|\mathcal{L}_0|}\, t_g\right)$

23:      **if** $i < \mathrm{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})$ **then** $\triangleright\ O(1)$

24:          $\mathbf{e}_2 \leftarrow f(i)$ $\triangleright\ O(k+l)$

25:          $\mathbf{e}_1 \leftarrow \mathbf{e}_2 \mathbf{H}_1^T - \mathbf{s}_1$ $\triangleright\ O((n-k-l)(k+l))$

26:          **return** $\mathbf{e} \leftarrow \mathbf{P}\begin{pmatrix}\mathbf{e}_1\\\mathbf{e}_2\end{pmatrix}$ $\triangleright\ O(n\log_2 n)$

27:      **else**

28:          **goto** 5:

29:      **end if**

30: **end loop**

Algorithm 9 Subroutine (QUANTUM_WAGNER_SUB)   *Running time*

Input: $\mathbf{H}_2 \in \mathbb{F}_q^{l\times(k+l)}$, $\mathbf{s}_2 \in \mathbb{F}_q^l$, $d, a \in \mathbb{N}$, $\mathcal{E}' \subseteq \mathbb{F}_q^{k+l}$

Output: $f : \big[\, \mathrm{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})\big] \to \mathbb{F}_q^{k+l}$

1: **for all** $\mathbf{e}' \in \mathcal{S}_q^M\big(\frac{k+l}{2^a+1}, \frac{d}{2^a+1}\big)$ **do**

2:    $\mathcal{L}_{(0,0)}.\textsc{insert}\big((\mathbf{e}', \mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}, \mathbf{0}, \mathbf{0})\mathbf{H}_2^T\big)$ $\triangleright\ O((k+l)l)$

3:    $\mathcal{L}_{(0,1)}.\textsc{insert}\big((\mathbf{0}, \mathbf{e}', \mathbf{0}, \ldots, \mathbf{0}, \mathbf{0}, \mathbf{0})\mathbf{H}_2^T\big)$ $\triangleright\ O((k+l)l)$

4:    $\ldots$

5:    $\mathcal{L}_{(0,2^a-2)}.\textsc{insert}\big((\mathbf{0}, \mathbf{0}, \ldots, \mathbf{e}', \mathbf{0}, \mathbf{0})\mathbf{H}_2^T\big)$ $\triangleright\ O((k+l)l)$

6: **end for**

7: **for all** $i \in [a-1]$ **do**

8:      **for all** $j \in [2^{a-i-1}-1]$ **do**

9:          $\mathcal{L}_{(i+1,j)} \leftarrow$ MERGE-JOIN$(\mathcal{L}_{(i,2j)}, \mathcal{L}_{(i,2j+1)}, (i+1)\frac{l}{a})$

10:      **end for**

11: **end for** $\triangleright t^{\mathrm{mj}}$

12: **function** $f(i \in \left[\,\mathrm{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})\right])$

13:      $f(i) \leftarrow \mathcal{E}'[i]\mathbf{H}_2^T - \mathbf{s}_2$ $\triangleright O(k+l)$

14:      **for all** $i \in [a]$ **do**

15:          $f(i) \leftarrow$ FUNCTION-MERGE-JOIN$(\mathcal{L}_{(i,2^{a-i}-2)}, f(i), (i+1)\frac{l}{a})$

16:      **end for** $\triangleright t^{\mathrm{fmj}}$

17: **end function**

18: **return** $f(\cdot)$ $\triangleright O(1)$

Algorithm 10 Function-merge-join (FUNC-MERGE-JOIN) *Running time*

Input: $\mathcal{L}_0 \subseteq \mathbb{F}_q^n, \ m, l \in \mathbb{N}, \ j \in [m], \ f : [m] \to \mathbb{F}_q^n,$

Output: $g : [m] \to \mathbb{F}_q^n$

1: $\mathcal{L}_0.$SORT() $\triangleright O(n|\mathcal{L}_0|)$

2: $i \leftarrow 0$ $\triangleright O(1)$

3: **while** $i < |\mathcal{L}_0|$ **do** $\triangleright O(1)$

4:      **if** $\mathcal{L}_0[i] < f(j)$ **then** $\triangleright O(1)$

5:          $i \leftarrow i + 1$ $\triangleright O(1)$

6:      **else**

7:          $i_0, i_1 \leftarrow i$ $\triangleright O(1)$

8:          **while** $i_1 < |\mathcal{L}_0|, \mathcal{L}_0[i_0] = \mathcal{L}_0[i_1]$ **do** $\triangleright O(n)$

9:              $i_1 \leftarrow i_1 + 1$ $\triangleright O(1)$

10:          **end while**

11:          $i \leftarrow i_1$ $\triangleright O(1)$

12:          $g(j) \leftarrow \mathcal{L}_0[i] + f(j)$ $\triangleright O(n)$

13:      **end if**

14: **end while**

Let us now compare classical Wagner's and quantum Wagner's algorithms. We first observe that the differences appear in the last two steps of the frame-

where

$$|\mathcal{L}_0| = \operatorname{surf}_q^M \left(\frac{k+l}{2^a + 1}, \frac{d}{2^a + 1} \right), \quad |\mathcal{L}_{2^a - 1}| = \operatorname{surf}_q^M \left(\frac{2(k+l)}{2^a + 1}, \frac{2d}{2^a + 1} \right),$$

$$(3.14)$$

$$|\mathcal{L}'| = \frac{\operatorname{surf}_M(q, \frac{k+l}{2^a+1}, \frac{d}{2^a+1})^{2^{a-1}}}{q^{\frac{(2^{a-1}-1)l}{a}}}, \quad |\mathcal{L}''| = \frac{\operatorname{surf}_M(q, \frac{k+l}{2^a+1}, \frac{d}{2^a+1})^{2^{a-1}-1} |\mathcal{L}_{2^a-1}|}{q^{\frac{(2^{a-1}-1)l}{a}}}$$

$$(3.15)$$

$$t_Q^{W\_sub} = O\left(|\mathcal{L}_0| \, (k+l)l + t_{\mathrm{mj}} + t_{\mathrm{fmj}} \right), \qquad (3.16)$$

$$t^{\mathrm{mj}} = O\left(\max\left(|\mathcal{L}_0|, \, |\mathcal{L}|' \right) l \right), \quad t_{\mathrm{fmj}} = O\left(\max\left(|\mathcal{L}_{2^a-1}|, \frac{|\mathcal{L}'| \, |\mathcal{L}''|}{q^{\frac{l}{a}}} \right) l \right),$$

$$(3.17)$$

$$t_g = O\big((n - k - l)(k + l)\big). \qquad (3.18)$$

and

$$p = \min\left(1, \frac{\operatorname{surf}_M(q, n-k-l, w-d)|\mathcal{L}_0|^{2^a-1}| \, |\mathcal{L}_{2^a-1}|}{\min\left(q^{n-k}, \, \operatorname{surf}_M(q, n, w) \right) q^{\frac{(2^a - a - 1)l}{a}}} \right). \qquad (3.19)$$

Proof. As in the case of the previously described classical algorithms, the running time of sampling a permutation is significantly smaller than the running time of other steps. The cost of one iteration thus comes down to:

$$t_Q^W = \frac{\kappa_2 + \kappa_3 + \kappa_4}{\sqrt{p}}$$

$$= O\left(\frac{n(n-k-l)^2 + t_Q^{W\_sub} + \sqrt{\operatorname{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})} \, t_g}{\sqrt{p}} \right),$$

where p is given by Lemma 3.2.1 and $t_q = O((n - k - l)(k + l))$ is the running time of $g : [\operatorname{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})] \to \mathbb{F}_q^{k+l}$ defined in Algorithm 8. By definition of Grover's search, the time complexity of searching over $g(\cdot)$ is given by $\sqrt{\operatorname{surf}_M(q, \frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})} \, t_g$, so we retrieve the cost of the last step. The cost of the second step is the cost of Gaussian elimination performed on $n - k - l$ columns of matrix $(n-k) \times n$, and it is given as $O((n-k-l)(k+l))$.

Finally, let us calculate the cost of the third step calling the quantum Wagner's sub-routine.

We first observe that using the MERGE-JOIN routine combined with the k-tree, at the level before the topmost one, the algorithm obtains the list of the expected size

$$|\mathcal{L}_1| = \frac{\operatorname{surf}_M(q, \frac{k+l}{2^{a+1}}, \frac{d}{2^{a+1}})^{2^{a-1}}}{q^{\frac{(2^{a-1}-1)l}{a}}}$$

in time

$$t_{\mathrm{mj}} = O\left(\operatorname{surf}_M(q, \frac{k+l}{2^a+1}, \frac{d}{2^a+1}), |\mathcal{L}_1|)l\right).$$

After merging the list with the output of $f(i)$ using FUNCTION-MERGE-JOIN routine, again in combination with k-tree algorithm, at the level before the topmost one, the algorithm obtains the list of expected size

$$|\mathcal{L}_2| = \frac{\operatorname{surf}_M(q, \frac{k+l}{2^{a+1}}, \frac{d}{2^{a+1}})^{2^{a-1}-1} \operatorname{surf}_M(q, \frac{2(k+l)}{2^{a+1}}, \frac{2d}{2^{a+1}})}{q^{\frac{(2^{a-1}-1)l}{a}}}$$

in time

$$t_{\mathrm{fmj}} = O\left(\left(\operatorname{surf}_q^M(\frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1}), |\mathcal{L}_2|)l\right).$$

Finally, merging the two lists gives a list $\mathcal{L}$ with the expected number of elements given as

$$|\mathcal{L}| = \frac{\operatorname{surf}_M(q, \frac{k+l}{2^{a+1}}, \frac{d}{2^{a+1}})^{2^a-1} surf_q^M(\frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1})}{q^{(2^a-1)\frac{l}{a}}},$$

in time $t_{\mathrm{mj}} + t_{\mathrm{fmj}}$. As the running time for creating the lists at the bottom level of the merging algorithms is given by $\operatorname{surf}_M(q^{\frac{k+l}{2^a+1}}, \frac{d}{2})$, we obtain the running time of the quantum Wagner's subroutine, and the running time of the third step, given by the expression 3.14. From Lemma 3.2.1 and 3.18, we finally derive the expression 3.19 that calculates the probability of finding a solution in one iteration of the algorithm. By definition of amplitude amplification, the expected number of iterations before the solution is found using amplitude amplification is given by $O(1/\sqrt{p})$. We thus obtain the running time of QUANTUM WAGNER algorithm given by expression 3.13. □

As in the case of classical Wagner's algorithm, we can set parameters l, d, and a so that $\operatorname{surf}_M(q, \frac{k+l}{2^a+1}, \frac{d}{2^a+1})^{2^a-1} surf_q^M(\frac{2(k+l)}{2^a+1}, \frac{2d}{2^a+1}) \approx q^{(2^a-1)\frac{l}{a}}$. We thus minimize the running time of the third step, in which the algorithm solves the M-SDP, so we obtain close to optimal (if not the optimal) quantum Wagner's algorithm.

3.5   Asymptotic analysis

As already explained, we rely on the ISD framework to learn the expected complexity of the syndrome decoding problem. More precisely, we estimate the running time of an ISD algorithm attacking an instance of the A-SDP whose input size grows beyond bounds. We take the code length, n, to be the size of the input, and all other parameters of the problem, as well as the parameters of the algorithm, to be functions of n. Furthermore, we take all these parameters to be constant fractions of n. We thus obtain constant values $R, L, \omega, \delta \in [0, 1]$ given as:

$$R := \frac{k}{n}, \quad L := \frac{l}{n}, \quad \omega := \frac{w}{\Delta n}, \quad \delta := \frac{d}{\Delta n}$$

where $\Delta := \max_{x \in \mathbb{F}_q} \mathrm{wt}_M(x)$. We observe here that the values of R and ω determine the instances of A-SDP, while the choice of L and δ determine the efficiency of the algorithm used for analyzing the problem.

Since we expect the running time of an ISD algorithm, $t : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, to be an exponential function[7], we take $t(n, k, w) = q^{\tau(R,\omega)n}$. We are then primarily interested in the asymptotic value of the exponent $\tau(\cdot, \cdot)$, so we calculate:

$$\tau(R, \omega) := \lim_{n \to \infty} \frac{1}{n} \log_q t(n, k, w).$$

To simplify the explanations, in the rest of the text, we omit "exponent" and refer to the exponents of asymptotic values simply as "asymptotic values". To obtain the asymptotic running time, we rely on the following assumption.

Assumption 1. *Let q be a prime number, $n, r \in \mathbb{N}$, $\delta \in [0, 1]$, and $\mathrm{surf}_M :$ $\mathbb{N} \times \mathbb{R} \times \mathbb{R} \to \mathbb{N}$ be the sphere surface area in a vector space $\mathbb{F}_q^n$ endowed with an elementwise weight function $\mathrm{wt}_M(\cdot)$. The sphere surface area then satisfies:*

$$\lim_{n \to \infty} \frac{\log_q \mathrm{surf}_M(q, \frac{n}{r}, \frac{\delta n}{r})}{n} = \lim_{n \to \infty} \frac{\log_q \mathrm{surf}_M(q, n, \delta n)^{1/r}}{n} = \frac{s_M(q, \delta)}{r},$$

---

[7]This is a reasonable assumption given that all known algorithms attacking the syndrome decoding problem run in time exponential in the input size of its instances.

> *where $s_M(q,\delta) := \lim_{n\to\infty} \frac{\log_q \operatorname{surf}_M(q,n,\delta n)}{n}$.*

For the Hamming weight, it is not hard to see that the assumption holds. Namely, we first recall that, for the Hamming weight, the asymptotic value of the sphere surface area is given by:

$$s_H(q,\delta) := \lim_{n\to\infty} \frac{surf_q^H(n,\delta n)}{n} = H_q(\delta),$$

where $H : [0,1] \to \mathbb{R}$ denotes the q-ary entropy function that is defined as $H_q(\delta) := \binom{\delta n}{n}$. We then observe that, by Stirling's approximation, $\binom{\delta n/r}{n/r} \approx \sqrt[r]{\binom{\delta n}{n}}$, which implies the claim of Assumption 1.

Different ISD algorithms   As remarked in the previous sub-section, we can adjust the parameters of Wagner's algorithm to simulate any of the four ISD algorithms we described, and the same goes for quantum Wagner's algorithm. In either classical or quantum cases, we then have the following correspondences between the chosen values of L, δ, a and the four ISD algorithms:

- $L = 0$, $\delta = 0$, $a = 1 \Rightarrow$ Prange's algorithm;

- $L = 0$, $\delta \geq 0$, $a = 1 \Rightarrow$ Lee-Brickel's algorithm;

- $L \geq 0$, $\delta \geq 0$, $a = 1 \Rightarrow$ Stern's/Dumer's algorithm;

- $L \geq 0$, $\delta \geq 0$, $a \geq 1 \Rightarrow$ Wagner's algorithm.

The asymptotic value of the running time of these algorithms is then calculated as the asymptotic running time of classical and quantum Wagner's algorithms with the appropriate choice of values of L, δ, and a. We express it through the following two propositions.

Proposition 3.5.1 (Asymptotic running time of classical ISD algorithms). *For fixed $L, \delta, a \in \mathbb{N}$, and parameters $R, \omega \in [0, 1]$, the asymptotic values of the exponent of the running time of a classical ISD algorithm, $\tau_C : [0, 1] \times [0, 1] \to \mathbb{R}$, is given as:*

$$\tau_C(R, \omega) = \left(\frac{2^a + 1}{2^a} \sigma_1 - (2^a - 1)\frac{L}{a} \right) - (1 - R - \sigma_0) + \sigma_2, \quad (3.20)$$

where

$$\sigma_0 = s_M(q, \omega), \quad \sigma_1 = (R + L)\, s_M\left(q, \frac{\delta}{R + L} \right), \quad (3.21)$$

$$\sigma_2 = (1 - R - L)s_M\left(q, \frac{\omega - \delta}{1 - R - L} \right). \quad (3.22)$$

Proof. The proof follows from Proposition 3.3.4 and Assumption 1. We start the proof by defining the following values:

$$\sigma_0 := \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n, w)}{n},$$

$$\sigma_1 := \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, k + l, d)}{n},$$

$$\sigma_2 := \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n - k - l, w - d)}{n}.$$

Taking Assumption 1, and using the definition of the asymptotic sphere sur-

face area, we then calculate $\sigma_0, \sigma_1,$ and σ_2 as:

$$\sigma_0 = \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n, \omega n)}{n}$$
$$= s_M(q, \omega),$$
$$\sigma_1 = \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, \frac{k+l}{(k+l)/n}, \frac{d}{(k+l)/n})^{(k+l)/n}}{n}$$
$$= (R + L) \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n, \frac{\delta}{R+L}n)}{n}$$
$$= (R + L)s_M\left(q, \frac{\delta}{R+L}\right),$$
$$\sigma_2 = \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, \frac{n-k-l}{(n-k-l)/n}, \frac{w-d}{(n-k-l)/n})^{(n-k-l)/n}}{n}$$
$$= (1 - R - L) \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n, \frac{\omega-\delta}{1-R-L}n)}{n}$$
$$= (1 - R - L)s_M\left(q, \frac{\omega - \delta}{1 - R - L}\right).$$

From Proposition 3.3.4, we can calculate the asymptotic values of the size of the initial lists, $|\mathcal{L}_0|$, as well as the asymptotic value of the size of the final list, $\mathcal{L}$. We thus obtain:

$$\lim_{n \to \infty} \frac{\log_q |\mathcal{L}_0|}{n} = \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M\left(q, \frac{k+l}{2^a}, \frac{d}{2^a}\right)}{n} = \frac{\sigma_1}{2^a},$$
$$\lim_{n \to \infty} \frac{\log_q |\mathcal{L}|}{n} = \lim_{n \to \infty} \frac{\log_q(\operatorname{surf}_M(q, \frac{k+l}{2^a}, \frac{d}{2^a})^{2^a}/q^{(2^a-1)\frac{l}{a}})}{n} = \sigma_1 - (2^a - 1)\frac{L}{a}.$$

As we know that the running time of the sub-routine is calculated as:

$$t_C^{\mathrm{sub}}(n, k, w) = O\left(\left(|\mathcal{L}_0|\,(k + l) + \max\left(|\mathcal{L}_0|, |\mathcal{L}|\right)\right)l\right),$$

its asymptotic value is then calculated as:

$$
\begin{aligned}
\lim_{n\to\infty} \frac{\log_q t_C^{\text{sub}}(n,k,w)}{n} &= \lim_{n\to\infty} \frac{\log_q |\mathcal{L}_0|}{n} + \lim_{n\to\infty} \frac{\log_q \max\left(|\mathcal{L}_0|,\ |\mathcal{L}|\right)}{n} \\
&= \lim_{n\to\infty} \frac{\log_q \max\left(|\mathcal{L}_0|,\ |\mathcal{L}|\right)}{n} \\
&= \lim_{n\to\infty} \frac{\log_q |\mathcal{L}_0|}{n} + \lim_{n\to\infty} \frac{\log_q |\mathcal{L}|)}{n} \\
&= \frac{2^a + 1}{2^a}\sigma_1 - (2^a - 1)\frac{L}{a}.
\end{aligned}
$$

The running time of the algorithm is given by:

$$
t_C(n,k,w) = O\left(\frac{(n-k-l)^2 n^3 + t_C^{W-\text{sub}}(n,k,w) + |\mathcal{L}|\,(n-k-l)(k+l)n^2}{p} \right).
$$

and its asymptotic value is calculated as:

$$
\begin{aligned}
\tau_C(R,\omega) &= \lim_{n\to\infty} \frac{\log_q t_C(n,k,w)}{n} \\
&= \lim_{n\to\infty} \frac{\log_q t_C^{\text{sub}}(n,k,w)}{n} + \lim_{n\to\infty} \frac{\log_q |\mathcal{L}|}{n} + \lim_{n\to\infty} \frac{\log_q n}{n}^{\,0} - \lim_{n\to\infty} \frac{\log_q p}{n} \\
&= \lim_{n\to\infty} \frac{\max\left(|\mathcal{L}_0|,\ |\mathcal{L}|\right)}{n} + \lim_{n\to\infty} \frac{\log_q |\mathcal{L}|}{n} - \lim_{n\to\infty} \frac{\log_q p}{n} \\
&= \left(\frac{2^a + 1}{2^a}\sigma_1 - (2^a - 1)\frac{L}{a} \right) - \lim_{n\to\infty} \frac{\log_q p}{n}.
\end{aligned}
$$

By Proposition 3.3.4, the probability of finding a solution in one iteration of Wagner's algorithm, p, is given by:

$$
p = \min\left(1,\ \frac{\operatorname{surf}_M(q, n-k-l, w-d)\operatorname{surf}_M\left(q, \frac{k+l}{2^a}, \frac{d}{2^a}\right)^{2^a}}{\min\left(q^{n-k},\ \operatorname{surf}_M(q,n,w)\right)q^{\frac{(2^a-a-1)l}{a}}}\right).
$$

Its asymptotic value is then calculated as:

$$
\lim_{n \to \infty} \frac{\log_q p}{n} = \min \left(0, \quad \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n - k - l, w - d)}{n} \right.
$$

$$
+ \ 2^a \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, \frac{k+l}{2^a}, \frac{d}{2^a})}{n}
$$

$$
- \min \left(1 - R, \ \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n, w)}{n} \right)
$$

$$
\left. - (2^a - a - 1)\frac{L}{a} \right)
$$

$$
= \min \left(0, \ \sigma_1 + \sigma_2 - \min(1 - R, \ \sigma_0) - (2^a - a - 1)\frac{L}{a} \right),
$$

$$
= \min(1 - R, \ \sigma_0) + (2^a - a - 1)\frac{L}{a} - \sigma_1 - \sigma_2,
$$

$$
= (1 - R - \sigma_0) - (\sigma_1 - (2^a - a - 1)\frac{L}{a}) - \sigma_2.
$$

Finally, we calculate the asymptotic value of the running time of classical algorithms using the following expression:

$$
\tau_C(R, \omega) = \left(\frac{2^a + 1}{2^a}\sigma_1 - (2^a - 1)\frac{L}{a} \right) - (1 - R - \sigma_0) + \sigma_2.
$$

This concludes the proof. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \square$

Proposition 3.5.2 (Asymptotic running time of quantum ISD algorithms). *For fixed* $L, \delta, a \in \mathbb{N}$, *and parameters* $R, \omega \in [0, 1]$, *the asymptotic values of the exponent of the running time of a quantum* ISD *algorithm,* $\tau_Q : [0, 1] \times [0, 1] \to \mathbb{R}$, *is given as:*

$$\tau_Q(R, \omega) = \max \left(\frac{2\sigma_1}{2^a + 1}, \ \sigma_1 - (2^a - 1)\frac{L}{a} \right) + \frac{\sigma_1}{2^a + 1} - \frac{\rho}{2}, \quad (3.23)$$

where

$$\sigma_0 = s_q^M(\omega), \quad \sigma_1 = (R + L) \, s_q^M \left(\frac{\delta}{R + L} \right), \quad (3.24)$$

$$\sigma_2 = (1 - R - L) s_q^M \left(\frac{\omega - \delta}{1 - R - L} \right) \quad (3.25)$$

and

$$\rho = \min \left(0, \ \sigma_1 + \sigma_2 - \min \left(1 - R, \ \sigma_0 \right) - (2^a - a - 1)\frac{L}{a} \right). \quad (3.26)$$

Proof. The proof follows the same line of reasoning as in the proof of 3.5.1, so we focus on the difference between the two cases and omit the details.

As in proof of 3.5.1, we start by calculating $\sigma_0, \sigma_1, \sigma_2$:

$$\sigma_0 := \lim_{n \to \infty} \frac{\log_q \mathrm{surf}_M(q, n, w)}{n} = s_M(q, \omega),$$

$$\sigma_1 := \lim_{n \to \infty} \frac{\log_q \mathrm{surf}_M(q, k + l, d)}{n} = (R + L) s_M \left(q, \frac{\delta}{R + L} \right),$$

$$\sigma_2 := \lim_{n \to \infty} \frac{\log_q \mathrm{surf}_M(q, n - k - l, w - d)}{n} = (1 - R - L) s_M \left(q, \frac{\omega - \delta}{1 - R - L} \right).$$

From Proposition 3.4.1, we can calculate the asymptotic values of the size of the initial lists, $|\mathcal{L}_0|$ and $|\mathcal{L}_{2^a - 1}|$, as well as the asymptotic value of the size of

the lists at the level before the topmost one, $|\mathcal{L}|'$ and $|\mathcal{L}|''$. We then obtain:

$$\lim_{n\to\infty} \frac{\log_q |\mathcal{L}_0|}{n} = \frac{\sigma_1}{2^a + 1}, \qquad \lim_{n\to\infty} \frac{\log_q |\mathcal{L}_{2^a-1}|}{n} = \frac{2\sigma_1}{2^a + 1},$$

$$\lim_{n\to\infty} \frac{\log_q |\mathcal{L}'|}{n} = \frac{2^{a-1}}{2^a + 1}\sigma_1 - (2^{a-1} - 1)\frac{L}{a},$$

$$\lim_{n\to\infty} \frac{\log_q |\mathcal{L}''|}{n} = \frac{2^{a-1} + 1}{2^a + 1}\sigma_1 - (2^{a-1} - 1)\frac{L}{a}.$$

As we know that the running time of the sub-routine is calculated as:

$$t_Q^{\text{sub}}(n, k, w) = O\left(\left(|\mathcal{L}_0|\,(k+l) + \max\left(|\mathcal{L}_0|,\ |\mathcal{L}'|\right) + \max\left(|\mathcal{L}_{2^a-1}|,\ \frac{|\mathcal{L}'||\mathcal{L}''|}{q^{\frac{l}{a}}}\right)\right)l\right),$$

its asymptotic value is then calculated as:

$$\lim_{n\to\infty} \frac{\log_q t_Q^{\text{sub}}(n, k, w)}{n} = \lim_{n\to\infty} \frac{\log_q |\mathcal{L}_0| + \log_q |\mathcal{L}_{2^a-1}| + \log_q |\mathcal{L}'| + \log_q |\mathcal{L}''|}{n} - \frac{L}{a}$$

$$= \frac{2^{a-1} + 1}{2^a + 1}\sigma_1 - (2^{a-1} - 1)\frac{L}{a}.$$

The running time of the algorithm is given by:

$$t_Q(n, k, w) = O\left(\frac{(1 - k - l)^2 n^3 + t_Q^{\text{sub}} + \sqrt{|\mathcal{L}_0|}\,(1 - k - l)(k + l)n^2}{p}\right).$$

and its asymptotic value is calculated as:

$$\tau_Q(R, \omega) = \lim_{n\to\infty} \frac{\log_q t_Q^{\text{sub}}(n, k, w)}{n} + \frac{1}{2}\lim_{n\to\infty} \frac{\log_q |\mathcal{L}_0|}{n} - \lim_{n\to\infty} \frac{\log_q p}{n}$$

$$= \left(\frac{2^{a-1} + 1}{2^a + 1}\sigma_1 - (2^{a-1} - 1)\frac{L}{a}\right) - \lim_{n\to\infty} \frac{\log_q p}{n}.$$

By Proposition 3.4.1, the probability of finding a solution in one iteration of Wagner's algorithm, p, is given by:

$$p = \min\left(1, \frac{\text{surf}_M(q, n - k - l, w - d)|\mathcal{L}_0|^{2^a - 1}\,|\mathcal{L}_{2^a-1}|}{\min\left(q^{n-k},\ \text{surf}_M(q, n, w)\right)q^{\frac{(2^a - a - 1)l}{a}}}\right). \qquad (3.27)$$

Its asymptotic value is then calculated as:

$$
\lim_{n \to \infty} \frac{\log_q p}{n} = \min \Bigg(0, \quad \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n - k - l, w - d)}{n}
$$

$$
+ \; (2^a + 1) \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, \frac{k+l}{2^a+1}, \frac{d}{2^a+1})}{n}
$$

$$
- \min \Bigg(1 - R, \; \lim_{n \to \infty} \frac{\log_q \operatorname{surf}_M(q, n, w)}{n} \Bigg)
$$

$$
- \; (2^a - a - 1)\frac{L}{a} \Bigg)
$$

$$
= (1 - R - \sigma_0) - (\sigma_1 - (2^a - a - 1)\frac{L}{a}) - \sigma_2.
$$

Finally, we calculate the asymptotic value of the running time of classical algorithms using the following expression:

$$
\tau_Q(R, \omega) = \left(\frac{2^a + 2^{a-1} + 1}{2^a + 1}\sigma_1 - (2^{a-1} - 1)\frac{L}{a} \right) - (1 - R - \sigma_0) + \sigma_2.
$$

This concludes the proof. $\qquad \qquad \square$

3.6   Numerical results

We now compare the asymptotic running time of the above-described ISD algorithms attacking the syndrome decoding problem in the Hamming and the Lee weight, for alphabet sizes in the range from 3 to 643.[8]. To calculate the asymptotic running time, we rely on the method for calculating the surface area of a sphere in a metric space endowed with an arbitrary weight function described in Chapter 2. More precisely, we use the convex optimization approach derived by generalizing the method presented in [Ast84]. For each algorithm, we then first calculate the optimal values of parameters that yield the shortest running times and we take the obtained shortest running time as the measurement of the algorithm's efficiency. The code used for obtaining numerical results is implemented in C++, and the convex optimization tasks are carried out using MOSEK solver [ApS19], more precisely, its Fusion API for C++. The code is available at the GitHub repository setinski.

Different ISD algorithms

In Figure 3.1 and Figure 3.2, we present the asymptotic running time of three classical ISD algorithms and one hybrid classical-quantum algorithm solving instances of A-SDP over the Hamming and the Lee weights, respectively. The three classical algorithms are Prange's, Stern's/Dumer's, Wagner's algorithm, and the hybrid classical-quantum algorithm we refer to as quantum Wagner's algorithm.

Let us now recall that we take the running time of an ISD algorithm, $t : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, to be a function of the code length, $n \in \mathbb{N}$, the code dimension, $k \in \mathbb{N}$, and the weight, $w \in \mathbb{N}$. In the asymptotic regime, we define the coefficient of the asymptotic running time of an algorithm, $\tau : [0,1] \times [0,1] \to \mathbb{R}$, given as:

$$\tau(R, \omega) := \lim_{n \to \infty} \frac{\log_2 t(n, k, w)}{n},$$

---

[8]Given that the number of constraints in the convex optimization problem grows linearly with the alphabet size, the computation becomes resource demanding for higher values of q. For $q = 643$, for example, the computation on the personal computer takes approximately one day with a moderate PC configuration.

where $R \in [0, 1]$ and $\omega \in [0, 1]$ represent the code rate and the normalized weight, respectively. The following figures show the comparison of the asymptotic running time of the four algorithms when solving the A-SDP for moderate code rates $R \approx 0.5$ and $\omega \in [0, 1]$.
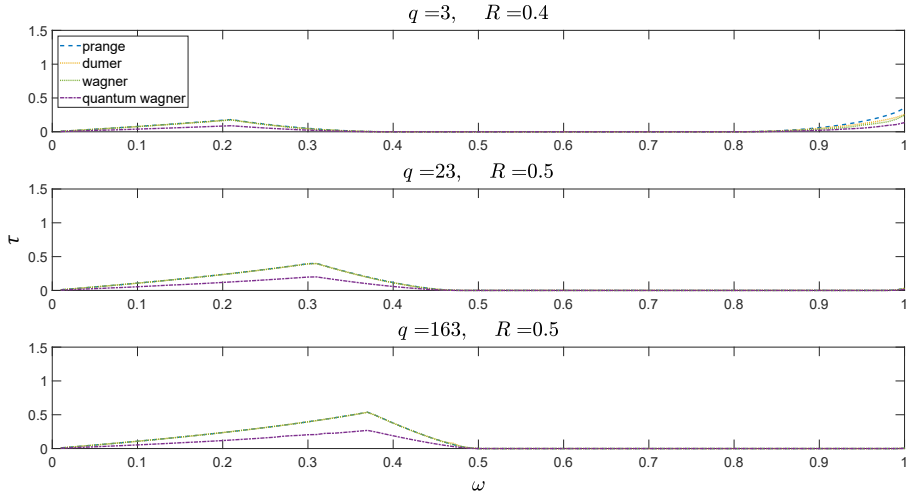
Figure 3.1: Complexity of four ISD algorithms when solving A-SDP over the Hamming weight

We notice here that the reducing the running time for solving A-SDP, obtained by introducing more advanced ISD algorithms, namely, Stern's/Dumer's and Wagner's algorithms, can mainly be observed for the higher values of ω and higher alphabet sizes. An almost quadratic reduction of running time of the hybrid classical-quantum algorithms, in comparison to all three classical algorithms, however, is noticeable for all parameter ranges and all alphabet sizes. In addition to that, we notice that, in the general case, the asymptotic running time of ISD algorithms solving the hardest instances of A-SDP over the Lee weight (observed as the local peaks in Figure 3.2) is longer than the asymptotic running time of algorithms solving the hardest instances of the problem in the Hamming weight case (observed as the local peaks in Figure 3.1). The numerical results thus indicate that, for the fixed alphabet size, the hardest A-SDP instances over the Lee weight are harder than the hardest A-SDP instances over the Hamming weight. We verify this claim through the numerical results on the hardest instances of the problem presented in the following subsection.
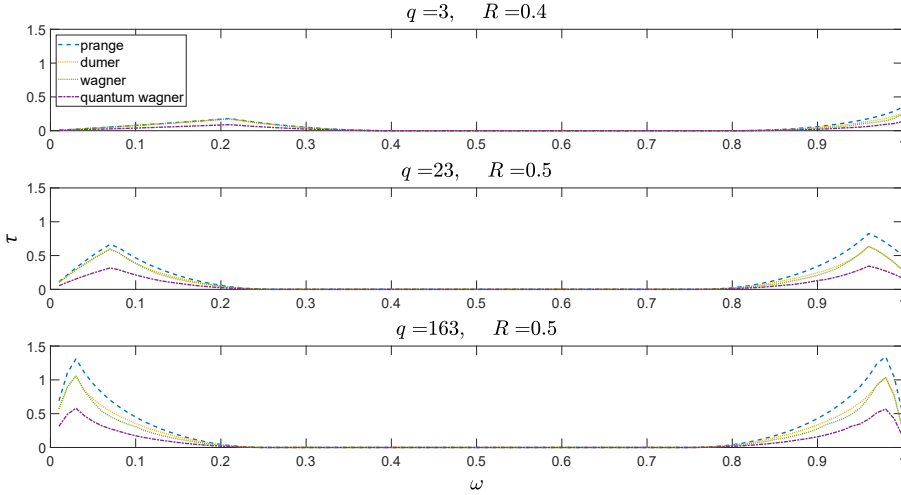
Figure 3.2: Complexity of four ISD algorithms when solving A-SDP over the Lee weight

Hamming weight case   We observe that the choice of alphabet size $q = 3$ is rather particular. Namely, for a fixed code rate, R, and the fixed alphabet size $q = 3$, there are two local maxima of the curve presenting the asymptotic complexity of the problem: one maximum is reached in what we call the *low-weight regime*, for which $\omega < 0.5$, and the other one in what we refer to as the *high-weight regime*, for which $\omega \geq 0.5$. We denote the two maxima by ω_-^*, and ω_+^*, respectively. We then observe that, for $q = 3$, the local maximum in the high-weight regime is slightly higher than the one in the low-weight regime. For a fixed R and given $q = 3$, we thus associate ω_+^* with the hardest instance of the problem. For $q > 3$, on the other hand, the numerical results show that the hardest instances of A-SDP over the Hamming weight can be found only in the low-weight regime. For a given code rate, R, and alphabet sizes $q > 3$, we then associate the corresponding values of ω_-^* to the hardest instances of the problem over the Hamming weight.

Lee weight case   In contrast to the Hamming weight case where, for a fixed code rate, R, the number of local maxima depends on the choice of q, in the case of the Lee weight, the number of local maxima is equal to two for all prime number alphabet sizes being analyzed. We can then say that the A-SDP over the Lee weight, in some sense, exhibits a more regular behavior

than the A-SDP over the Hamming weight. Again, we denote the two local maxima by ω_-^* and ω_+^*, respectively. We then observe that, for a fixed code rate, R, and a fixed alphabet size q, the second local maximum, obtained in the high-weight regime, is always higher. For the given q and R, the hardest instance of the problem is then associated with ω_+^*.

Hardest A-SDP instances

For a fixed alphabet size, q, and a fixed weight function, which is in our case either the Hamming weight, $\mathrm{wt}_H(\cdot)$, or the Lee weight, $\mathrm{wt}_L(\cdot)$, we now search for the code rate, R, and the normalized weight, ω, that yield the hardest instance of A-SDP. Namely, we search for the global maximum of the asymptotic running time given as a function of code rate and the normalized weight. The values of R and ω for which the global maximum is reached then yield the hardest A-SDP instance for the given alphabet size and weight function. To limit the range of values we search over and thus speed up the computation, we rely on the above-presented observations about the local maxima of the asymptotic running time. More details on the optimization procedure can be found in our paper [CDE21].

The obtained numerical results are presented in Table 3.1 and Table 3.2. For the given q, R, and ω, we then express the relative hardness of an instance as two coefficients, $\tau(\cdot, \cdot)$, and $\hat{\tau}(\cdot, \cdot)$, defined as follows:

$$\tau(R, \omega) := \lim_{n \to \infty} \frac{\log_2 t^W(n, k, w)}{n}, \quad \hat{\tau}(R, \omega) := \lim_{n \to \infty} \frac{\log_q t^W(n, k, w)}{n},$$

where $t^W(n, k, q)$ is the running time of (quantum) Wagner's algorithm solving an instance of A-SDP for the given code length, n, code dimension, k, and weight, w.[9] The asymptotic running time is a function of the code rate, R, and normalized weight, ω, obtained for the optimal choice of parameters of Wagner's algorithm, i.e. for the values of a, l, and d that yield the shortest running time of the (quantum) Wagner algorithm.

---

[9]Wagner's algorithm is chosen as the most generalized and the most evolved version of the presented ISD algorithms.

Hamming weight, $\mathrm{wt}_H(\cdot)$							
q	Wagner's algorithm				quantum Wagner's algorithm			
	R	ω	τ	$\hat{\tau}$	R	ω	τ	$\hat{\tau}$
3	0.370	1.000	0.269	0.170	0.369	1.000	0.148	0.093
13	0.456	0.311	0.356	0.096	0.453	0.314	0.180	0.049
43	0.459	0.368	0.459	0.085	0.459	0.368	0.230	0.042
163	0.463	0.405	0.541	0.074	0.463	0.405	0.271	0.037
643	0.468	0.427	0.602	0.065	0.475	0.420	0.316	0.034

Table 3.1: Hardest instances of A-SDP over the Hamming weight

Lee weight, $\mathrm{wt}_L(\cdot)$							
q	Wagner's algorithm				quantum Wagner's algorithm			
	R	ω	τ	$\hat{\tau}$	R	ω	τ	$\hat{\tau}$
3	0.370	1.000	0.269	0.170	0.369	1.000	0.148	0.093
13	0.475	0.955	0.522	0.141	0.501	0.962	0.283	0.076
43	0.459	0.955	0.794	0.146	0.467	0.957	0.429	0.079
163	0.445	0.968	1.117	0.152	0.462	0.971	0.607	0.083
643	0.437	0.980	1.455	0.156	0.455	0.982	0.794	0.085

Table 3.2: Hardest instances of A-SDP over the Lee weight

The reason we introduce two coefficients for expressing the asymptotic running time, namely, $\tau(\cdot, \cdot)$ and $\hat{\tau}(\cdot, \cdot)$, is the following. The common way of expressing the asymptotic running time of an algorithm is in the form $t(n, k, w) = 2^{\tau(R,\omega)n}$, where $n \to \infty$, and $\tau(\cdot, \cdot)$ is determined by the algorithm. Nevertheless, it appears that this representation does not reflect accurately the difficulty of the syndrome decoding problem in the generalized setting we focus on. We thus introduce yet another coefficient of the asymptotic running time, $\hat{\tau}(\cdot, \cdot)$, obtained when $t(\cdot, \cdot, \cdot)$ is given as $t(n, k, w) = q^{\hat{\tau}(R,\omega)n}$ and $n \to \infty$. This coefficient seems to reflect more accurately the difficulty of the problem as it is consistent with the previous observations regarding the syndrome decoding problem over the Hamming weight for alphabet sizes $q > 2$ (see, for example, [Pet10]).

Code rate and normalized weight   We observe that, both in the Hamming weight case and in the Lee weight case, the code rate of the hardest instances of the problem is in the range $[0.35, 0.5]$. This was already provisioned by the plots of previously obtained numerical results. We also observe that, in the Hamming weight setting and for $q > 3$, for the value of R yielding the hardest instances of the problem, the normalized weight attains the so-called *Gilbert-Varshamov bound*, defined in Chapter 1. The hardest instances of the problem over the Hamming weight for $q = 3$ or over the Lee weight for any observed q, on the other hand, are obtained in the high-weight regime. To the best of our knowledge, this parameter regime does not correspond to any known bound and it is commonly omitted in the numerical analysis though it indeed yields the hardest instances of the problem. We thus highlight the importance of taking the high-weight regime into account when analyzing the generalized syndrome decoding problem.

Hamming weight and Lee weight   The first observation to be made is that both coefficients (namely, $\tau(\cdot, \cdot)$ and $\hat{\tau}(\cdot, \cdot)$), in the classical as well as in the quantum settings, are higher in the Lee weight case. Again, this was already provisioned by the plots of previously obtained numerical results. This further suggests that the hardest A-SDP instances over the Lee weight are harder than the hardest A-SDP instances over the Hamming weight. Let us then observe only the values of $\tau(\cdot, \cdot)$ in the two cases. We then see that $\tau(\cdot, \cdot)$ increases with the increase of q in all the cases. This gives the impression that the complexity of the problem increases with the rise of the alphabet size. However, in the previous studies of this problem for the Hamming weight, it was shown that, in fact, this is not the case. Let us then observe $\hat{\tau}(\cdot, \cdot)$. In the Hamming weight case, both for the classical and the quantum setting, there is a sudden drop in $\hat{\tau}(\cdot, \cdot)$ when we switch from $q = 3$ and $q = 13$, and the coefficient continues to decrease as the alphabet size increases. In the Lee weight case, on the other hand, the drop is less dominant, and, in fact, the coefficient starts to increase again for alphabet sizes between $q = 43$ and $q = 163$. The open question is then what would be the limit case? Can we then hope that for some really high alphabet sizes, the complexity will get higher than for $q = 3$?

Classical and quantum Wagner's algorithm   If we now compare $\tau(\cdot, \cdot)$ (equivalently, $\hat{\tau}(\cdot, \cdot)$) obtained in the classical and quantum settings for the

Hamming weight, we can see that the coefficients in the classical setting are twice as big as those in the quantum setting. This quadratic speed-up was exactly what was expected from Grover's search and amplitude amplification introduced in the hybrid classical-quantum approach. In the Lee weight case, on the other hand, we observe that the speed-up is not exactly quadratic. Namely, the coefficients in the classical setting are not exactly twice as big as in the classical case. This small gap leaves a place for future improvements.

STERN DIGITAL SIGNATURE SCHEME

In this chapter, we study the adaptation of the so-called *Stern identification scheme*, and its digital signature version, to the generalized syndrome decoding problem. We thus propose a new scheme construction and verify its security. During the course of the chapter, we also propose an attack on a particular version of the scheme construction, and we then suggest techniques for mitigating this attack. This work will appear in what is currently a submitted preprint [CE23].

4.1 Stern Identification Scheme

Along with the *McEliece encryption protocol* [McE78], what we call the *Stern identification scheme* [Ste93] is one of the first examples of cryptographic protocols based on the syndrome decoding problem. In its original version, the scheme is based on the binary syndrome decoding problem in the Hamming weight. This section shows how the original scheme can be adapted to the generalized syndrome decoding setting, introduced in Chapter 2. We remark that the adapted version of the protocol does not preserve the zero-knowledge property of the original scheme. To overcome this shortcoming, we adapt the scheme again by replacing an instance of the generalized syndrome decoding problem with an instance of the permuted kernel problem, presented in Chapter 2, knowing that A-SDP is polynomial time reducible to A-PKP. We then show the basic construction of a scheme based on the A-PKP, the proof of its security, and some techniques for improving the scheme's efficiency.

4.1.1 Basic scheme construction

In the basic construction of the Stern identification protocols, the key generator samples an instance of an A-SDP problem and takes the error vector, $\mathbf{e}$, to be the secret key and the parity check matrix, $\mathbf{H}$, along with the syndrome, $\mathbf{s}$, to be the public key. We now present the key generation part of the protocol.

Stern's protocol - basic construction

Prover   **Verifier**

$\dots\dots\dots\dots\dots\dots\dots\dots$ *Key Generation* $\dots\dots\dots\dots\dots\dots\dots\dots$

$\mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k)\times n}, \mathbf{e} \xleftarrow{\$} \mathcal{S}_M(q,n,w),$

$\mathbf{s} \leftarrow \mathbf{He},$

$\mathsf{sk} \leftarrow \mathbf{e}, \mathsf{pk} \leftarrow (\mathbf{H}, \mathbf{s}, w)$

$$\xrightarrow{\quad\mathsf{pk}\quad} \quad \dots$$
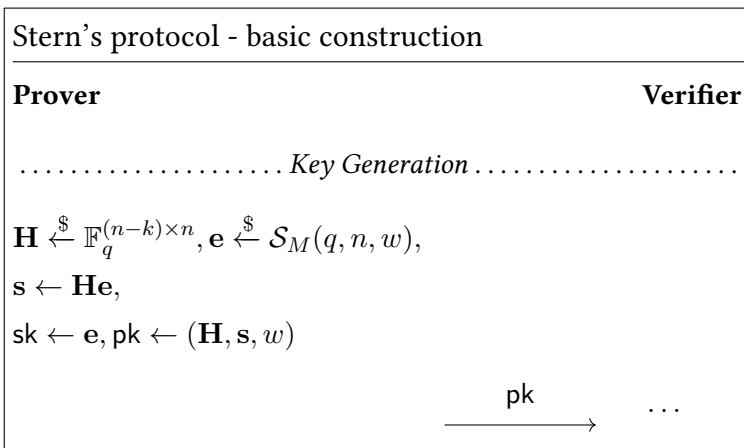
In the interactive part of the protocol, the prover aims to prove it has the secret key, i.e. the error vector $\mathbf{e}$, preferably without revealing any information on it. By doing so, the prover effectively allows the verifier to check the prover's identity, ideally, without revealing any additional information. When the secret key is binary, i.e. $\mathbf{e} \in \mathbb{F}_2^n$, the two basic strategies for disguising it are to permute the error vector and to scramble it using some other, randomly generated vector in $\mathbb{F}_2^n$. In the more general case when $\mathbf{e} \in \mathbb{F}_q^n$ for $q \neq 2$, however, the permutation does not allow for a complete randomization vector $\mathbf{e}$. We will touch a bit more upon this point in the part of the zero-knowledge property of the scheme. Here we present the original Stern's protocol adapted to the generalized syndrome decoding setting, which guarantees that the prover will be correctly recognized with probability 1 but does not promise that no information on $\mathbf{e} \in \mathbb{F}_q^n$ will be leaked in the process if $q \neq 2$.

The interactive part of the protocol thus starts via sampling a random permutation, $\pi \in \Pi[n]$, and a random vector, $\mathbf{y} \in \mathbb{F}_q^n$, which are then used for disguising the secret key. The prover commits to three values, $\mathbf{c}_0$, $\mathbf{c}_1$, $\mathbf{c}_2 \in \{0,1\}^d, d \in \mathbb{N}$, using the hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^d$.[1] The three commitments are then sent to the verifier that stores these and responds by a random challenge $c \in \{0,1,2\}$. Based on the challenge obtained from the verifier, the prover forms the opening, $\mathbf{o} \in \{0,1\}^*$, and sends it to the verifier. In the case of the honest prover, the opening contains sufficient information for checking two out of three commitments, i.e. the two commitments that do not correspond to the challenge, $\{\mathbf{c}_i\}_{i \in [3] \backslash c}$. These two commitments along with the opening then should enable the verifier to check the prover's identity. The interactive part of the protocol is given in the scheme that follows.

---

[1]We assume here that the hash function is unkeyed but that it takes the randomness $\mathbf{r}_i$ as its input.

Stern's protocol - basic construction

Prover   **Verifier**

. . . . *Interaction* . . . .

$\pi \xleftarrow{\$} \Pi[n]$, $\mathbf{y} \xleftarrow{\$} \mathbb{F}_q^n$, $\mathbf{t} \leftarrow \mathbf{Hy}$

$\mathbf{r}_0 \xleftarrow{\$} \{0,1\}^m$, $\mathbf{c}_0 \leftarrow \mathcal{H}(\pi, \mathbf{t}, \mathbf{r}_0)$,

$\mathbf{r}_1 \xleftarrow{\$} \{0,1\}^m$, $\mathbf{c}_1 \leftarrow \mathcal{H}(\pi(\mathbf{y}), \mathbf{r}_1)$,

$\mathbf{r}_2 \xleftarrow{\$} \{0,1\}^m$, $\mathbf{c}_2 \leftarrow \mathcal{H}(\pi(\mathbf{y} + \mathbf{e}), \mathbf{r}_2)$

$$\xrightarrow{\{\mathbf{c}_i\}_{i \in [3]}}$$

$c \xleftarrow{\$} [3]$

$$\xleftarrow{c}$$

$\mathbf{o} \leftarrow \begin{cases} (\pi(\mathbf{y}), \pi(\mathbf{e}), \mathbf{r}_1, \mathbf{r}_2), & \text{if } c = 0 \\ (\pi, \mathbf{y} + \mathbf{e}, \mathbf{r}_0, \mathbf{r}_2), & \text{if } c = 1 \\ (\pi, \mathbf{y}, \mathbf{r}_0, \mathbf{r}_1), & \text{if } c = 2 \end{cases}$.

$$\xrightarrow{\mathbf{o}}$$ . . .
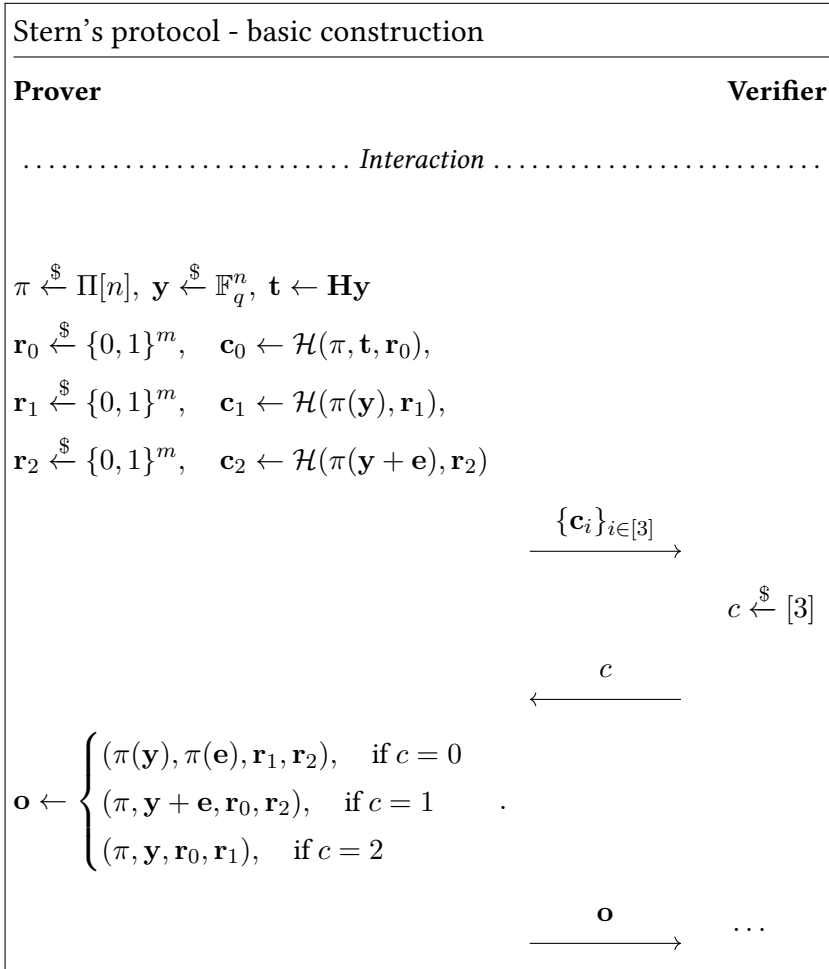
In the verification part of the protocol, the verifier checks the consistency between the two commitments and the corresponding opening provided by the prover in the interactive part of the protocol. If these are consistent, the verifier answers positively i.e. returns 1 and thus verifies the prover's identity. If the verifier recognizes the inconsistency, it rejects the proof of identity and answers negatively, i.e. returns 0. We present the verification part of the protocol in the scheme that follows.

Stern's protocol - basic construction

Prover	**Verifier**

. . . . *Verification* . . . .

$\quad$ if $c = 0$:

$\qquad \tilde{\mathbf{c}}_1 \leftarrow \mathcal{H}(\mathbf{o}[0], \mathbf{o}[2])$

$\qquad \tilde{\mathbf{c}}_2 \leftarrow \mathcal{H}(\mathbf{o}[0] + \mathbf{o}[1], \mathbf{o}[3])$

$\qquad$ if $\mathbf{c}_1 = \tilde{\mathbf{c}}_1, \mathbf{c}_2 = \tilde{\mathbf{c}}_2 \wedge \mathrm{wt}_M(\mathbf{o}[1]) = w$:

$\qquad \quad a \leftarrow 1$

$\qquad$ else : $a \leftarrow 0$

$\qquad$ end if

$\quad$ else if $c = 1$:

$\qquad \tilde{\mathbf{c}}_0 \leftarrow \mathcal{H}(\mathbf{o}[0], \mathbf{H}\mathbf{o}[1] - \mathbf{s}, \mathbf{o}[2])$

$\qquad \tilde{\mathbf{c}}_2 \leftarrow \mathcal{H}(\mathbf{o}[0](\mathbf{o}[1]), \mathbf{o}[3])$

$\qquad$ if $\mathbf{c}_0 = \tilde{\mathbf{c}}_0 \wedge \mathbf{c}_2 = \tilde{\mathbf{c}}_2$:

$\qquad \quad a \leftarrow 1$

$\qquad$ else : $a \leftarrow 0$

$\qquad$ end if

$\quad$ else if $c = 2$:

$\qquad \tilde{\mathbf{c}}_0 \leftarrow \mathcal{H}(\mathbf{o}[0], \mathbf{H}\mathbf{o}[1], \mathbf{o}[2])$

$\qquad \tilde{\mathbf{c}}_1 \leftarrow \mathcal{H}(\mathbf{o}[0](\mathbf{o}[1]), \mathbf{o}[3])$

$\qquad$ if $\mathbf{c}_0 = \tilde{\mathbf{c}}_0 \wedge \mathbf{c}_1 = \tilde{\mathbf{c}}_1$:

$\qquad \quad a \leftarrow 1$

$\qquad$ else : $a \leftarrow 0$

$\qquad$ end if

$\quad$ end if

□ $\qquad \xleftarrow{\quad a \quad}$

We will now show that the above-presented protocol is computational *complete* and *sound*. Namely, the protocol allows an honest prover to respond

with an opening that is consistent with the commitments for each verifier's challenge. In the same time, the protocol does not allow a cheating prover to respond with an opening that is consistent with all the challenges.

Completness To show the protocol is complete, we assume the prover is honest and then consider the consistency between the commitments and the opening for different challenges. Namely, we observe the following three cases:

- $c = 0$: The verifies checks the consistency between:

$$\mathbf{c}_1 = \mathcal{H}(\pi(\mathbf{y}), \mathbf{r}_1), \ \mathbf{c}_2 = \mathcal{H}(\pi(\mathbf{y} + \mathbf{e}), \mathbf{r}_2),$$

and the opening $\mathbf{o} = (\pi(\mathbf{y}), \pi(\mathbf{e})), \mathbf{r}_1, \mathbf{r}_2)$. The verifier thus calculates:

$$\tilde{\mathbf{c}}_1 = \mathcal{H}(\mathbf{o}[0], \mathbf{o}[2]) = \mathcal{H}(\pi(\mathbf{y}), \mathbf{r}_1),$$
$$\tilde{\mathbf{c}}_2 = \mathcal{H}(\mathbf{o}[0] + \mathbf{o}[1], \mathbf{o}[3]) = \mathcal{H}(\pi(\mathbf{y} + \mathbf{e}), \mathbf{r}_2),$$

and confirms that indeed $\mathbf{c}_1 = \tilde{\mathbf{c}}_1$, $\mathbf{c}_2 = \tilde{\mathbf{c}}_2$, and $\mathrm{wt}_M(\pi(\mathbf{e})) = w$.

- $c = 1$: The verifies checks the consistency between:

$$\mathbf{c}_0 = \mathcal{H}(\pi, \mathbf{t}, \mathbf{r}_0), \ \mathbf{c}_2 = \mathcal{H}(\pi(\mathbf{y} + \mathbf{e}), \mathbf{r}_2),$$

and the opening $\mathbf{o} = (\pi, \mathbf{y} + \mathbf{e}), \mathbf{r}_0, \mathbf{r}_2)$. The verifier thus calculates:

$$\tilde{\mathbf{c}}_0 = \mathcal{H}(\mathbf{o}[0], \mathbf{H}\mathbf{o}[1] + \mathbf{s}, \mathbf{o}[2]) = \mathcal{H}(\pi, \mathbf{t}, \mathbf{r}_0),$$
$$\tilde{\mathbf{c}}_2 = \mathcal{H}(\mathbf{o}[0](\mathbf{o}[1]), \mathbf{o}[3]) = \mathcal{H}(\pi(\mathbf{y} + \mathbf{e}), \mathbf{r}_2),$$

and confirms that indeed $\mathbf{c}_0 = \tilde{\mathbf{c}}_0$ and $\mathbf{c}_2 = \tilde{\mathbf{c}}_2$.

- $c = 2$: The verifies checks the consistency between:

$$\mathbf{c}_0 = \mathcal{H}(\pi, \mathbf{t}, \mathbf{r}_0), \ \mathbf{c}_1 = \mathcal{H}(\pi(\mathbf{y}), \mathbf{r}_1),$$

and the opening $\mathbf{o} = (\pi, \mathbf{y}, \mathbf{r}_0, \mathbf{r}_1)$. The verifier thus calculates:

$$\tilde{\mathbf{c}}_0 = \mathcal{H}(\mathbf{o}[0], \mathbf{H}\mathbf{o}[1], \mathbf{o}[2]) = \mathcal{H}(\pi, \mathbf{t}, \mathbf{r}_0),$$
$$\tilde{\mathbf{c}}_1 = \mathcal{H}(\mathbf{o}[0](\mathbf{o}[1]), \mathbf{o}[3]) = \mathcal{H}(\pi(\mathbf{y}), \mathbf{r}_1),$$

and confirms that indeed $\mathbf{c}_0 = \tilde{\mathbf{c}}_0$ and $\mathbf{c}_2 = \tilde{\mathbf{c}}_2$.

Therefore, in each of these cases when the prover is honest, it responds with an opening consistent with the two required commitments, and the verifier accepts the proof of identity.

Soundness   To show the protocol is sound, we argue that no cheating strategy allows a dishonest prover to answer all three challenges consistently with overhelmingly high probability (unless A-SDP is computationally easy to solve or there exists a non-negligible probability of finding a collision in the hash function used for committing). We summarize it in the following proposition.

> **Proposition 4.1.1** (Soundness of Stern's identification protocol). *The Stern identification protocol based on A-SDP is computationally sound with a soundness error of* $2/3$.

Proof. Let us assume that for each $c \in \{0, 1, 2\}$, a prover can provide the three commitments, $\{c_i^c\}_{i \in [3]}$, and the opening, o^c, that are consistent among each other and that pass the verifier's check. For each $i \in \{0, 1, 2\}$, the commitments then satisfy:

$$c_i^0 = c_i^1 = c_i^2 = c_i,$$

and for each $c \in \{0, 1, 2\}$, the opening o^c is consistent with the corresponding two commitments, $\{c_i\}_{i \in [3] \setminus c}$.

For a dishonest prover, this happens in two cases. The first case is when the prover manages to find a collision in the hash function used for committing to $\{c_i\}_{i \in [3]}$. The found collision then allows one to find an *impersonation strategy* which gives commitments consistent with the opening even without the prover's knowledge of the secret key. The soundness of the scheme thus reduces the hardness of finding a collision in the hash function $\mathcal{H}(\cdot)$.

In the second case, the cheating prover indeed commits to the same values for each challenge and these values are consistent with the openings. In that case, we have that:

$$c_0 = \mathcal{H}(\pi, \mathbf{t}, \mathbf{r}_0), \ c_1 = \mathcal{H}(\pi(\mathbf{y}), \mathbf{r}_1), \ c_2 = \mathcal{H}(\pi(\mathbf{y} + \tilde{\mathbf{e}}), \mathbf{r}_2),$$

where $\pi \xleftarrow{\$} \Pi[n]$, $\mathbf{y} \xleftarrow{\$} \mathbb{F}_q^n$, and $\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2 \xleftarrow{\$} \{0, 1\}^m$ are sampled by the prover, $\mathbf{t} \leftarrow \mathbf{H}\mathbf{y}$ is calculated by the prover, and $\tilde{\mathbf{e}} \in \mathbb{F}_q^n$ is chosen so that the commitments and the opening pass the verifier consistency test at the end

of the protocol. The chosen $\tilde{\mathbf{e}}$ thus needs to satisfy $\mathbf{H}\tilde{\mathbf{e}} + \mathbf{H}\mathbf{y} - \mathbf{s} = \mathbf{H}\mathbf{y}$, checked when $c = 1$, as well as $\mathrm{wt}_M(\pi(\tilde{\mathbf{e}})) = w$, checked when $c = 0$. If both satisfied, $\tilde{\mathbf{e}}$ also satisfies $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ and $\mathrm{wt}_M(\tilde{\mathbf{e}}) = w$, which implies $\tilde{\mathbf{e}}$ is a solution of the A-SDP instance. This further implies that the cheating prover is able to find a solution to the generalized syndrome decoding problem. The soundness of the scheme thus reduces the hardness of finding a solution to A-SDP. This concludes the proof.

$\square$

To show that the cheating prover could still cheat with the probability of $2/3$ (assuming the challenges are sampled uniformly at random) even without the knowledge of the solution to the A-SDP instance, let us observe the following impersonation strategy. The prover makes a guess on the error vector by sampling it uniformly at random from the set $\mathcal{S}_q(n, w)$, i.e. it takes $\tilde{\mathbf{e}} \xleftarrow{\$} \mathcal{S}_q(n, w)$. The prover then commits to the following values:

$$\mathbf{c}_0 = \mathcal{H}(\pi, \mathbf{H}\mathbf{y}, \mathbf{r}_0), \ \ \mathbf{c}_1 = \mathcal{H}(\pi(\mathbf{y}), \mathbf{r}_1), \ \ \mathbf{c}_2 = \mathcal{H}(\pi(\mathbf{y} + \tilde{\mathbf{e}}), \mathbf{r}_2),$$

and gives the following opening:

$$\mathbf{o} \leftarrow \begin{cases} (\pi(\mathbf{y}), \pi(\tilde{\mathbf{e}}), \mathbf{r}_1, \mathbf{r}_2), & \text{if } c = 0 \\ (\pi, \mathbf{y} + \tilde{\mathbf{e}}, \mathbf{r}_0, \mathbf{r}_2), & \text{if } c = 1 \ . \\ (\pi, \mathbf{y}, \mathbf{r}_1, \mathbf{r}_2), & \text{if } c = 2 \end{cases}$$

This gives us the soundness error of $2/3$. We observe that the soundness error of $2/3$ is rather high as it implies that a cheating prover succeeds in providing the proper proof of identity on average $2/3$ of the time. To reduce the soundness error, we can repeat the protocol r times, where $r \in \mathbb{N}$, and thus obtain the so-called r-*fold parallel repetition* protocol. In this type of protocol, the overall proof is accepted if and only if in each parallel repetition, the proof is accepted by the verifier. Otherwise, the overall proof is rejected. This approach gives us the soundness error that can be arbitrarily small. More precisely, the soundness error is equal to the probability that a cheating prover succeeds in each parallel repetition, and it is equal to $(2/3)^r$.

Zero-knowledge The original Stern identification scheme, based on the binary syndrome decoding problem in the Hamming weight, is zero-knowledge.

Without going into formal proof, we can see this through an argument that neither the commitments nor the openings reveal any information about the secret key. The modified scheme we presented, on the other hand, is not zero-knowledge in the general case. To see that, let us observe a permutation of the error vector, $\pi(\mathbf{e})$. It is obtained as the opening for $c = 0$, and, as such, it reveals the distribution of the elements in the error vector, i.e. the number of zeros, ones, ..., values of $q - 1$ in $\mathbf{e}$. Therefore, the protocol is not zero-knowledge.

To circumvent this problem, we replace an instance of the generalized syndrome decoding problem, used in the key-generation part, with an instance of the permuted kernel problem and obtain the following version of the key-generation part.

Stern's protocol - PKP construction

Prover   **Verifier**

.... *Key Generation* ....

$\mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k) \times n}$, $\sigma \xleftarrow{\$} \Pi[n]$

$\mathbf{s} \leftarrow \mathbf{H}\sigma(\mathbf{v})$

$\mathsf{sk} \leftarrow \sigma$, $\mathsf{pk} \leftarrow (\mathbf{H}, \mathbf{s}, \mathbf{v})$

$$\xrightarrow{\quad\quad \mathsf{pk} \quad\quad} \quad \ldots$$
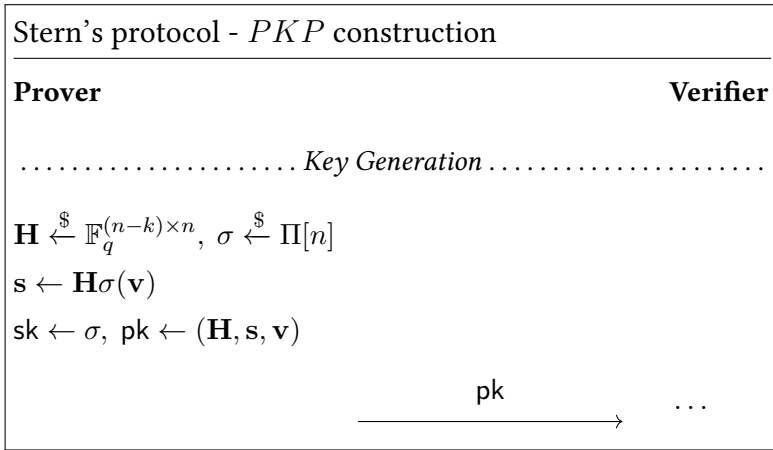
We recall here that the replacement of the A-SDP instance with and A-PKP instance is possible given the reduction of A-SDP to A-PKP presented in Chapter 2. The secret key thus becomes a permutation, $\sigma \in \Pi[n]$, and the public key comprises the parity check matrix, the syndrome, and the error vector $\mathbf{v} \in \mathcal{S}_M(q, n, w)$. The remaining parts of the scheme remain exactly the same as in the original version. We now show that the modified version of the scheme is *honest verifier zero-knowledge*, as suggested by Proposition 4.1.2.

Proposition 4.1.2 ($HZVK$ property of modified Stern's protocol)**.** *The Stern identification protocol based on the permuted kernel problem is computationally honest-verifier zero-knowledge.*

Stern's protocol - simulator of PKP construction

Simulator

$c \xleftarrow{\$} [3]$

...................Committing....................

$\tilde{\pi} \xleftarrow{\$} \Pi[n], \; \tilde{\mathbf{y}} \xleftarrow{\$} \mathbb{F}_q^n,$

$\tilde{\mathbf{r}}_0, \tilde{\mathbf{r}}_1, \tilde{\mathbf{r}}_2 \xleftarrow{\$} \{0,1\}^m$

if $c = 2$:

  $\tilde{\sigma}, \tilde{\mathbf{v}} \leftarrow$ solution to $\mathbf{H}\tilde{\sigma}(\tilde{\mathbf{v}}) = \mathbf{s},$

else :

  $\tilde{\sigma} \xleftarrow{\$} \Pi[n],$

end if

$\tilde{\mathbf{t}} \leftarrow \begin{cases} \mathbf{H}(\tilde{\mathbf{y}} + \tilde{\sigma}(\mathbf{v})) - \mathbf{s}, & \text{if } c = 1 \\ \mathbf{H}\tilde{\mathbf{y}}, & \text{otherwise} \end{cases}$

$\mathbf{c}_0 \leftarrow \mathcal{H}(\tilde{\pi}, \tilde{\mathbf{t}}, \tilde{\mathbf{r}}_0),$

$\mathbf{c}_1 \leftarrow \mathcal{H}(\tilde{\pi}(\tilde{\mathbf{y}}), \tilde{\mathbf{r}}_1),$

$\mathbf{c}_2 \leftarrow \begin{cases} \mathcal{H}(\tilde{\pi}(\tilde{\mathbf{y}} + \tilde{\sigma}(\tilde{\mathbf{v}})), \tilde{\mathbf{r}}_2), & \text{if } c = 2, \\ \mathcal{H}(\tilde{\pi}(\tilde{\mathbf{y}} + \tilde{\sigma}(\mathbf{v})), \tilde{\mathbf{r}}_2), & \text{otherwise} \end{cases}$

.... Opening ....

$\mathbf{o} \leftarrow \begin{cases} (\tilde{\pi}(\tilde{\mathbf{y}}), \tilde{\pi}(\tilde{\sigma}(\mathbf{v})), \tilde{\mathbf{r}}_1, \tilde{\mathbf{r}}_2), & \text{if } c = 0 \\ (\tilde{\pi}, \tilde{\mathbf{y}} + \tilde{\sigma}(\mathbf{v}), \tilde{\mathbf{r}}_0, \tilde{\mathbf{r}}_2), & \text{if } c = 1 \\ (\tilde{\pi}, \tilde{\mathbf{y}}, \tilde{\mathbf{r}}_1, \tilde{\mathbf{r}}_2), & \text{if } c = 2 \end{cases}$.

Let us observe a simulator of the interactive part of the scheme, which simulates the interaction between the prover and verifier for the case when the verifier accepts the proof of identity given by the prover. To prove the scheme is an honest verifier zero-knowledge, we need to show that the distribution of the transcript of the original interaction between the prover and the hon-

est verifier is indistinguishable from the distribution of the transcript of the simulated interaction.

Let us then denote by $\text{trans}^{\text{orig}}$ the transcript of the original interaction that consists of the commitments, $\{\mathbf{c}_i^{\text{orig}}\}_{i \in [3]}$ given by the prover, a challenge, c^{orig}, sampled by the honest verifier, and the opening, $\mathbf{o}^{\text{orig}}$, given by the prover. We further denote by $\text{trans}^{\text{sim}}$ the transcript of the simulated interaction that comprises the commitments, $\{\mathbf{c}_i^{\text{sim}}\}_{i \in [3]}$, a challenge, c^{sim}, and the opening, $\mathbf{o}^{\text{sim}}$, all of them being generated by the simulator. We will now show that the distribution of the two transcripts is indistinguishable, by arguing that distinguishing the two transcripts comes down to inverting a hash function used for committing, which we assume is computationally hard.

Let us thus first observe that both c^{orig} and c^{sim} are sampled uniformly at random from $\{0, 1, 2\}$, which implies that their distributions are indistinguishable. Therefore, we will denote a challenge by c and proceed by analyzing the distribution of the commitments and the openings conditioned on the challenges sampled from the same distribution. We thus obtain the following original transcript:

$$
\begin{aligned}
\mathbf{c}_0^{orig} &= \mathcal{H}(\pi, \mathbf{Hy}, \mathbf{r}_0), \\
\mathbf{c}_1^{orig} &= \mathcal{H}(\pi(\mathbf{y}), \mathbf{r}_1), \\
\mathbf{c}_2^{orig} &= \mathcal{H}(\pi(\mathbf{y} + \sigma(\mathbf{v})), \mathbf{r}_2), \\
\mathbf{o}^{orig} &= \begin{cases} (\pi(\mathbf{y}),\ \pi(\sigma(\mathbf{v})), \mathbf{r}_1, \mathbf{r}_2), & \text{if } c = 0 \\ (\pi,\ \mathbf{y} + \sigma(\mathbf{v}), \mathbf{r}_0, \mathbf{r}_2), & \text{if } c = 1 \\ (\pi,\ \mathbf{y}, \mathbf{r}_0, \mathbf{r}_1), & \text{if } c = 2 \end{cases},
\end{aligned}
$$

where $\pi, \sigma \xleftarrow{\$} \Pi[n]$, $\mathbf{y} \xleftarrow{\$} \mathbb{F}_q^n$, and $\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2 \xleftarrow{\$} \{0, 1\}^m$. Let us then observe the simulated transcript.

149

- If $c = 0$:

$$\mathbf{o}^{\text{sim}} = (\tilde{\pi}(\tilde{\mathbf{y}}), \tilde{\pi}(\tilde{\sigma}(\mathbf{v})), \tilde{\mathbf{r}}_1, \tilde{\mathbf{r}}_2),$$
$$\mathbf{c}_0^{\text{sim}} = \mathcal{H}(\tilde{\pi}, \mathbf{H}\tilde{\mathbf{y}}, \tilde{\mathbf{r}}_0),$$
$$\mathbf{c}_1^{\text{sim}} = \mathcal{H}(\tilde{\pi}(\tilde{\mathbf{y}}), \tilde{\mathbf{r}}_1),$$
$$\mathbf{c}_2^{\text{sim}} = \mathcal{H}(\tilde{\pi}(\tilde{\mathbf{y}} + \tilde{\sigma}(\mathbf{v})), \tilde{\mathbf{r}}_2),$$

Since $\mathbf{y}, \tilde{\mathbf{y}} \xleftarrow{\$} \mathbb{F}_q^n$, $\pi \circ \sigma, \tilde{\pi} \circ \tilde{\sigma} \xleftarrow{\$} \Pi[n]$, as well as $\mathbf{r_1}, \mathbf{r_2}, \tilde{\mathbf{r}}_1, \tilde{\mathbf{r}}_2 \xleftarrow{\$} \{0,1\}^m$, and $\mathbf{v} \in \mathbb{F}_q^n$ is a fixed value common for both transcripts, the distributions of $\mathbf{o}^{\text{orig}}$ and $\mathbf{o}^{\text{sim}}$ are indistinguishable. The distributions of the second and the third commitments, $\mathbf{c}_1^{\text{sim}}$ and $\mathbf{c}_2^{\text{sim}}$, that are opened are then also indistinguishable from the distribution of $\mathbf{c}_1^{\text{orig}}$ and $\mathbf{c}_2^{\text{orig}}$. Finally, the first commitments in both transcripts, namely, $\mathbf{c}_0^{\text{orig}}$ and $\mathbf{c}_0^{\text{sim}}$, are not opened. By the hiding property of the commitment scheme, the distribution of the two is thus computationally indistinguishable.

- If $c = 1$:

$$\mathbf{o}^{\text{sim}} = (\tilde{\pi}, \tilde{\mathbf{y}} + \tilde{\sigma}(\mathbf{v}), \tilde{\mathbf{r}}_0, \tilde{\mathbf{r}}_2),$$
$$\mathbf{c}_0^{\text{sim}} = \mathcal{H}(\tilde{\pi}, \mathbf{H}(\tilde{\mathbf{y}} + \tilde{\sigma}(\mathbf{v})) - \mathbf{s}, \tilde{\mathbf{r}}_0),$$
$$\mathbf{c}_1^{\text{sim}} = \mathcal{H}(\tilde{\pi}(\tilde{\mathbf{y}}), \tilde{\mathbf{r}}_1),$$
$$\mathbf{c}_2^{\text{sim}} = \mathcal{H}(\tilde{\pi}(\tilde{\mathbf{y}} + \tilde{\sigma}(\mathbf{v})), \tilde{\mathbf{r}}_2).$$

Since $\mathbf{y}, \tilde{\mathbf{y}} \xleftarrow{\$} \mathbb{F}_q^n$, $\sigma, \tilde{\sigma} \xleftarrow{\$} \Pi[n]$, as well as $\mathbf{r_1}, \mathbf{r_2}, \tilde{\mathbf{r}}_1, \tilde{\mathbf{r}}_2 \xleftarrow{\$} \{0,1\}^m$, and $\mathbf{v} \in \mathbb{F}_q^n$ is a fixed value common for both transcripts, the distributions of $\mathbf{o}^{\text{orig}}$ and $\mathbf{o}^{\text{sim}}$ are indistinguishable. The distributions of the first and the third commitments, $\mathbf{c}_0^{\text{sim}}$ and $\mathbf{c}_2^{\text{sim}}$, that are opened are then also indistinguishable from the distribution of $\mathbf{c}_0^{\text{orig}}$ and $\mathbf{c}_2^{\text{orig}}$. Finally, the second commitments in both transcripts, namely, $\mathbf{c}_1^{\text{orig}}$ and $\mathbf{c}_1^{\text{sim}}$, are not opened. By the hiding property of the commitment scheme, the distribution of the two is thus computationally indistinguishable.

- If $c = 2$:

$$\mathbf{o}^{\mathrm{sim}} = (\tilde{\pi}, \tilde{\mathbf{y}}, \tilde{\mathbf{r}}_0, \tilde{\mathbf{r}}_1),$$
$$\mathbf{c}_0^{\mathrm{sim}} = \mathcal{H}(\tilde{\pi}, \mathbf{H}\tilde{\mathbf{y}}, \tilde{\mathbf{r}}_0),$$
$$\mathbf{c}_1^{\mathrm{sim}} = \mathcal{H}(\tilde{\pi}(\mathbf{y}), \tilde{\mathbf{r}}_0),$$
$$\mathbf{c}_2^{\mathrm{sim}} = \mathcal{H}(\tilde{\pi}(\tilde{\mathbf{y}} + \tilde{\sigma}(\mathbf{v})), \tilde{\mathbf{r}}_2).$$

Since $\mathbf{y}, \tilde{\mathbf{y}} \xleftarrow{\$} \mathbb{F}_q^n$, $\pi, \tilde{\pi} \xleftarrow{\$} \Pi[n]$, , as well as $\mathbf{r_1}, \mathbf{r}_2, \tilde{\mathbf{r}}_1, \tilde{\mathbf{r}}_2 \xleftarrow{\$} \{0, 1\}^m$ the distributions of $\mathbf{o}^{\mathrm{orig}}$ and $\mathbf{o}^{\mathrm{sim}}$ are indistinguishable. The distributions of the first and the second commitments, $\mathbf{c}_0^{\mathrm{sim}}$ and $\mathbf{c}_1^{\mathrm{sim}}$, that are opened are then also indistinguishable from the distribution of $\mathbf{c}_0^{\mathrm{orig}}$ and $\mathbf{c}_1^{\mathrm{orig}}$. Finally, the first commitments in both transcripts, namely, $\mathbf{c}_0^{\mathrm{orig}}$ and $\mathbf{c}_0^{\mathrm{sim}}$, are not opened and, by the hiding property of the commitment scheme, the distribution of the two is computationally indistinguishable.

$\square$

To simplify the comparison with the original Stern scheme, in the rest of the chapter, whenever we describe our version of the Stern protocol, we observe it as if it was based on the generalized syndrome decoding problem. Nevertheless, we assume that A-SDP in the key-generation part is replaced by A-PKP whenever the underlying field $\mathbb{F}_q$ is of size $q \neq 2$.

Communication cost The cost of communication of an interactive protocol is commonly expressed as the number of bits that are exchanged between the prover and the verifier in the interactive part of the protocol. In the case of Stern's identification protocol, the communication cost, $\mathrm{cost} \in \mathbb{R}$, is calculated as:

$$\mathrm{cost} = r\bigg(3m + \log_2(3) + \frac{1}{3}\big(n\log_2(q) + \log_2(\mathrm{surf}_M(q, n, w)) + 2m\big)$$
$$+ \frac{2}{3}\big(\log_2(n!) + n\log_2(q) + 2m\big)\bigg),$$

where $m \in \mathbb{N}$ is the number of bits of secret coins, $\mathrm{surf}_M(q, n, w) \in \mathbb{N}$ denotes the surface area of a sphere of radius $w \in [\Delta n]$ in the vector space $\mathbb{F}_q^n$

endowed with the weight function $\text{wt}_M(\cdot)$, where $\Delta =: \max_{\mathbf{x} \in \mathbb{F}_q^n} \text{wt}_M(\mathbf{x})$, and $r \in \mathbb{N}$ denotes the number of parallel repetitions of Stern's protocol that guarantee the soundness error of $(2/3)^r$. The term $\text{surf}_M(q, n, w)$ corresponds to the cost of sending the permuted error vector, $\pi(\mathbf{e})$, while $n \log_2(q)n$ and $\log_2 n!$ correspond to the cost of sending a random vector in $\mathbb{F}_q^n$ (such as $\mathbf{y}$ or $(\mathbf{y} + \mathbf{e})$), and a random permutation π, respectively.

For parameters used in practice, the communication cost of Stern's protocol is considered to be rather high (order of $\approx 100kb$) which implies that the scheme is considered to be not efficient enough for practical implementation. We thus aim to reduce the communication cost by using pseudo-random generators, as described in the following subsection.

4.1.2 A more efficient construction

To reduce the communication cost of the Stern protocol, we use the following two approaches. The first one is the most common approach in practice and it relies on the use of pseudo-random generators. Namely, we replace all the randomly generated vectors and permutations in the provers' openings via random *seeds*, which are then expanded to pseudo-random permutations and pseudo-random vectors in the committing phase and in the verification part. This approach reduces the communication cost significantly and it is considered to be rather reliable as its security relies on the security of pseudorandom generators, which are well-studied.

Another approach, also commonly encountered in practice (see, for example, [CVE11], [BGS21]), is to replace the (non-deterministic) commitment scheme with a deterministic one. Like the use of previously introduced pseudo-random generators, this approach reduces the communication cost significantly. However, the approach based on deterministic commitments is less studied and commonly introduced into schemes without much security analysis. As such, it sometimes reveals an unexpected weakness, as we will show in the next subsection. We now present a version of the scheme that combines the two above-described approaches.

Stern's protocol - a more efficient construction

Prover   **Verifier**

.... *Interaction* ....

seed $\overset{\$}{\leftarrow} \{0,1\}^l$,

$(\text{seed}_\pi, \text{seed}_\mathbf{y}) \leftarrow e(\text{seed})$

$\pi \leftarrow e_\pi(\text{seed}_\pi)$, $\pi(\mathbf{y}) \leftarrow e_\mathbf{y}(\text{seed}_\mathbf{y})$

$\mathbf{t} \leftarrow \mathbf{H}\pi^{-1}(\pi(\mathbf{y}))$

$\mathbf{c}_0 \leftarrow \mathcal{H}(\pi, \mathbf{t})$,

$\mathbf{c}_1 \leftarrow \mathcal{H}(\pi(\mathbf{y}))$,

$\mathbf{c}_2 \leftarrow \mathcal{H}(\pi(\mathbf{y} + \mathbf{e})))$

$$\xrightarrow{\{\mathbf{c}_i\}_{i \in [3]}}$$

$$c \overset{\$}{\leftarrow} [3]$$

$$\xleftarrow{c}$$

$\mathbf{o} \leftarrow \begin{cases} (\text{seed}_\mathbf{y}, \pi(\mathbf{e})), & \text{if } c = 0 \\ (\text{seed}_\pi, \mathbf{y} + \mathbf{e}), & \text{if } c = 1 \\ \text{seed}, & \text{if } c = 2 \end{cases}$.

$$\xrightarrow{\mathbf{o}} \quad \ldots$$
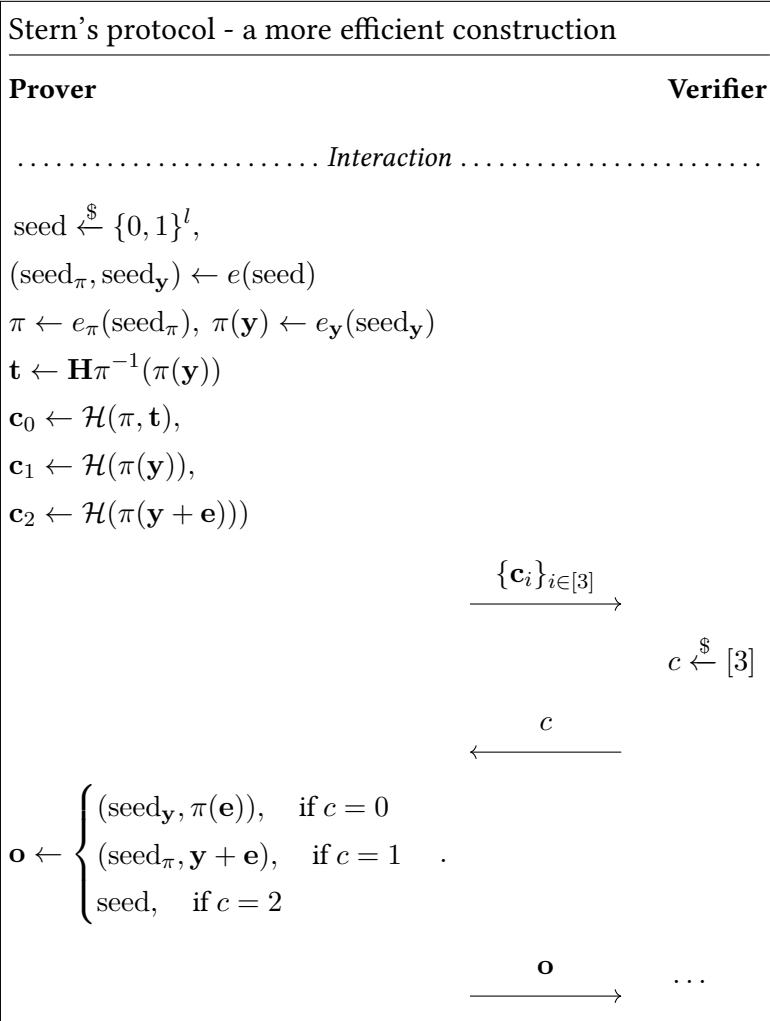
We denote by $e : \{0,1\}^l \rightarrow \{0,1\}^l \times \{0,1\}^l$, $l \in \mathbb{N}$, a pseudo-random generator used for expanding a pseudo-random seed into another two pseudo-random seeds, and by $e_\pi : \{0,1\}^l \rightarrow \Pi[n]$, and $e_\mathbf{y} : \{0,1\}^l \rightarrow \mathbb{F}_q^n$, the pseudo-random generators used for expanding pseudo-random seeds into a pseudo-random permutation and a pseudo-random vector, respectively.

Stern's protocol - a more efficient construction

Prover   **Verifier**

.... *Verification* ....

$\quad$ if $c = 0$:

$\qquad \pi(\mathbf{y}) \leftarrow e_{\mathbf{y}}(\mathbf{o}[0])$

$\qquad \tilde{\mathbf{c}}_1 \leftarrow \mathcal{H}(\pi(\mathbf{y})),\ \tilde{\mathbf{c}}_2 \leftarrow \mathcal{H}(\pi(\mathbf{y}) + \mathbf{o}[1])$

$\qquad$ if $\mathbf{c}_1 = \tilde{\mathbf{c}}_1, \mathbf{c}_2 = \tilde{\mathbf{c}}_2 \wedge \operatorname{wt}_M(\mathbf{o}[1]) = w$:

$\qquad\quad a \leftarrow 1$

$\qquad$ else $: a \leftarrow 0$

$\qquad$ end if

$\quad$ else if $c = 1$:

$\qquad \pi \leftarrow e_\pi(\mathbf{o}[0])$

$\qquad \tilde{\mathbf{c}}_0 \leftarrow \mathcal{H}(\pi, \mathbf{H}\mathbf{o}[1] - \mathbf{s}),\ \tilde{\mathbf{c}}_2 \leftarrow \mathcal{H}(\pi(\mathbf{o}[1]))$

$\qquad$ if $\mathbf{c}_0 = \tilde{\mathbf{c}}_0 \wedge \mathbf{c}_2 = \tilde{\mathbf{c}}_2$:

$\qquad\quad a \leftarrow 1$

$\qquad$ else $: a \leftarrow 0$

$\qquad$ end if

$\quad$ else if $c = 2$:

$\qquad (\operatorname{seed}_\pi, \operatorname{seed}_{\mathbf{y}}) \leftarrow e(\mathbf{o}[0]),$

$\qquad \pi \leftarrow e_\pi(\operatorname{seed}_\pi), \pi(\mathbf{y}) \leftarrow e_{\mathbf{y}}(\operatorname{seed}_{\mathbf{y}})$

$\qquad \tilde{\mathbf{c}}_0 \leftarrow \mathcal{H}(\pi, \mathbf{H}\pi^{-1}(\pi(\mathbf{y}))),\ \tilde{\mathbf{c}}_1 \leftarrow \mathcal{H}(\pi(\mathbf{y}))$

$\qquad$ if $\mathbf{c}_0 = \tilde{\mathbf{c}}_0 \wedge \mathbf{c}_1 = \tilde{\mathbf{c}}_1$:

$\qquad\quad a \leftarrow 1$

$\qquad$ else $: a \leftarrow 0$

$\qquad$ end if

$\quad$ end if

$\square \qquad \xleftarrow{\quad a \quad}$

We observe here that the processing time increases as both the verifier and the prover need to first expand the seeds and then continue with the rest of the protocol. Nevertheless, the communication cost is significantly reduced, as we will show in short.

Reduced communication cost   The reduced cost of communication, obtained by introducing pseudo-random generation and deterministic commitments into the protocol, is calculated as:

$$
\begin{aligned}
\text{cost} =& r\Big(3m + \log_2(3) + \frac{1}{3}\left(d + \log_2(\text{surf}_M(q, n, w))\right) \\
& + \frac{1}{3}\left(d + n\log_2(q)\right) + \frac{1}{3}d\Big),
\end{aligned}
$$

where $d \in \mathbb{N}$ is the length of the seeds used for the pseudo-random generation, $\text{surf}_M(q, n, w) \in \mathbb{N}$ denotes the surface area of a sphere of radius $w \in [\Delta n]$ in the vector space $\mathbb{F}_q^n$ endowed with the weight function $wt_M(\cdot)$, where $\Delta =: \max_{\mathbf{x} \in \mathbb{F}_q^n} wt_M(\mathbf{x})$, and $r \in \mathbb{N}$ denotes the number of parallel repetitions of Stern's protocol that guarantee the soundness error of $(2/3)^r$. The term $\text{surf}_M(q, n, w)$ corresponds to the cost of sending the permuted error vector, $\pi(\mathbf{e})$, while $n\log_2(q)n$ corresponds to the cost of sending a random vector $(\mathbf{y} + \mathbf{e}) \in \mathbb{F}_q^n$.

4.2 Stern Digital Signature Scheme

What we refer to as the *Stern digital signature scheme,* is a digital signature scheme derived from the r-fold parallel repetition version of the Stern identification protocol. In this section, we give the scheme description and explain some of the techniques for reducing the signature size, which is taken as a fundamental measure of the scheme's efficiency.

4.2.1 Basic construction

Here we present the basic scheme construction, where the goal of the signer is to convince the verifier that the message $\mathbf{m} \in \{0,1\}^*$, signed on the signer side, is authentic. To do so, the signer relies on the expected computational hardness of the syndrome decoding problem (i.e. the permuted kernel problem). More precisely, the signer uses the tuples of commitments, challenges, and openings, derived from the Stern identification protocol, to construct a signature that convinces the verifier of its knowledge of the secret key. The key generation part and the signing part are presented in the schemes that follow.

Stern's digital signature - basic construction
Signer   **Verifier**
. . . . *Key Generation* . . . .
$\mathbf{H} \xleftarrow{\$} \mathbb{F}_q^{(n-k)\times n}, \mathbf{e} \xleftarrow{\$} \mathcal{S}_M(q,n,w),$
$\mathbf{s} \leftarrow \mathbf{He},$
$\mathsf{sk} \leftarrow \mathbf{e}, \mathsf{pk} \leftarrow (\mathbf{H}, \mathbf{s}, w)$
$\xrightarrow{\quad \mathsf{pk} \quad}$ . . .
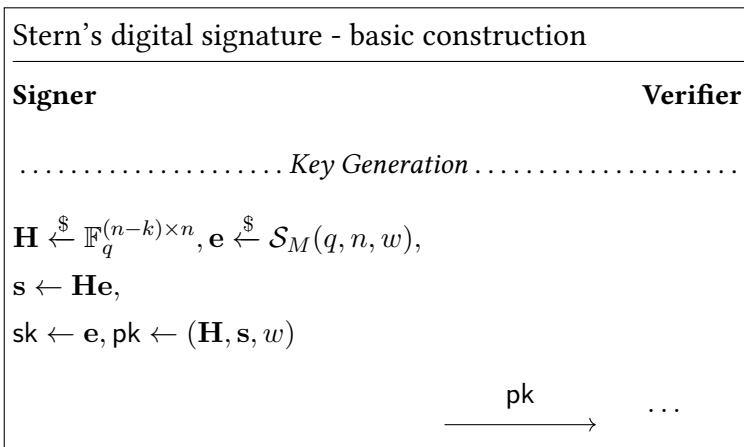
Stern's digital signature - basic construction

Signer   **Verifier**

$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$*Signing*$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$

for $i \in [r]$:

   $\pi^i \xleftarrow{\$} \Pi[n]$, $\mathbf{y}^i \xleftarrow{\$} \mathbb{F}_q^n$, $\mathbf{t}^i \leftarrow \mathbf{H}\mathbf{y}^i$

   $\mathbf{r}_0^i \xleftarrow{\$} \{0,1\}^m$, $\mathbf{c}_0^i \leftarrow \mathcal{H}(\pi^i, \mathbf{t}^i, \mathbf{r}_0^i)$,

   $\mathbf{r}_1^i \xleftarrow{\$} \{0,1\}^m$, $\mathbf{c}_1^i \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i), \mathbf{r}_1^i)$,

   $\mathbf{r}_2^i \xleftarrow{\$} \{0,1\}^m$, $\mathbf{c}_2^i \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i + \mathbf{e}), \mathbf{r}_2^i)$

end for

$\{c^i\}_{i\in[r]} \leftarrow \mathcal{H}_{\text{FS}}(\mathbf{m}, \{\mathbf{c}_j^i\}_{i\in[r],\, j\in[3]})$

$\mathbf{o}^i \leftarrow \begin{cases} (\pi^i(\mathbf{y}^i), \pi^i(\mathbf{e}), \mathbf{r}_1^i, \mathbf{r}_2^i), & \text{if } c^i = 0 \\ (\pi^i, \mathbf{y}^i + \mathbf{e}, \mathbf{r}_0^i, \mathbf{r}_2^i), & \text{if } c^i = 1 \\ (\pi^i, \mathbf{y}^i, \mathbf{r}_0^i, \mathbf{r}_1^i), & \text{if } c = 2 \end{cases}$.

$\text{sign} \leftarrow (\{\mathbf{c}_j^i\}_{i\in[r],\, j\in[3]}, \{c^i\}_{i\in[r]}, \{\mathbf{o}^i\}_{i\in[r]})$

                                                        sign

                                                 $\xrightarrow{\hspace{2cm}}$ . . .

We observe here that the signature is obtained by removing the interaction present in the Stern identification scheme using the Fiat-Shamir transformation, as described in Chapter 1. Namely, the challenges $\{c^i\}_{i\in[r]}$ are obtained from the message $\mathbf{m}$ and the commitments $\{\mathbf{c}_j^i\}_{i\in[r],j\in[3]}$ using the hash function $\mathcal{H}_{\text{FS}} : \{0,1\}^* \to \{0,1\}^r$. Once the interaction with the verifier is removed, the signature is constructed from the obtained commitments, challenges and openings.

Stern's digital signature - basic construction

Signer   **Verifier**

. . . . *Verification* . . . .

for $i \in [r]$:

    if $c^i = 0$:

        $\tilde{\mathbf{c}}_1^i \leftarrow \mathcal{H}(\mathbf{o}^i[0], \mathbf{o}^i[2])$

        $\tilde{\mathbf{c}}_2^i \leftarrow \mathcal{H}(\mathbf{o}^i[0] + \mathbf{o}^i[1], \mathbf{o}^i[3])$

        if $\mathbf{c}_1^i = \tilde{\mathbf{c}}_1^i, \mathbf{c}_2^i = \tilde{\mathbf{c}}_2^i \wedge \mathrm{wt}_M(\mathbf{o}^i[1]) = w$:

          $a^i \leftarrow 1$

        else : $a^i \leftarrow 0$

        end if

    else if $c^i = 1$:

        $\tilde{\mathbf{c}}_0^i \leftarrow \mathcal{H}(\mathbf{o}^i[0], \mathbf{H}\mathbf{o}^i[1] - \mathbf{s}, \mathbf{o}^i[2])$

        $\tilde{\mathbf{c}}_2^i \leftarrow \mathcal{H}(\mathbf{o}^i[0](\mathbf{o}^i[1]), \mathbf{o}^i[3])$

        if $\mathbf{c}_0^i = \tilde{\mathbf{c}}_0^i \wedge \mathbf{c}_2^i = \tilde{\mathbf{c}}_2^i$:

          $a^i \leftarrow 1$

         else : $a^i \leftarrow 0$

        end if

    else if $c^i = 2$:

        $\tilde{\mathbf{c}}_0^i \leftarrow \mathcal{H}(\mathbf{o}^i[0], \mathbf{H}\mathbf{o}^i[1], \mathbf{o}[2])$

        $\tilde{\mathbf{c}}_1^i \leftarrow \mathcal{H}(\mathbf{o}^i[0](\mathbf{o}^i[1]), \mathbf{o}^i[3])$

        if $\mathbf{c}_0^i = \tilde{\mathbf{c}}_0^i \wedge \mathbf{c}_1^i = \tilde{\mathbf{c}}_1^i$:

          $a^i \leftarrow 1$

         else : $a^i \leftarrow 0$

        end if

    end if

end for

$\square$ $\wedge_{i \in [r]} a^i$

       $\longleftarrow$

STERN DIGITAL SIGNATURE SCHEME

The completeness of the above-presented scheme and the soundness error of $(2/3)^r$ follow from the completeness and the soundness error of the corresponding identification scheme. Since the identification scheme is honest-verifier zero-knowledge, the resulting signature scheme is EUF-CMA secure in the *random oracle model*.[2]

Signature size

The basic measure of the scheme efficiency is given as the signature size, expressed as:

$$size = r\left(3m + \log_2(3) + \frac{1}{3}\left(n\log_2(q) + \mathrm{surf}_M(q,n,w) + 2m\right)\right.$$
$$\left. + \frac{2}{3}\left(\log_2(n!) + n\log_2(q) + 2m\right)\right),$$

where $m \in \mathbb{N}$ is the number of bits of randomness, $\mathrm{surf}_M(q,n,w) \in \mathbb{N}$ denotes the surface area of a sphere of radius $w \in [\Delta n]$ in the vector space $\mathbb{F}_q^n$ endowed with the weight function $\mathrm{wt}_M(\cdot)$, where $\Delta =: \max_{\mathbf{x} \in \mathbb{F}_q^n} \mathrm{wt}_M(\mathbf{x})$, and $r \in \mathbb{N}$ denotes the number of parallel repetitions of the corresponding Stern identification protocol that guarantee the soundness error of $(2/3)^r$. The term $\mathrm{surf}_M(q,n,w)$ corresponds to the cost of sending the permuted error vector, $\pi(\mathbf{e})$, while $n\log_2(q)n$ and $\log_2 n!$ correspond to the cost of sending a random vector in $\mathbb{F}_q^n$ (such as $\mathbf{y}$ or $(\mathbf{y}+\mathbf{e})$), and a random permutation π, respectively.

4.2.2 A more efficient construction

In a more efficient scheme construction, we use the same idea as in the case of the Stern identification protocol. Namely, we replace the purely generated random vectors and permutations via pseudo-random generated ones and then provide only their seeds in the openings. Moreover, we use only deterministic commitments. In other words, we take the above-described more efficient version of the Stern identification scheme and apply the Fiat-Shamir transform to it to remove the interaction and obtain a digital signature scheme. As in the case of the identification scheme, the signer (i.e. the

---

[2]This follows from the security of the Fiat-Shamir transformation.

prover in the identification scheme) needs to extract the original seeds and expands these on its side when committing, and the verifier needs to extract and expands the seeds while verifying the consistency between the commitments and the openings. We illustrate this approach through the signing and verifying parts of the scheme that follows.

Stern's digital signature - a more efficient construction

Signer   **Verifier**

$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots Signing\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$

for $i \in [r]$:

$\quad \text{seed}^i \xleftarrow{\$} \{0,1\}^l$

$\quad (\text{seed}^i_\pi, \text{seed}^i_{\mathbf{y}}) \leftarrow e(\text{seed}^i)$

$\quad \pi^i \leftarrow e_\pi(\text{seed}^i_\pi),\ \pi^i(\mathbf{y}^i) \leftarrow e_{\mathbf{y}}(\text{seed}^i_{\mathbf{y}})$

$\quad \mathbf{t}^i \leftarrow \mathbf{H}(\pi^i)^{-1}(\pi^i(\mathbf{y}^i))$

$\quad \mathbf{c}^i_0 \leftarrow \mathcal{H}(\pi^i, \mathbf{t}^i),$

$\quad \mathbf{c}^i_1 \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i)),$

$\quad \mathbf{c}^i_2 \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i + \mathbf{e}))$

$\quad \mathbf{c}^i \leftarrow \mathcal{H}(\{\mathbf{c}^i_j\}_{i \in [r],\ j \in [3]})$

end for

$\{c^i\}_{i \in [r]} \leftarrow \mathcal{H}_{\mathrm{FS}}(\mathbf{m}, \{\mathbf{c}^i_j\}_{i \in [r],\ j \in [3]})$

$\mathbf{o}^i \leftarrow \begin{cases} (\text{seed}^i_{\mathbf{y}}, \pi^i(\mathbf{e})), & \text{if } c^i = 0 \\ (\text{seed}^i_\pi, \mathbf{y}^i + \mathbf{e}), & \text{if } c^i = 1 \\ (\text{seed}^i), & \text{if } c = 2 \end{cases}.$

$\text{sign} \leftarrow (\mathbf{c}^i, \{\mathbf{c}^i_{c^i}\}_{i \in [r]},\ \{c^i\}_{i \in [r]}, \{\mathbf{o}^i\}_{i \in [r]})$

$\xrightarrow{\text{sign}}$ $\dots$

Stern's digital signature - a more efficient construction

Signer   **Verifier**

. . . . *Verification* . . . .

$\quad$ for $i \in [r]$:

$\quad\quad$ if $c^i = 0$:

$\quad\quad\quad$ $\pi^i(\mathbf{y}^i) \leftarrow e_{\mathbf{y}}(\mathbf{o}^i[0])$

$\quad\quad\quad$ $\tilde{\mathbf{c}}_1^i \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i))$, $\tilde{\mathbf{c}}_2^i \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i) + \mathbf{o}^i[1])$

$\quad\quad\quad$ if $\mathrm{wt}_M(\mathbf{o}^i[1]) \neq w$:

$\quad\quad\quad\quad$ $a^i \leftarrow 0$

$\quad\quad\quad$ end if

$\quad\quad$ else if $c^i = 1$:

$\quad\quad\quad$ $\pi^i \leftarrow e_{\pi}(\mathbf{o}^i[0])$

$\quad\quad\quad$ $\tilde{\mathbf{c}}_0^i \leftarrow \mathcal{H}(\pi^i, \mathbf{H}\mathbf{o}^i[1] - \mathbf{s})$, $\tilde{\mathbf{c}}_2^i \leftarrow \mathcal{H}(\pi^i(\mathbf{o}^i[1]))$

$\quad\quad$ else if $c^i = 2$:

$\quad\quad\quad$ $(\mathrm{seed}_{\pi}^i, \mathrm{seed}_{\mathbf{y}}^i) \leftarrow e(\mathbf{o}^i[0])$,

$\quad\quad\quad$ $\pi^i \leftarrow e_{\pi}(\mathrm{seed}_{\pi}^i), \pi^i(\mathbf{y}^i) \leftarrow e_{\mathbf{y}}(\mathrm{seed}_{\mathbf{y}}^i)$

$\quad\quad\quad$ $\tilde{\mathbf{c}}_0^i \leftarrow \mathcal{H}(\pi^i, \mathbf{H}(\pi^i)^{-1}(\pi^i(\mathbf{y}^i)))$, $\tilde{\mathbf{c}}_1^i \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i))$

$\quad\quad$ end if

$\quad\quad$ $\tilde{\mathbf{c}}^i \leftarrow \mathcal{H}(\{\tilde{\mathbf{c}}_j^i\}_{i \in [r], j \in [3] \setminus c^i}, \{\mathbf{c}_{c^i}^i\}_{i \in [r]})$

$\quad$ end for

$\quad$ if $\mathrm{sign} = (\tilde{\mathbf{c}}^i, \{\mathbf{c}_{c^i}^i\}_{i \in [r]},\ \{c^i\}_{i \in [r]}, \{\mathbf{o}^i\}_{i \in [r]})$:

$\quad\quad$ $a \leftarrow 1$

$\quad$ else : $a \leftarrow 0$

$\quad$ end if

$\square$ $\quad\quad \xleftarrow{\quad a \quad}$

CHAPTER 4

Attack on the more efficient construction

Introducing pseudo-random generators, in general, improves the scheme's efficiency but comes with potential security risks when the scheme uses deterministic, unkeyed hash functions, defined in Chapter 1 when committing to a certain value. However, if the values the prover commits to are sampled uniformly at random from a big enough set of elements, the deterministic hash functions do not pose a security threat. This is, in fact, the case in the basic construction of the Stern signature scheme where the signer commits to either random vectors from $\mathbb{F}_q^n$, where $|\mathbb{F}_q^n| = q^n$, or to a random permutation from $\Pi[n]$, where $|\Pi[n]| = n!$. However, in the setting where instead of purely random values, the signer gives the random seeds in its opening, which are then expanded by pseudo-random generators on the verifier side, using a pseudo-random generator decreases the security of the scheme. In Algorithm 11, we describe an attack that exploits this vulnerability.

Algorithm 11 Attack on Stern's scheme (STERN_ATTACK)   *Running time*

Input: $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, $d, r, w \in \mathbb{N}$
Output: $\mathbf{e} \in \mathcal{S}_M(q, n, w)$ s.t. $\mathbf{He} = \mathbf{s}$

1: $q_s \leftarrow \lceil \frac{2^{\lambda/2}}{r} \rceil$ $\triangleright O(1)$
2: **for all** $i \in [q_s]$ **do**
3:      $\mathbf{m} \leftarrow \{0,1\}^*$ $\triangleright O(1)$
4:      $\mathrm{sign}[i] \leftarrow$ GET_SIGNATURE$(\mathbf{m})$ $\triangleright t_{\mathrm{sign}}$
5: **end for**
6: **for all** $i \in [q_s]$ **do**
7:      **for all** $j \in [r]$ **do**
8:          $\mathbf{c}_0[i,j] \leftarrow \mathrm{sign}[i,j].\,\mathrm{commitment}(0)$ $\triangleright O(d)$
9:          $\mathbf{c}_1[i,j] \leftarrow \mathrm{sign}[i,j].\,\mathrm{commitment}(1)$ $\triangleright O(d)$
10:         $\mathbf{c}_2[i,j] \leftarrow \mathrm{sign}[i,j].\,\mathrm{commitment}(2)$ $\triangleright O(d)$
11:         $c[i,j] \leftarrow \mathrm{sign}[i,j].\,\mathrm{challenge}()$ $\triangleright O(1)$
12:         $\mathbf{o}[i,j] \leftarrow \mathrm{sign}[i,j].\,\mathrm{opening}()$ $\triangleright O(\max(l,m)+n)$
13:     **end for**
14: **end for**

15: $\mathcal{L}_0 \leftarrow$ COLLISION_FIND$(\mathbf{c}_1)$	$\triangleright t_{\text{collision\_find}}$
16: **if** $\lvert \mathcal{L}_0 \rvert = 0$ **then**	$\triangleright O(1)$
17: **goto** 1:	$\triangleright O(1)$
18: **end if**	
19: **for all** $((i, j), (i', j')) \in \mathcal{L}_0$ **do**	
20: **if** $c[i, j] = 1 \wedge c[i', j'] \neq 1$ **then**	$\triangleright O(1)$
21: **goto** 26:	$\triangleright O(1)$
22: **end if**	
23: **end for**	
24: **goto** 1:	$\triangleright O(1)$
25: $\text{seed}_\pi \leftarrow \mathbf{o}[i, j][0], \text{seed}_\mathbf{y} \leftarrow \mathbf{o}[i', j'][0]$	$\triangleright O(d)$
26: $\mathbf{y} + \mathbf{e} \leftarrow \mathbf{o}[i, j][1]$	$\triangleright O(n)$
27: $\pi \leftarrow e_\pi(\text{seed}_\pi), \pi(\mathbf{y}) \leftarrow e_\mathbf{y}(\text{seed}_\mathbf{y})$	$\triangleright O(n)$
28: **return** $\mathbf{e} \leftarrow \pi^{-1}(\pi(\mathbf{y} + \mathbf{e}) - \pi(\mathbf{y}))$	$\triangleright O(n)$

The algorithm takes as input the parity check matrix, $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$, the syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$, and the weight $w \in \mathbb{N}$, obtained as a public key in the key generation part of the protocol, as well as $r \in \mathbb{N}$ that is the number of repetitions of the corresponding identification protocol that guarantee soundness error of $\left(\frac{2}{3}\right)^r$. The algorithm's goal is then to fully recover the secret key, $\mathbf{e} \in \mathbb{F}_q^n$. It starts by extracting commitments, challenges, and openings from the signature (lines $6 - 14$ of the algorithm) to obtain the lists of the first, the second, and the third commitments, namely, $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2$, the list of challenges c, and the list of opening, $\mathbf{o}$. Using the list of second commitments, $\mathbf{c}_1$, the algorithm finds collisions between the list elements (line 15). The collisions are then returned as another list, $\mathcal{L}_0$, comprising $((i, j), (i', j'))$ satisfying $\mathbf{c}_1[i, j] = \mathbf{c}_1[i', j']$. If $\mathcal{L}_0$ is empty, the algorithm returns to the beginning (lines 16-18). If $\mathcal{L}_0$ is non-empty, the algorithm continues with a search for $((i, j), (i', j'))$ that, apart from satisfying the constraint on commitments, also satisfies the constraint on challenges given as $c[i, j] = 1 \wedge c[i', j'] \neq 1$ (lines 19-23 of the algorithm). If such an $((i, j), (i', j'))$ exists, the algorithm obtains seed_π and $\mathbf{y} + \mathbf{e}$ from $\mathbf{o}[i, j]$, and $\text{seed}_\mathbf{y}$ from the $\mathbf{o}[i', j']$, as presented by the lines 25-26 of the algorithm. Having all three values, the algorithm first expands seed_π into π and $\text{seed}_\mathbf{y}$ into $\pi(\mathbf{y})$ (line 27), and then reconstructs the

error vector (line 28) as:

$$\mathbf{e} \leftarrow \pi^{-1}(\pi(\mathbf{y} + \mathbf{e}) - \pi(\mathbf{y})).$$

We observe here that the core idea of the algorithm is to match the information obtained through different openings, namely, an opening that reveals information on the values the signer committed to in the first and the third commitment, and an opening that reveals information on the value used for the second commitment. By the scheme construction, it is impossible to extract the information on all three values used for the commitment in one round of the corresponding identification protocol. However, by finding the collision in the commitments, one can aim for gathering information about matching commitments through different rounds of the protocol.

The reason this attack does not work for the original scheme is that finding a collision in the second commitment happens with an overwhelmingly small probability as the number of possible values to commit to is q^n. Finding a collision in the second commitment with a high probability would thus require $O(q^{2n})$ commitments. The more efficient scheme we described, on the other hand, reduces the number of values from which the second commitment can be obtained and increases the probability of finding a collision. This probability becomes a constant value, which leads to a practically implementable attack. We summarize this reasoning in the following proposition.

> **Proposition 4.2.1** (Cost of the attack on the Stern's protocol). *The attack presented above finds the secret key, $\mathbf{e}$, in time $O(2^{\lambda/2})$ having $O\left(\lceil \frac{2^{\lambda/2}}{r} \rceil\right)$ signatures at its disposal. The scheme thus preserves at most $\lambda/2$ bits of security.*

Proof. We first observe that under the assumption that the hash function $\mathcal{H}(\cdot)$ is *collision resistant*, finding $\left((i, j), (i', j')\right)$ that satisfies $\mathbf{c}_1[i, j] = \mathbf{c}_1[i', j']$ implies that the corresponding values of $\pi(\mathbf{y})$ are equal. This further implies that the values of $\text{seed}_\mathbf{y}$, used for calculating $\pi(\mathbf{y})$, are also equal. Having $c[i, j] = 1$ then gives the opening for the first and the third commitment, namely the values of seed_π and $\mathbf{y} + \mathbf{e}$ in $\mathbf{o}[i, j]$, and having $c[i', j'] \neq 1$ gives the value of $\text{seed}_\mathbf{y}$ in the opening for the second commitment, $\mathbf{o}[i', j'][0]$. The three values are enough to recover the secret key $\mathbf{e}$. The question now is what is the probability of finding such $\left((i, j), (i', j')\right)$.

Let us observe that having $\lceil \frac{2^{\lambda/2}}{r} \rceil$ signatures at the algorithm's disposal implies that there are $rq_s = r\lceil \frac{2^{\lambda/2}}{r} \rceil \geq 2^{\lambda/2}$ values of second commitment in these signatures and the same number of corresponding seeds, $\text{seed}_{\mathbf{y}}[i,j] \in \{0,1\}^{\lambda}$, used for obtaining the second commitment. Now, since the algorithm has $2^{\lambda/2}$ commitments at its disposal and there exists 2^{λ} seeds taken from the set $\{0,1\}^{\lambda}$, by birthday paradox, the probability that the algorithm finds a collision in the seeds is lower bounded by $\Omega(1)$. Furthermore, the probability that for the given $\big((i,j),(i',j')\big)$, $c[i,j] = 1$ and $c[i',j'] \neq 1$ is a constant value, more precisely, it is equal to $2/9$. Therefore, the overall probability of finding $\big((i,j),(i',j')\big)$ that satisfies both the constraint on the second commitment and the constraint on the challenge is given by $\Omega(1)$. Having $O\big(\lceil \frac{2^{\lambda/2}}{r} \rceil\big)$ signatures then guarantees that using the above-described attack, the secret key is recovered with probability $\Omega(1)$. $\qquad \square$

This attack impacts in particular the scheme presented in [BGS21]. In the next section, we propose a way to mitigate the attack.

Mitigating the attack

Given that the attack exploits the vulnerability of the approach suggested for reducing the signature size, the method of preventing it relies primarily on adding randomness to the hash functions used for committing and generating challenges through the Fiat-Shamir transformation. However, instead of adding 'fresh' randomness for each iteration of the corresponding identification protocol, which would significantly increase the signature size, we use the so-called $\text{salt} \in \{0,1\}^*$ that is generated once per each signature and used in each hash function. More precisely, we modify the scheme so that the commitments are now obtained as follows:

$$\mathbf{c}_0^i \leftarrow \mathcal{H}(\pi^i, \mathbf{t}^i, \text{salt}, \text{index}),$$
$$\mathbf{c}_1^i \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i), \text{salt}, \text{index}),$$
$$\mathbf{c}_2^i \leftarrow \mathcal{H}(\pi^i(\mathbf{y}^i + \mathbf{e}), \text{salt}, \text{index}),$$

and challenges are calculated as:

$$\{c^i\}_{i \in [r]} \leftarrow \mathcal{H}_{\text{FS}}(\mathbf{m}, \{\mathbf{c}_j^i\}_{i \in [r],\, j \in [3]}, \text{salt}, \text{index})$$

where $\text{salt} \in \{0,1\}^{2\lambda}$ is sampled uniformly at random for each call of the signing oracle, and $\text{index} \in \mathbb{N}$ is initialized at 0 and then increased with each next instantiation of index. The signature is then obtained as:

$$\text{sign} \leftarrow \left(\mathbf{c}^i, \{\mathbf{c}^i_{c^i}\}_{i\in[r]}, \; \{c^i\}_{i\in[r]}, \{\mathbf{o}^i\}_{i\in[r]}, \text{salt}, \text{index} \right).$$

Finally, the verification part follows the same line of reasoning and adds the random salt and the index when calculating the commitments, challenges, and signature.

To see that the introduced modification results in the signature scheme resistant to the above-described attack, we recall that the previous attack relies on finding the collision in the second commitment. In the previous version of the protocol, namely, the version without the salt and the index in the hash functions, finding the collision was possible with probability close to 1 when the algorithm had $q_s = \lceil \frac{2^{\lambda/2}}{r} \rceil$ signatures at its disposal. Introducing the index into the hash function enables one to reduce the probability of finding the collision in one signature while introducing the salt reduces the probability of finding a collision between different signatures. The salt size of 2λ thus forces an attacker to obtain at least $q_s = \lceil \frac{2^\lambda}{r} \rceil$ signatures before recovering the key, and thus regains λ bits of security. The signature scheme then resists the attack in exchange for a slight increase of the signature size, namely, at the cost of $2\lambda + 1$ bits, where $\lambda \in \mathbb{N}$ is a security parameter. While we do not present here a full security argument that this method preserves the security of the scheme, security arguments for this mitigation appear in our preprint.

Signature size

The size of the signature in the more efficient construction is expressed as:

$$\text{size} = r \left(l + \log_2(3) + \frac{1}{3} \left(d + \log_2(\text{surf}_M(q,n,w)) \right) + \frac{1}{3} \left(d + n\log_2(q) \right) + \frac{1}{3}d \right) + l + 2\lambda + 1,$$

where $l \in \mathbb{N}$ is the length of the output of the hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^l$, $d \in \mathbb{N}$ is the length of the seeds used for the pseudo-random generation, $\text{surf}_M(q,n,w) \in \mathbb{N}$ denotes the surface area of a sphere of radius

$w \in [\Delta n]$ in the vector space $\mathbb{F}_q^n$ endowed with the weight function $\text{wt}_M(\cdot)$, where $\Delta =: \max_{\mathbf{x} \in \mathbb{F}_q^n} \text{wt}_M(\mathbf{x})$, and $r \in \mathbb{N}$ denotes the number of parallel repetitions of Stern's protocol that guarantee the soundness error of $(2/3)^r$. The term $\text{surf}_M(q, n, w)$ corresponds to the cost of sending the permuted error vector, $\pi(\mathbf{e})$, while $n \log_2(q) n$ corresponds to the cost of sending a random vector $(\mathbf{y} + \mathbf{e}) \in \mathbb{F}_q^n$.

4.2.3   Numerical results

We present here the signature sizes obtained for the parameter choices corresponding to the hardest instances of the syndrome decoding problem over the Hamming and Lee weight. To obtain these, we rely on the numerical results presented in Chapter 3, and the standard concrete security requirements that dictate the choice of r, l, and d.[3] We then obtain the following numerical results for the basic and optimized schemes.

Signature sizes, size in kB				
q	Non-optimized scheme		Optimized scheme	
	wt_H	wt_L	wt_H	wt_L
2	253.05	253.05	26.21	26.21
3	116.54	116.54	21.81	21.81
5	138.54	95.48	27.62	21.41
7	126.47	90.94	28.29	22.71
13	113.23	79.27	29.38	23.29

Table 4.1: Signature sizes of non-optimized and optimized schemes

We observe here that the decrease in the signature size obtained by introducing pseudorandom generators is significantly bigger than the one introduced by the change of the weight function. Nevertheless, we remark that, as we have shown in the previous subsection, the use of pseudo-random generators can potentially lead to a security leak in certain settings. Replacing

---

[3]More details on the choices of the values of parameters r, l and d are presented in our paper [CE23].

the Hamming weight with the Lee weight, on the other hand, comes at no security costs. Taking all of this into account, weight function replacement can be seen not as a stand-alone optimization technique that, in combination with some other techniques, can lead to a further decrease in the signature sizes and potentially real-world applicable schemes. Another observation to be made is about the choice of the alphabet size. Namely, we observe that the smallest signature size is obtained for $q = 5$, which is the alphabet size that is not commonly used in the design of cryptographic protocols. As such, it might be an interesting choice for new protocol designs.

CHAPTER 5

CONCLUSION

The focus of this thesis was on the syndrome decoding problem, its generalization, cryptanalysis, and application to post-quantum cryptography. We now give a brief summary of the content presented in each chapter of the thesis along with the final observations and directions for future work.

Chapter 1 We gave a brief overview of the concepts in coding theory, algorithms and complexity, and cryptography that are relevant to the context of this thesis.

Chapter 2 We introduced the generalized version of the syndrome decoding problem and showed that its average-case version is polynomial-time reducible to the average-case version of the permuted kernel problem. We then explained the method for calculating the surface area of a sphere in an arbitrary vector space endowed with an elementwise weight function. Calculating the sphere surface area was a stepping stone for showing the abovementioned reduction as well as for determining the asymptotic complexity of the problem from the cryptanalytic perspective, as presented in Chapter 3. The method itself, previously known only for the Lee weight, was generalized to encounter an arbitrary elementwise weight function and can be observed as a contribution of its own.

Chapter 3 We adapted the information set decoding framework, known as the set of the best attacks against the syndrome decoding problem over the Hamming weight, to our more general setting. We then used this framework to determine the asymptotic complexity of the generalized syndrome decoding problem in both classical and quantum settings. At the end of the chapter, we presented the numerical results on the asymptotic running time of these algorithms when attacking the syndrome decoding problem over the Hamming weight and the Lee weight. The numerical results indicate that the problem in the Lee weight setting can be considered a more consistent version of the syndrome decoding problem when the alphabet size increases. Moreover, the numerical results on the hardest instances of the problem indicate that the problem over the Lee weight is harder than over the Hamming weight for all alphabet sizes greater than 3.

Chapter 4 We finished our analysis with Chapter 4 in which we presented the cryptographic application of the generalized syndrome decoding problem. Namely, we adapted the digital signature scheme, which was originally based on the syndrome decoding problem over the Hamming weight, to our more general setting. We then replaced the syndrome decoding problem with the permuted kernel problem to preserve the zero-knowledge property of the original scheme. The scheme was then optimized to increase the efficiency using pseudo-random generators. We then showed the vulnerability of this optimization method through an attack on the scheme and then suggested a method for mitigating this attack. We ended the chapter by presenting the numerical results of the signature sizes of the digital signature we designed.

Future directions In the follow-up work, we would like to address the following questions. The first, and the most natural one would be to ask if we can combine some other, more recent optimization techniques, such as *MPC-in-the-head*, to obtain more efficient digital signatures based on the generalized syndrome decoding problem. On the cryptanalysis side, we would be interested to know if it would be possible to utilize some more advanced ISD techniques, like those based on representations or a nearest neighbor search, to obtain a better attack on the generalized problem, or in particular, its Lee weight version. Would other quantum search algorithms outperform our approach based on the Grover search and amplitude amplification?

Bibliography

[AB09]   Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009. ISBN: 978-0-521-42426-4 (cit. on pp. 2, 46).

[ApS19]   MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.* 2019 (cit. on pp. 85, 132).

[Ast84]   Jaakko Astola. "On the asymptotic behaviour of Lee-codes". In: *Discret. Appl. Math.* 8.1 (1984), pp. 13–23 (cit. on pp. 77, 81, 82, 84, 132).

[Bar94]   Sasha Barg. "Some New NP-Complete Coding Problems". In: *Probl. Peredachi Inf.* (1994) (cit. on p. 74).

[Bec+12]   Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. "Decoding Random Binary Linear Codes in $2^{n/20}$: How 1+1=0 Improves Information Set Decoding". In: *IACR Cryptol. ePrint Arch.* (2012), p. 26 (cit. on p. 115).

[Ber10]   Daniel J. Bernstein. "Grover vs. McEliece". In: *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings.* Ed. by Nicolas Sendrier. Vol. 6061. Lecture Notes in Computer Science. Springer, 2010, pp. 73–80 (cit. on pp. 90, 115).

[Ber68]   Elwyn R. Berlekamp. *Algebraic coding theory.* McGraw-Hill series in systems science. McGraw-Hill, 1968. ISBN: 0070049033 (cit. on pp. 2, 79).

[BGS21]   Loïc Bidoux, Philippe Gaborit, and Nicolas Sendrier. *Quasi-Cyclic Stern Proof of Knowledge.* 2021. arXiv: 2110.05005 [cs.CR] (cit. on pp. 152, 165).

[Bog00]   K.P. Bogart. *Introductory Combinatorics.* Harcourt/Academic Press, 2000. ISBN: 9780121108304 (cit. on p. 71).

[Car+22]   Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. "Statistical Decoding 2.0: Reducing Decoding to LPN". In: *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13794. Lecture Notes in Computer Science. Springer, 2022, pp. 477–507 (cit. on p. 93).

[CDE21]   André Chailloux, Thomas Debris-Alazard, and Simona Etinski. "Classical and Quantum Algorithms For Generic Syndrome Decoding Problems and Applications to the Lee Metric". In: *Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20–22, 2021, Proceedings*. Daejeon, Korea (Republic of): Springer-Verlag, 2021, 44–62. ISBN: 978-3-030-81292-8 (cit. on pp. 70, 93, 135).

[CE23]   André Chailloux and Simona Etinski. *On the (In)security of optimized Stern-like signature schemes*. 2023 (cit. on pp. 139, 167).

[CVE11]   Pierre-Louis Cayrel, Pascal Véron, and Sidi Mohamed El Yousfi Alaoui. "A Zero-Knowledge Identification Scheme Based on the q-ary Syndrome Decoding Problem". In: *SAC*. 2011, pp. 171–186 (cit. on p. 152).

[Deb23]   Thomas Debris-Alazard. *Code-based Cryptography: Lecture Notes*. 2023. arXiv: 2304.03541 [cs.CR] (cit. on p. 12).

[DJ92]   D Deutsch and R Jozsa. *Rapid Solution of Problems by Quantum Computation*. Tech. rep. GBR, 1992 (cit. on p. 36).

[Dum91]   Ilya Dumer. "On minimum distance decoding of linear codes". In: Jan. 1991 (cit. on p. 105).

[FS09]   Matthieu Finiasz and Nicolas Sendrier. "Security Bounds for the Design of Code-Based Cryptosystems". In: *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*. Ed. by Mitsuru Matsui.

Vol. 5912. Lecture Notes in Computer Science. Springer, 2009, pp. 88–105 (cit. on p. 94).

[FS86]   Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*. Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 186–194 (cit. on p. 65).

[GMR88]   Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks". In: *SIAM J. Comput.* 17.2 (1988), pp. 281–308 (cit. on p. 56).

[Got98]   Daniel Gottesman. "The Heisenberg Representation of Quantum Computers". In: (1998) (cit. on p. 27).

[GS91]   Danièle Gardy and Patrick Solé. "Saddle Point Techniques in Asymptotic Coding Theory". In: *Algebraic Coding, First French-Soviet Workshop, Paris, France, July 22-24, 1991, Proceedings*. Ed. by Gérard D. Cohen, Simon Litsyn, Antoine Lobstein, and Gilles Zémor. Vol. 573. Lecture Notes in Computer Science. Springer, 1991, pp. 75–81 (cit. on p. 81).

[HW21]   Anna-Lena Horlemann-Trautmann and Violetta Weger. "Information set decoding in the Lee metric with applications to cryptography". In: *Adv. Math. Commun.* 15.4 (2021), pp. 677–699 (cit. on p. 81).

[Kir18]   Elena Kirshanova. "Improved Quantum Information Set Decoding". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*. Ed. by Tanja Lange and Rainer Steinwandt. Vol. 10786. Lecture Notes in Computer Science. Springer, 2018, pp. 507–527 (cit. on p. 115).

[KL14]   Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. ISBN: 9781466570269 (cit. on pp. 2, 53, 66).

[KT17]   Ghazal Kachigar and Jean-Pierre Tillich. "Quantum Information
           Set Decoding Algorithms". In: *Post-Quantum Cryptography - 8th
           International Workshop, PQCrypto 2017, Utrecht, The Netherlands,
           June 26-28, 2017, Proceedings*. Ed. by Tanja Lange and Tsuyoshi
           Takagi. Vol. 10346. Lecture Notes in Computer Science. Springer,
           2017, pp. 69–89 (cit. on p. 115).

[LB88]   Pil Joong Lee and Ernest F. Brickell. "An Observation on the Se-
           curity of McEliece's Public-Key Cryptosystem". In: *Advances in
           Cryptology - EUROCRYPT '88, Workshop on the Theory and Ap-
           plication of of Cryptographic Techniques, Davos, Switzerland, May
           25-27, 1988, Proceedings*. Ed. by Christoph G. Günther. Vol. 330.
           Lecture Notes in Computer Science. Springer, 1988, pp. 275–280
           (cit. on pp. ix, 103).

[Lee58]   C. Y. Lee. "Some properties of nonbinary error-correcting codes".
           In: *IRE Trans. Inf. Theory* 4.2 (1958), pp. 77–82 (cit. on p. 80).

[McE78]   Robert J. McEliece. "A public key cryptosystem based on alge-
           braic coding theory". In: 1978 (cit. on pp. 17, 52, 140).

[MMT11]   Alexander May, Alexander Meurer, and Enrico Thomae. "Decod-
           ing Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$".
           In: *Advances in Cryptology - ASIACRYPT 2011 - 17th International
           Conference on the Theory and Application of Cryptology and In-
           formation Security, Seoul, South Korea, December 4-8, 2011. Pro-
           ceedings*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073.
           Lecture Notes in Computer Science. Springer, 2011, pp. 107–124
           (cit. on p. 115).

[NC16]   Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation
           and Quantum Information (10th Anniversary edition)*. Cambridge
           University Press, 2016. ISBN: 978-1-10-700217-3 (cit. on pp. 2, 31).

[Pet10]   Christiane Peters. "Information-Set Decoding for Linear Codes
           over F_q". In: *Post-Quantum Cryptography, Third International Work-
           shop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Pro-
           ceedings*. Ed. by Nicolas Sendrier. Vol. 6061. Lecture Notes in Com-
           puter Science. Springer, 2010, pp. 81–94 (cit. on pp. xi, 94, 136).

[Pra62]   Eugene Prange. "The use of information sets in decoding cyclic codes". In: *IRE Trans. Inf. Theory* 8.5 (1962), pp. 5–9 (cit. on pp. ix, 94, 101).

[PS96]   David Pointcheval and Jacques Stern. "Security Proofs for Signature Schemes". In: *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding.* Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Springer, 1996, pp. 387–398 (cit. on p. 65).

[RU08]   Thomas J. Richardson and Rüdiger L. Urbanke. *Modern Coding Theory.* Cambridge University Press, 2008. ISBN: 978-0-521-85229-6 (cit. on pp. 2, 11).

[Sha89]   Adi Shamir. "An Efficient Identification Scheme Based on Permuted Kernels (Extended Abstract)". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings.* Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 606–609 (cit. on p. 75).

[Sho94]   Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994.* IEEE Computer Society, 1994, pp. 124–134 (cit. on p. 36).

[Sim97]   Daniel R. Simon. "On the Power of Quantum Computation". In: *SIAM J. Comput.* 26.5 (1997), pp. 1474–1483 (cit. on p. 36).

[SS79]   Richard Schroeppel and Adi Shamir. "A T Sˆ2 = O(2ˆn) Time/Space Tradeoff for Certain NP-Complete Problems". In: *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979.* IEEE Computer Society, 1979, pp. 328–336 (cit. on pp. 107, 110).

[SS81]   Richard Schroeppel and Adi Shamir. "A $T=O(2^{n/2})$, $S=O(2^{n/4})$ Algorithm for Certain NP-Complete Problems". In: *SIAM J. Comput.* 10.3 (1981), pp. 456–464 (cit. on p. 110).

[Ste88]   Jacques Stern. "A method for finding codewords of small weight".
           In: *Coding Theory and Applications, 3rd International Colloquium,*
           Toulon, France, November 2-4, 1988, Proceedings. Ed. by Gérard D.
           Cohen and Jacques Wolfmann. Vol. 388. Lecture Notes in Com-
           puter Science. Springer, 1988, pp. 106–113 (cit. on pp. ix, 105).

[Ste93]   Jacques Stern. "A New Identification Scheme Based on Syndrome
           Decoding". In: *Advances in Cryptology - CRYPTO '93, 13th An-*
           nual International Cryptology Conference, Santa Barbara, Califor-
           nia, USA, August 22-26, 1993, Proceedings. Ed. by Douglas R. Stin-
           son. Vol. 773. Lecture Notes in Computer Science. Springer, 1993,
           pp. 13–21 (cit. on p. 140).

[Wag02]   David A. Wagner. "A Generalized Birthday Problem". In: *Advances*
           in Cryptology - CRYPTO 2002, 22nd Annual International Cryp-
           tology Conference, Santa Barbara, California, USA, August 18-22,
           2002, Proceedings. Ed. by Moti Yung. Vol. 2442. Lecture Notes in
           Computer Science. Springer, 2002, pp. 288–303 (cit. on p. ix).

[Weg+20]   Violetta Weger, Massimo Battaglioni, Paolo Santini, Anna-Lena
           Horlemann-Trautmann, and Edoardo Persichetti. "On the Hard-
           ness of the Lee Syndrome Decoding Problem". In: *CoRR* abs/2002.12785
           (2020). arXiv: 2002.12785 (cit. on p. 74).