RUHR
UNIVERSITÄT
BOCHUM

RUB

Practical Cryptanalysis of Code-Based Crypto Schemes

Doctoral Dissertation

Submitted in partial fulfillment of the requirements for the degree of
doctor rerum naturalium (Dr. rer. nat.)
to the Faculty of Computer Science
Ruhr-Universität Bochum

by
Floyd Zweydinger

Acknowledgements

First and foremost I would like to thank my advisor Alexander May for accepting me as his student and for giving me the opportunity to work in his group during the last three years. I would also like to thank Gregor Leander for agreeing to be the second reviewer of this thesis. Additionally, I want to thank Andre - Don Rainer - Esser, without you, the whole work wouldn't be possible. As well as Marion Reinhardt-Kalender for kicking my ... to get things done. Furthermore, I thank the whole group and friends: Lars, Julian, Maya, Önder, Richard, Timo for their fruitful discussions. And of course: Meret, thank you for everything. My work was funded by BMBF under Industrial Blockchain – iBlockchain. Special thanks also to them.

Eigenständigkeitserklärung

Ich versichere an Eides statt, dass ich die eingereichte Dissertation selbstständig und ohne unzulässige fremde Hilfe verfasst, andere als die in ihr angegebene Literatur nicht benutzt und dass ich alle ganz oder annähernd übernommenen Textstellen sowie verwendete Grafiken, Tabellen und Auswertungsprogramme kenntlich gemacht habe. Außerdem versichere ich, dass die vorgelegte elektronische mit der schriftlichen Version der Dissertation übereinstimmt und die Abhandlung in dieser oder ähnlicher Form noch nicht anderweitig als Promotionsleistung vorgelegt und bewertet wurde.

Ich erkläre weiterhin, dass digitale Abbildungen nur die originalen Daten ent-halten oder eine eindeutige Dokumentation von Art und Umfang der inhalts-verändernden Bildbearbeitung beigefügt wurde und dass keine kommerzielle Vermittlung oder Beratung in Anspruch genommen wurde.

Bochum, den _____ _____

(Floyd Zweydinger)

Contents

1 Introduction

The field of *Post-Quantum Cryptography* concerns the design and analysis of cryptographic protocols that are robust against attackers utilizing large-scale quantum computers. This area of research is of increasing importance, given the potential advancements in quantum computing and the consequences for the security of classical cryptographic schemes. These consequences could be widespread breaches of sensitive data, compromising individuals' privacy and financial security. Therefore, the urgency to develop and implement secure post-quantum cryptographic schemes cannot be overstated.

Current classical cryptographic constructions rely heavily on number-theoretic assumptions, like the factoring problem or the discrete logarithm problem. Thus, the existence of efficient quantum algorithms for solving these problems has prompted exploration into various security foundations. Consequently, submissions to the NIST standardization effort in 2016 encompass a broad spectrum of security assumptions. Among these, lattice-based candidates stand out, drawing on challenging problems from algebraic number theory, while code-based assumptions are based upon hard problems in coding theory.

In total 82 schemes were submitted, of which 69 met the acceptance criteria. Among these, 26 were lattice-based and 19 code-based submissions. After seven years in 2023, NIST announced the lattice-based scheme CRYSTALS-Kyber [BDK+18] as the new encryption standard. At the same time, the NIST also announced that three other schemes would advance into a final round to select an additional standard, namely *Classic McEliece* [BCL+17], *BIKE* [ABB+17], and *HQC* [MAB+18]. These three schemes are all based on linear codes, highlighting the importance of code-based cryptography. This was done by NIST, specifically to diversify the landscape of encryption standards.

The Computational Syndrome Decoding Problem (CSD) is one of the most well-known computational problems from coding theory. It can be described as follows. Given a random (binary) parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ and a syndrome $\mathbf{s} = \mathbf{He} \in \mathbb{F}_2^{n-k}$ decode to an error vector $\mathbf{e} \in \mathbb{F}_2^n$ with hamming weight ω. This problem serves as the foundation for many code-based cryptographic schemes, making assessing its complexity crucial for evaluating the security of proposed systems. When appropriately choosing the parameters n, k, ω the CSD problem becomes hard, with the most efficient algorithms needing exponential runtime. This is particularly true when $\omega = c \cdot n$, for some constant c.

A significant lineage of generic decoding algorithms solving the CSD, originating from Prange's work in 1962 [Pra62], is known as Information Set Decoding (ISD). Over the past six decades, these algorithms have garnered considerable attention from coding theorists. At a conceptual level, these algorithms transform the original Computational Syndrome Decoding (CSD) instance $\mathbf{H}, \mathbf{s}$, with parameters n, k, ω, into a related CSD instance, $\mathbf{H}', \mathbf{s}'$ with reduced parameters n', k', ω'. Subsequently, one solves the somewhat smaller instance and then tries to map the resulting solution back to a solution of the original problem. In essence, this transformation is a randomized procedure, predominantly yielding instances $\mathbf{H}', \mathbf{s}'$ that lack solutions to the original problem. This is resolved by repetitively applying the randomized transformation a sufficient number of times.

Almost 30 years after Prange's original work, Stern [Ste89] published the first exponential speedup in 1989. More than twenty years later May, Meurer and Thomae [MMT11] as well as Becker, Joux, May and Meurer [BJMM12] published two innovative algorithms. Both methods utilized the *representation technique*, to varying degrees, to solve the transformed problem more efficiently.

The main idea behind the representation technique is to artificially expand the search space while increasing the expected number of solutions even more. For simplicity let us introduce the following notation. By $\mathcal{W}_\omega^n$ we denote the set of all vectors $\mathbf{e} \in \mathbb{F}_2^n$ with hamming weight ω. Consider the following basic algorithm for the CSD problem. Suppose ω is divisible by 2 and there exists a solution $\mathbf{e} = (\mathbf{e}_1\mathbf{e}_2)$, with $\mathbf{e}_i \in \mathcal{W}_{\omega/2}^{n/2}$. By similarly splitting the parity check matrix $\mathbf{H} = (\mathbf{H}_1\mathbf{H}_2)$, with $\mathbf{H}_i \in \mathbb{F}_2^{(n-k)\times n/2}$, we can rewrite the parity check equation as follows:

$$\begin{aligned} \mathbf{He} &= \mathbf{s} \\ \mathbf{H}_1\mathbf{e}_1 + \mathbf{H}_2\mathbf{e}_2 &= \mathbf{s} \\ \mathbf{H}_1\mathbf{e}_1 &= \mathbf{H}_2\mathbf{e}_2 + \mathbf{s}. \end{aligned} \tag{1}$$

The algorithm now computes a list $\mathcal{L}_1$ containing all candidates $\mathbf{e}_1 \in \mathcal{W}_{\omega/2}^{n/2}$ and sorts the lists according to the values $\mathbf{H}_1\mathbf{e}_1$. Next, it computes $\mathbf{H}_2\mathbf{e}_2 + \mathbf{s}$ for all $\mathbf{e}_2 \in \mathcal{W}_{\omega/2}^{n/2}$, and searches for a collision, i.e. an $\mathbf{e_1}$, s.t. $\mathbf{H}_1\mathbf{e}_1 = \mathbf{H}_2\mathbf{e}_2 + \mathbf{s}$. Finding such a collision recovers a solution $\mathbf{e}$ to the CSD problem. Neglecting all polynomial factors inherited by sorting and searching, constructing the list $\mathcal{L}_1$ has a complexity of $\binom{n/2}{\omega/2}$. Similarly, the worst-case complexity of finding a collision can be determined by the number of $\mathbf{e}_2$ vectors that need to be enumerated, which is also $\binom{n/2}{\omega/2}$. Thus, we obtained an algorithm for the CSD problem with complexity $\binom{n/2}{\omega/2} \approx \binom{n}{\omega}^{1/2}$.

Note that the algorithm splits the search space $\mathcal{W}_\omega^n$ into two distinct parts, meaning the solution vector $\mathbf{e}$ can be written as a *unique* sum of $\mathbf{e}_1, \mathbf{e}_2 \in \mathcal{W}_{\omega/2}^{n/2} \times \mathcal{W}_{\omega/2}^{n/2}$. The core idea of the representation technique is to relax this decomposition by allowing the vectors $\mathbf{e}_1, \mathbf{e}_2$ to be chosen from $\mathcal{W}_{\omega/2}^n \times \mathcal{W}_{\omega/2}^n$. Consequently, many such vectors will exist, enabling every $\mathbf{e} \in \mathcal{W}_\omega^n$ to be *represented* in numerous ways. Using this decomposition yields $R = \binom{\omega}{\omega/2}$ different *representations* of the unique solution $\mathbf{e}$. On the downside, this approach increases the search space to $|\mathcal{W}_{\omega/2}^n| = \binom{n}{\omega/2} \gg \binom{n/2}{\omega/2}$. The second key insight of the representation technique is that it is sufficient to find just one of these representations to solve the original problem. Specifically, we aim to construct two lists $\mathcal{L}_1, \mathcal{L}_2 \subset \mathcal{W}_{\omega/2}^n$, s.t. there is a high probability of finding a representation $(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$. These lists must be at least as large as $|\mathcal{W}_{\omega/2}^n|/R = \binom{n}{\omega/2}/R$. Assuming an algorithm computing these lists in minimal time, recovering the solution $\mathbf{e}$ then becomes a matter of finding a collision, similar to the previous algorithm. Overall, this results in an algorithm with a time complexity of $\binom{n}{\omega/2}/R \leq \binom{n/2}{\omega/2}$, which holds for all $2 \leq \omega \leq n$. This simplified overview hides significant technical details, particularly regarding the algorithm for computing the lists. One of the primary technical contributions of this thesis is the implementation of such an algorithm.

However, the evolution of new ISD algorithms did not cease there. Presently, the two asymptotic fastest algorithms for solving the CSD problem are due to May and Ozerov [MO15] and Both and May [BM17]. These algorithms heavily rely on a subroutine for computing the Nearest Neighbor Problem, allowing them to solve the transformed problem more efficiently than the previous algorithms. The Nearest Neighbor Problem asks to find all vectors within

two lists from $\mathbb{F}_2^n$ that are close to each other according to the Hamming metric. That is, given $\mathcal{L}_1, \mathcal{L}_2 \subset \mathbb{F}_2^n$, find all $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}_1 \times \mathcal{L}_2$, s.t. $\mathrm{wt}(\mathbf{x} + \mathbf{y}) = \omega$.

The use of an algorithm for solving the Nearest Neighbor Problem becomes apparent through the following observation. Assume that after applying the transformation to obtain a reduced CSD problem, the resulting parity check matrix has the form $\mathbf{H}' = (\mathbf{I}_{n-k}\mathbf{H}_2)$, where $\mathbf{I}_{n-k}$ is the identity matrix and $\mathbf{H}_2 \in \mathbb{F}_2^{(n-k)\times k}$ and $\mathbf{e}_1 \in \mathbb{F}_2^{n-k}, \mathbf{e}_2 \in \mathbb{F}_2^k$. Then Equation (1) becomes

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

Note that $\mathbb{E}[\mathrm{wt}(\mathbf{e}_1)] = \omega \cdot \frac{n-k}{n}$. Assuming that the expectation is met and $\omega \ll n$, it follows that $\mathrm{wt}(\mathbf{e}_1) \ll n - k$, meaning the error vector $\mathbf{e}_1$ is *small*. Consequently, $\mathbf{H}_2\mathbf{e_2}$ is equal to $\mathbf{s}$ up to a small amount of errors in $\mathbf{e}_1$. In other words, $\mathbf{H}_2\mathbf{e_2}$ is *close* to the syndrome $\mathbf{s}$. This information is useful for an algorithm computing the lists containing a representation of the solution $\mathbf{e}$. May and Ozerov [MO15] proposed an efficient algorithm for certain parameter regimes for the Nearest Neighbour Problem, but left its practical applicability as an open question. Consequently, another technical contribution of this thesis is a new algorithm for solving the Nearest Neighbour Problem that extends the May-Ozerov algorithm for all parameter regimes. Additionally, a proof of concept implementation is provided.

In 2016, Canto-Torres and Sendrier [CTS16] proved an observation that was made far earlier, which is related to the very low weight of the instances of code-based schemes. They proved that as long as the weight is a linear in n, e.g. $\omega = \Theta(n)$, all ISD algorithms improve asymptotically over the previous one. However, in the case of sublinear error weight, $\omega = o(n)$, as in the case of McEliece, BIKE, or HQC, all improvements of subsequent algorithms are limited to second-order terms. Hence, the runtime of the different ISD algorithms converges to the asymptotic runtime of Prange's original ISD algorithm. This means that all algorithmic improvements in the setting of cryptographic instances are of a practical nature.

As previously mentioned, NIST selected three different code-based schemes as probable next standards. A major challenge lies in selecting secure parameter sets that match the security level set by NIST. Therefore, estimating their bit-security - the expected number of bit operations required to successfully recover the solution - is of utmost importance. Two things are necessary to make such estimations. First, a precise description of the best algorithm for solving the problem at hand—in this case, the CSD. Second, an implementation of this algorithm is needed to realistically model real-world implications, such as slowdowns caused by memory access. All of this is thoroughly provided and discussed in this thesis.

What most ISD algorithms have in common is a significant memory requirement. These immense memory demands either cause a substantial slowdown of the algorithm due to physical memory access times or prevent the algorithm from running altogether. In practical implementations, both scenarios demand a fallback to a more memory-efficient but asymptotically slower algorithm. In such cases, time-memory trade-offs become invaluable, as they allow the memory requirements to be adjusted according to the available resources, while ideally only slightly increasing the runtime.

Consequently, another contribution of this thesis is a new time-memory trade-off for ISD algorithms and their implementation. This implementation is particularly important as it demonstrates that the practical improvements of subsequent ISD algorithms, resulting from the work of Canto-Torres and Sendrier, translate into practical application, even though they rely on complex algorithmic techniques that may introduce overhead. With this implementation, new computational records were achieved, leading to more precise bit-security estimations for the three code-based schemes in round four of the NIST standardization process.

Goal of this work

The objective of the first half of this work is to conduct a thorough evaluation of the security of code-based cryptographic schemes. This is achieved by implementing the most efficient attacks for those code-based schemes. As a result of these implementations, we were able to break several records on `decodingchallenge.org`, a website containing cryptanalytic, thus downsized, versions of the proposed schemes. This, in turn, allowed us to extrapolate the security of the proposed parameter sets. With these extrapolations, we can make statements, whether a given scheme is reaching its claimed level of security or not.

As a secondary aim, new time-memory trade-offs for the generic attacks are proposed, implemented, and evaluated through their application in breaking additional challenges. Initially, we develop these time-memory trade-offs for the subset sum problem, improving the state of the art across most memory regimes. These trade-offs are therefore of independent interest and are likely to enable improved time-memory trade-offs also for other problems beyond the CSD as well.

To condense this, the primary objective of this work is:

Devise a fast and efficient implementation of the asymptotically fastest generic decoding algorithms for random binary codes to extrapolate the exact security of post-quantum code-based cryptographic schemes

and

Develop new Time-Memory Trade-Offs for the Subset Sum problem as well as the computational syndrome decoding problem.

Additionally, this work includes an analysis of the Nearest Neighbour Problem in Hamming metric. As a result, this dissertation comprises the following chapters:

- **Chapter 2:** *McEliece needs a Break–Solving McEliece-1284 and Quasi-Cyclic-2918 with Modern ISD*, co-authored with Andre Esser and Alexander May. This chapter is dedicated to a practical implementation of the MMT/BJMM algorithm. Through this implementation, we were able to demonstrate the practical superiority of advanced ISD algorithms. Moreover, we employed this implementation to extrapolate the bit security of the three code-based participants in the fourth round of the NIST standardization, *McEliece*, *BIKE*, and *HQC*. We conducted these assessments using different memory models to accurately estimate the runtime, even if the algorithms utilize a significant amount of memory.

- **Chapter 3:** *New Time-Memory Trade-Offs for Subset Sum—Improving ISD in Theory and Practice*, co-authored with Andre Esser. This chapter extends on the previous chapter by introducing a novel time-memory trade-off for the Subset Sum problem. This new trade-off unifies the algorithmic landscape of Subset Sum under memory constraints to two algorithms. Moreover, we translate this trade-off into the realm of Syndrome Decoding and apply it to the MMT/BJMM algorithm. Additionally, we implemented this trade-off and employed the same extrapolation methodology as in the previous chapter to calculate the bit-security of code-based schemes.

- **Chapter 4:** *A Faster Algorithm for Finding Closest Pairs in Hamming Metric*, co-authored with Andre Esser and Robert Kübler. In this chapter, a new algorithm for the *Nearest Neighbour Problem* is presented. This extends the previously best algorithm by

May and Ozerov in the regime of large distances. Our proposed algorithm is notably simpler to analyze and can be applied to non-uniform distributions. Furthermore, we provide a proof of concept implementation, which was not possible with the previous algorithm.

- **Chapter 5:** *Legendre PRF (multiple) key attacks and the power of preprocessing*, co-authored with Alexander May. The final chapter of this dissertation is centered around precomputation attacks on the Legendre Pseudorandom Function (PRF). These attacks specifically target scenarios where an attacker precomputes a hint, and upon access to an oracle containing a secret key, the attacker can accelerate the process of retrieving the secret key using the precomputed hint. Three distinct scenarios are highlighted, the first being a precomputation attack on a single key, followed by a precomputation attack on multiple keys. The final scenario involves an attack on multiple keys that does not rely on a precomputation hint but rather utilizes the *distinguished points* technique introduced by van Oorschot and Wiener [vW94].

Security Extrapolation

The challenge of selecting secure parameters for cryptographic schemes is not a new problem the cryptographic community is facing in the NIST selection process. Over the years, for the two most widespread used security foundations, namely factorization of large composite numbers and the Discrete Logarithm, secure parameters have been selected not just once, but several times.

In this context, a methodology for selecting secure parameters has been established. This methodology combines theoretical and practical results and mainly evolves the following three steps: First, a theory is developed to answer the question: what is the best algorithm for solving the problem at hand? For this algorithm, a usually asymptotic runtime formula T_d is derived. This formula provides a scaling of the number of operations required to solve the problem, but it is not directly used to estimate the bit security of the proposed cryptography scheme. This is because purely theoretical estimates do not account for implementation challenges, such as extensive random memory access into large amounts of RAM, which can significantly slow down the algorithm. Therefore, translating these theoretical estimates directly into practical metrics such as CPU years or CPU cycles does not adequately address these issues.

Instead, the typical approach, and thus the second step of the methodology, is to implement the proposed algorithm and perform experiments on smaller parameter sets. Consequently, one extrapolates an experiment in dimension d to a larger scale instance in dimension d', by multiplying the experimentally obtained runtime with the quotient $T_{d'}/T_d$. This is demonstrated in Figure 1.1 on the left side.

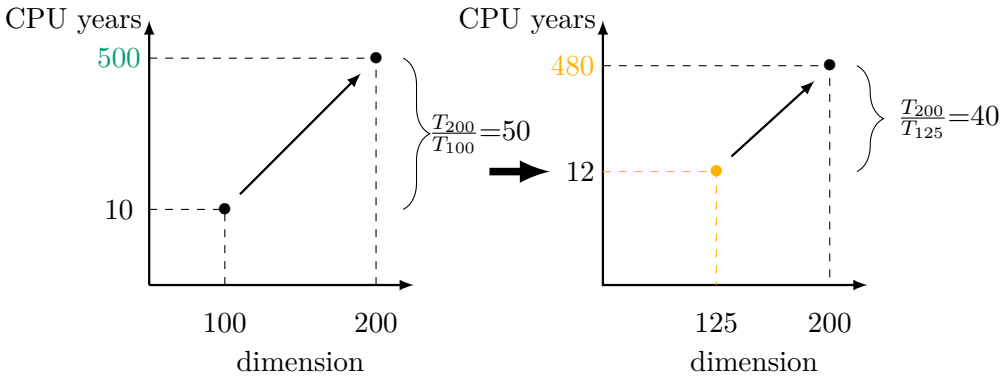


Figure 1.1: One possible way to increase the confidence in an extrapolation is to increase the solved instance size. On the left-hand size, an instance of dimension $d = 100$ is shown. This instance was solved in 10 CPU years, from which an instance of dimension $d' = 200$ is extrapolated, which can be solved in expected 500 CPU years. On the right-hand side, we have an instance of dimension $d'' = 125$ which was solved in 12 CPU years, from which a new (lower) expected runtime (480 CPU years) can be extrapolated.

Of course, such extrapolations are not perfect, they inherit uncertainties. One of these uncertainties is the accurate modeling of memory accesses, as previously discussed. Thus, the question that arises is how to enhance the accuracy of this extrapolation process. The two straightforward approaches are to either increase the size of the experiments or to refine the runtime formula T_d. Increasing the size of the experiments to $d'' > d$ is an effective way of narrowing the gap between the start and end points of the extrapolation. By doing so, the quotient $T_{d'}/T_{d''}$ becomes smaller, and can hence carry less uncertainties, thus improving the reliability of the extrapolated result. The second approach to increasing the accuracy of extrapolation involves a more thorough examination of the theoretical algorithm under more accurate models. For instance, this includes accounting for the memory consumption of the algorithm. This approach is further explored and analyzed in detail in Chapter 3 and Chapter 4.

Before examining the security of code-based schemes, it is useful to explore the historical evolution of RSA and how the extrapolation methodology was applied. Following this, a comparative analysis with the current state-of-the-art in code-based cryptanalysis will be conducted to identify any missing parts of the extrapolation methodology. These parts will then be addressed in the subsequent chapters.

Timeline Comparing RSA with Decoding.

The publication of RSA in 1977 by Rivest, Shamir and Adleman [RSA78], marks the beginning of a new era in cryptography as it is one of the first public-key cryptographic schemes. The concept is as follows: two prime numbers of equivalent bit size p and q are selected, their product $N = p \cdot q$ and $\Phi(N) = (p-1) \cdot (q-1)$ are computed. Next, a public key $e \in \mathbb{Z}^*_{\Phi(N)}$ is chosen and its inverse $d = e^{-1} \mod \Phi(N)$ is calculated. The public key is denoted by the pair (e, N), while the secret key is the triplet (d, p, q). Encryption and decryption are defined as follows:

$$\begin{aligned} \text{Enc}\colon \mathbb{Z}_N &\to \mathbb{Z}_N \quad \text{and} \quad \text{Dec}\colon \mathbb{Z}_N \to \mathbb{Z}_N \\ m &\mapsto m^e, \qquad\qquad c \mapsto c^d. \end{aligned}$$

The hardness is based on the assumption that integer factorization is hard: Given $N = p \cdot q$, computing the prime numbers p and q is hard. Currently, the most practical and efficient method for attacking RSA is by factoring the public modulus N through the *General Number Field Sieve*, which has sub-exponential runtime. Once the secret factorization is known, the secret key d can be easily computed.

The algorithm, known today as *General Number Field Sieve*, by Lenstra, Lenstra, Manasse and Pollard [LL93] was a breakthrough in the development of factorization algorithms. As of today, this is still the asymptotically fastest known integer factorization algorithm. While there have been improvements of the algorithm over the past 30 years, they have been primarily of a practical nature, providing at most a polynomial factor or second-order term improvement.

As a result, it is challenging to judge the significance of such improvements from a pure asymptotic point of view. These improvements often result from more complex routines which may introduce overhead, or they are based upon heuristics that are uncertain in their validity. To accurately assess the impact of these improvements on the security of proposed RSA parameters, it is crucial to consider their implementations and eventually refined extrapolations. Often then these implementations have been able to break challenges and achieve new computational records, providing new data points in the extrapolation methodology of RSA.

Consequently, we can see that after the introduction of the number field sieve, a long stream of approximately 30 years of practical work was published. Up to the work by Shor [Sho97] who published his famous quantum polynomial-time factorization algorithm, which initiated the search for new post-quantum alternatives to RSA.

The picture changes in the setting of code-based cryptography. Starting from 1962 when the first algorithm by Prange [Pra62] was published, long before the first public encryption scheme based upon codes was published by McEliece [McE78] in 1978. This algorithm by Prange laid the foundations for the class of *Information Set Decoding* (see Section 2.2) algorithms, which are still the main tools for estimating the security of code-based schemes.

From that point on a long stream of practical improvements was published, which then were interrupted by the algorithms by Stern [Ste89] and Dumer [Dum91] in the year 1989. These algorithms improved asymptotically over Prange‘s via an enumeration strategy. In subsequent years, the first implementations of Stern's algorithm were carried out, and these were subsequently employed for estimating bit security. These estimates indicated that the parameter set proposed by McEliece did not meet the required bit security level of 64 bits. It wasn't until 2008 that Bernstein, Lange, and Peters [BLP08] published the first record computation for breaking an instance that followed this original parameter set proposed by McEliece. This record computation was achieved using an optimized version of Stern's algorithm.

Following this, several asymptotic improvements to Stern's and Dumer's algorithms were published, which sped up the enumeration step. First, the work by May, Meurer and Thomae [MMT11] leveraged the *Representation Technique* which then was subsequently improved by Becker, Joux, May and Meurer [BJMM12] with a finer analysis. After that the work of May and Ozerov [MO15] improved significantly over this by introducing an efficient *Nearest Neighbour* (see Section 2.4) algorithm. This work was extended by Both and May [BM18] to the currently asymptotically fastest known Information Set Decoding algorithm. This demonstrates the strength of the research on theoretical algorithms in the past 60 years in the field of code-based cryptography.

As already mentioned, Canto-Torres and Sendrier [CTS16] proved that in the case of sublinear error weight, $\omega = o(n)$, as in the case of McEliece, BIKE, or HQC, all improvements of

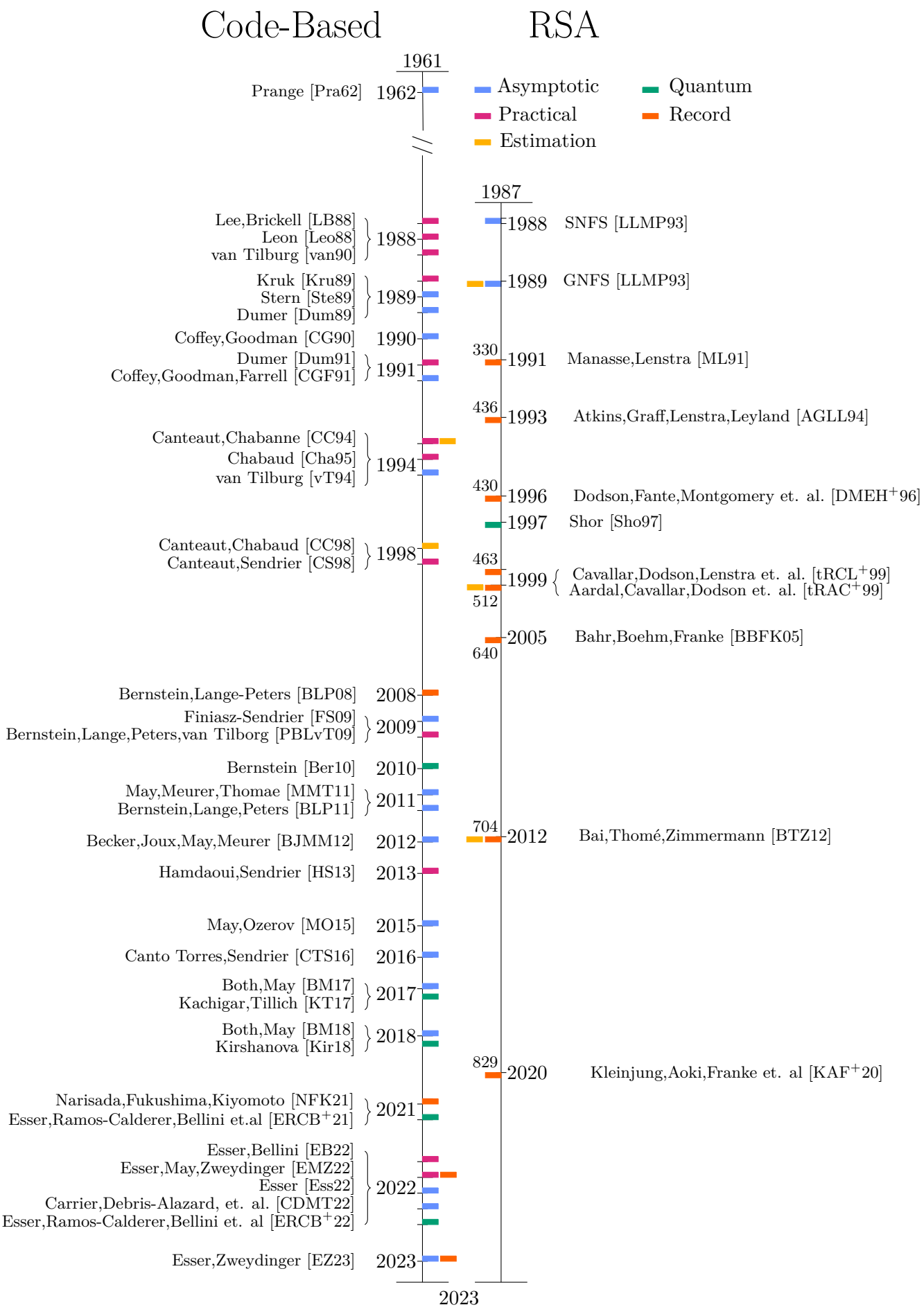


Figure 1.2: Comparison of the history of research in the field of integer factorization vs. code-based cryptography. With the work of Canto-Torres and Sendrier [CTS16], all asymptotic improvements in the code-based setting changed to practical improvements for cryptographic instances. The values displayed above the record computation symbol on the RSA side correspond to the bit size of the challenge that has been successfully broken.

subsequent algorithms are limited to second-order terms. Hence, all algorithmic improvements, in this setting, are of a practical nature. Thus, drawing an analogy to the RSA case, Prange's algorithm can be compared to the number field sieve, serving as a foundation for a long series of practical improvements. However, unlike RSA, there appears to be a lack of implementation work and computational records that confirm the significance of those improvements. One main goal of this thesis is to address this discrepancy.

History of Computational Records of RSA

The RSA Laboratories [Lab91] published in 1991 a list of 50 RSA modules in increasing size, to encourage research into computational number theory and the practical difficulty of factoring integers. Successfully factorizing one of them would grant a price money, ranging from 1000$ to 200000$. The size of the numbers ranges from 330 to 2048 bits.

Until the closing of the challenge in 2007, 11 numbers were factored. Despite the end of the prize money incentive, the scientific community continued their efforts, successfully factoring another 10 bigger instances of up to 829 bits. Table 1.1 presents a list of the most significant challenges that have been solved.

Date	Bits	Authors	CPU years	Time	Algorithm
1991	330	[ML91]	7[1]	-[0]	ppmpqs [Mis13]
1994	426	[AGLL94]	5000[1]	-[0]	ppmpqs
1996	430	[DMEH+96]	500	19d	GNFS[a]
1999	463	[tRCL+99]	15.8	2.5m	GNFS[b]
1999	512	[tRAC+99]	35.7	5.2m	GNFS[b]
2005	640	[BBFK05]	33.9	5m	GNFS
2012	704	[BTZ12]	512	1.2y	CADO-NFS [Tea]
2009	768	[KAF+10]	2000	2.5y	CADO-NFS [Tea]
2020	829	[KAF+20]	2700	-[0]	CADO-NFS [Tea]

Table 1.1: The most important integer factorization records.
[0] neither the hardware or the wall clock was specified in the record data,
[1] MIPS based,
[a] with lattice sieving, block Lanczos [Mon95] and improved square root algorithms,
[b] with a new polynomial selection method developed by Murphy and Montgomery [Mur99], with new line sieving

While examining Table 1.1, it may be mistakenly perceived that only three distinct implementations were utilized. However, it should be noted that these implementations underwent continuous development over time, with certain subroutines being replaced with more advanced and efficient ones, as indicated in the table's accompanying footnotes. The first publicly known implementation used for record computations is pppmqs, based on the double large primes procedure variation of the multiple polynomial quadratic sieve. Following the development of the general number field sieve in 1989, the initial implementation was introduced by Bernstein and Lenstra in 1991 [BL06]. However, it was not until five years later that the first record was successfully broken using the number field sieve [DMEH+96]. The culmination of implementation efforts resulted in the release of CADO-NFS in 2006 [Tea], which has since become the leading implementation of the general number field sieve.

The first two records do need further explanation. As these computations were performed prior to the establishment of public record databases, the sole source of information regarding these records is a private email exchange originating from RSA Laboratories, lacking details

regarding the specific hardware utilized and the precise duration of time taken. Furthermore, due to the significant advancements in CPU performance since that time, a direct comparison between modern CPUs and older MIPS CPUs used in these record computations is not meaningful. As a result, the MIPS-based records have been excluded from subsequent comparisons.

History of Computational Records in Code-Based Cryptography

This chapter presents the publicly available computational records for code-based cryptography. In 2019, the release of the `decodingchallenge.org` website [ALL19] initiated a pursuit to attain the highest records in various code-based categories. Notably, the website offers two categories relevant to this work. First Goppa-Code based challenges on which McEliece [McE78, BCL+17] is based on and secondly Quasi-Cyclic codes on which the schemes BIKE [ABB+17] and HQC [MAB+18] are based on.

Most records previous to this work are either made with the implementation by Landais [Lan], or by Vasseur [Vas], both implementing the algorithm by Dumer [Dum91], one of the early improvements of Prange's original ISD algorithm. Vasseur's implementation is extending the original Dumer algorithm, to handle multiple syndromes, which is especially important in the Quasi-Cyclic setting. The notable exception in the Table 1.2 of computational records, is the implementation by Narisada, Fukushima and Kiyomoto [NFK22] which is a GPU implementation of the Dumer's algorithm.

Date	n	Bit Security	Authors	CPU Time	Time	Algorithm
2019	640	32	[Lan]	0.09h	0.01h	Dumer
2020	923	45	[Vas]	3.3d	10h	Dumer
2020	1041	45	[NFK22]	-	108h	Dumer GPU
2021	1223	54	[EMZ22]	1254.4d	2.5d	MMT/BJMM
2021	1284	56	[EMZ22]	8047d	37.5d	MMT/BJMM

Table 1.2: The most important record computations in the McEliece setting from `https://decodingchallenge.org/goppa/`. The column *Bit Security* states the estimated bit security of the given instance. The $n = 1041$ record was broken with GPUs, thus no valid CPU time can be stated.

As the record by Narisada, Fukushima and Kiyomoto [NFK] was made using GPUs and only the wall time was stated by the authors, a CPU runtime cannot be computed. A detailed theoretic description of the used algorithm for the records is given in Section 2.2 in the case of Dumer and in Section 3.2 and Section 4.5.1 in the case of MMT/BJMM.

The computational records for the quasi-cyclic setting are given in Table 1.3. Understanding the table presented requires familiarity with the term *DOOM*, which stands for *Decoding One Out of Many*. It was first introduced by [Sen11]. Informally speaking, it's a method that utilizes N instances to speed up the computation of only one of those instances by a factor of $\sqrt{N}$. Therefore, using the fact that quasi-cyclic codes have a structure that holds a linear amount of cyclic equivalent solutions for each challenge, an attacker can find one of these equivalent solutions to recover the original solution.

Using the implementation provided with this work we were able to obtain new computational records as can be seen in Table 1.3. Those records contribute substantially to a better security understanding of code-based cryptographic schemes and are one of the major contributions of this thesis. The following Chapter 3 and Chapter 4 provide an in-depth

Date	n	Bit Security	Authors	CPU Time	Time	Algorithm
2019	1158	34	[Vas]	0.31d	0.2h	Dumer
2019	1766	42	[Vas]	47.9d	31.9h	Dumer + DOOM
2021	2118	46	[EMZ22]	22.4d	1.05h	MMT/BJMM + DOOM
2021	2918	54	[EMZ22]	1719d	80h	MMT/BJMM + DOOM
2022	3138	56	[EZ23]	817d	38.31h	mem. opt. MMT/BJMM + DOOM

Table 1.3: The most important record computations in the quasi-cyclic setting from `https://decodingchallenge.org/q-c`. The runtime is stated in wall time. The column *Bit Security* states the estimated bit security of the given instance.

analysis of how those records were achieved. Especially Chapter 3 is about the algorithm solving the challenges $n = 2118, n = 2918$, whereas Chapter 4 describes the algorithmic improvements made to get the last record $n = 3138$.

Comparing just the accumulated CPU years, roughly 5250 in the case of RSA, and 32.6 for the code-based cryptography (only counting the McEliece and Quasi-Cyclic setting) shows the tremendous backlog of practical record computations for code-based schemes.

Extension to non-binary codes The main instances solved in the context of this work are defined over a binary field. However, the CSD defined over non-binary fields has shown to become increasingly relevant, especially with respect to recent signature constructions [BMPS20, DST19, AFG$^+$23]. We therefore provide our implementation in full generality, capable of handling arbitrary non-binary linear codes. To demonstrate the efficiency of this implementation we obtained the five largest records for the ternary syndrome decoding challenges, following parameters found in WAVE[1].

[1] See `https://decodingchallenge.org/large-weight`

2 Preliminaries

In this chapter, we will introduce the general notation and provide basic definitions and results from coding theory. Additionally, an introduction to the Information Set Decoding algorithms will be presented.

2.1 Notation

Throughout this thesis, we denote with $\mathbb{F}_q$ the finite field with q elements. We are mostly interested in the *binary* case, e.g. $q = 2$. Following this notation, we describe with $\mathbb{F}_2^n$ the corresponding n-dimensional vector space, which elements will be denoted with a bold lowercase letter $\mathbf{v} \in \mathbb{F}_2^n$. Additionally, matrices are denoted as bold capital letters: $\mathbf{G} \in \mathbb{F}_2^{n \times k}$, and with $\mathbf{G}^\intercal \in \mathbb{F}_2^{k \times n}$ its transposed. Furthermore, we denote by $\mathrm{wt}(x) := |\{i \in [n] \,|\, x_i \neq 0\}|$ the *Hamming Weight* of the vector $\mathbf{x}$, where $[n] = \{1, \ldots, n\}$. Often we are also interested in comparing only parts of a vector with each other, therefore we denote with $\pi_\ell(\mathbf{x}) = \{\mathbf{x}_{n-\ell}, \ldots, \mathbf{x}_n\} \in \mathbb{F}_2^\ell$ the projection onto the last ℓ coordinates of $\mathbf{x} \in \mathbb{F}_2^n$. As we often need to enumerate binary sets with a specific hamming weight we denote $\mathcal{W}_\omega^n = \{\mathbf{x} \in \mathbb{F}_2^n \,|\, \mathrm{wt}(\mathbf{x}) = \omega\}$ as the set of vectors of hamming weight ω. Finally, all logarithms are to the base 2. In addition to that, we describe with $x \in_R D$ the random choice of an element of the distribution/set D. If D is a set we always sample following the uniform distribution.

Algorithms: For a deterministic algorithm $\mathtt{A}$, we write $y := \mathtt{A}(x)$ if y is the output of $\mathtt{A}$ with input x. To indicate a probabilistic Algorithm $\mathtt{A}$, we write $y \leftarrow \mathtt{A}(x)$ instead.

The Binary Entropy Function is defined as

$$H\colon [0,1] \to [0,1]$$
$$x \mapsto \begin{cases} -x\log(x) - (1-x)\log(1-x) & x \in (0,1) \\ 0 & \text{else} \end{cases}$$

Note that the binary entropy function is therefore symmetric around $x = \frac{1}{2}$. Additionally, we define the inverse function of H by setting the value of $H^{-1}(y)$ to the unique $x \in [0, \frac{1}{2}]$, which solves $H(x) = y$.

Stirling's Formula To be able to better analyse the asymptotic behavior of an algorithm, we make heavy use of the following theorem:

$$\begin{aligned} \log n! &= (n - \frac{1}{2})\log n - n + \frac{1}{2}\log(2\pi) + o(1) \\ &= n\log n - n + \mathcal{O}(\log n), \text{ for } n \to \infty \end{aligned}$$

Using this formula we are able to approximate the binomial coefficient as

$$1/n \log \binom{n}{\alpha n} = H(\alpha), \text{ for } n \to \infty$$

for proofs we refer to [Meu12], chapter 2.3.

CPU Years and other type of measurements In this work, we state runtime in the unit of *CPU years* or *core years*. We use these terms interchangeably, as a *CPU year* is equivalent to a *core year*. Consequently, it should be noted that a modern CPU typically consists of multiple cores that can perform computations in parallel. For example, an 8-core CPU can perform 8 *core years* of computation in a single *wall year*, where *wall year* refers to the actual time elapsed.

While this unit of measurement provides a precise way to predict the runtime of an algorithm irrespective of the number of CPUs in use, it also introduces inaccuracies. These inaccuracies stem from the rapid development of CPUs over time. CPUs manufactured in 2023 are significantly more powerful than those produced in 2013, not just in terms of the number of cores, but also in clock speed and instructions per cycle. Furthermore, the Instruction Set Architecture (ISA) is continuously evolving. Modern CPUs now have up to 512-bit-wide instructions and registers, which enable even higher throughput of operations.

Consequently, to enable a fair comparison of different implementations, it is necessary to run them on the same hardware. Thus, all our benchmark and stated CPU years were performed on an AMD EPYC 7742 with 2TB of RAM.

2.2 Code-Based Cryptography

As previously stated, there is an urgent need to replace current encryption schemes with new constructions based on other hard problems on lattices, codes, or multivariate polynomial equations. The objective of this short introduction is to demonstrate how a one-way trapdoor function can be constructed using hard problems from coding theory. Specifically, the Computational Syndrome Decoding Problem (SDP) is the fundamental computational problem in code-based cryptography.

In the following, we assume that the reader has some familiarity with the basic definitions and concepts of binary linear codes, for reference see [PBH98, VL98].

Linear Codes The concept of adding redundant information to data that will be transmitted over a noisy channel dates back to Shannon [Sha48]. This process is known as *encoding*, while the process of removing redundant data and potentially correcting errors is referred to as *decoding*. In the following, we will denote the application of these processes as *Encode* and *Decode*, respectively.

Over the years, numerous codes have been developed that possess distinct properties or efficiency. Codes with common structures are grouped, and the simplest but most useful is likely the family of linear codes, in particular the binary linear codes.

Definition 1. (Binary) Linear Codes. Let $\mathcal{C} \subset \mathbb{F}_2^n$ be a subspace of dimension k. We call $\mathcal{C}$ an $[n, k]$ code and $R := \frac{k}{n}$ is called (code) rate of $\mathcal{C}$.

One usually identifies a (binary) linear code $\mathcal{C}$ by the row-space of a $k \times n$ *generator matrix* $\mathbf{G}$, e.g. $C = \{\mathbf{m}^\intercal \mathbf{G} \mid \mathbf{m} \in \mathbb{F}_2^k\}$ or as the kernel of an $(n-k) \times n$ *parity check matrix* $\mathbf{H}$, e.g.

$C = \{\mathbf{c} \in \mathbb{F}_2^n \mid \mathbf{Hc} = \mathbf{0}\}$. Note that we explicitly defined only binary codes, as they are of particular interest in this work. Furthermore, the following nomenclature is used throughout this work: a codeword is denoted by $\mathbf{c}$, an error vector by $\mathbf{e}$, an erroneous codeword by $\mathbf{x} = \mathbf{c} + \mathbf{e}$ and a syndrome by $\mathbf{s} = \mathbf{Hx} = \mathbf{H}(\mathbf{c} + \mathbf{e}) = \mathbf{He}$.

Consequently, we introduce *Minimum decoding distance*, an important property that every code possesses.

Definition 2. Minimum decoding distance d of a linear $[n, k]$ code $\mathcal{C}$ is defined as the minimum hamming distance of every two distinct codewords:

$$d := d(\mathcal{C}) = \min_{\mathbf{c} \in \mathcal{C} \setminus \{\mathbf{0}\}} \mathrm{wt}(\mathbf{c}).$$

Note that $\omega := \lfloor \frac{d-1}{2} \rfloor$ is the largest distance where decoding is still uniquely possible. A lower bound of the minimum decoding distance is given by Gilbert and Varshamov.

The (Relative) Gilbert-Varshamov Distance $d_{GV}(R) \in \mathbb{R}$, given a code rate $R \in (0, 1)$, is the unique solution of the equation

$$H(x) = 1 - R.$$

with $0 \leq x \leq \frac{1}{2}$.

The usefulness of this bound is supported by the following theorem:

Theorem 1. *For almost all linear codes $\mathcal{C}$ of rate R, it holds true that*

$$d(\mathcal{C}) \geq \lfloor D_{GV}(R) \rfloor \cdot n$$

The proof is given in [Meu12] Theorem 2.4.9. These properties of linear codes lead to the problem that lies at the center of most code-based cryptographic schemes:

Definition 3. (Binary) Computational Syndrome Decoding Problem (SDP): Given a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$, a syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$ and a target weight $\omega \in \mathbb{N}$, find an (error) vector $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathbf{He} = \mathbf{s}$ and $\mathrm{wt}(\mathbf{e}) \leq \omega$.

Note that we again explicitly only state the binary case of the problem. This problem is particularly intriguing due to its well-known NP-hardness, as demonstrated by Berlekamp [BMVT78], and its resilience against algorithms utilizing quantum computers. Naturally, this leads to the question, for which (n, k, ω) the SDP is hard, or better, the hardest, such that one can use such parameters as the foundation of a code-based scheme. In the asymptotic setting, it is often assumed heuristically that the SDP is the hardest if $\omega = D_{GV}(R)$. See [Meu12] chapter 3.1 for justification.

The usefulness of the Gilbert-Varshamov Distance comes from the following consideration. We expect the parity check equation $\mathbf{He} = \mathbf{s}$ to be solvable if the search-space of $\mathbf{e}$ is at least of size 2^{n-k}. Thus, for weight-d error vectors $\mathbf{e} \in \mathcal{W}_d^n$ we have $\mathcal{O}(|\mathcal{W}_d^n|) = \mathcal{O}(2^{H(d)n}) = \mathcal{O}(2^{n-k})$. Whereby the last equation follows from the Gilbert-Varshamov Distance. Therefore, this guarantees, together with the well-known fact that random linear codes asymptotically reach the Gilbert-Varshamov Distance, that we can always expect a solution to the SDP. Thus, we refer to this case of $\omega = D_{GV}(R)$ as *full distance decoding* and we refer to the case $\omega = \lfloor \frac{D_{GV}(R)-1}{2} \rfloor$ as the *half distance decoding* case, the latter being the largest case in which decoding is still uniquely possible.

Code-Based One-Way Function For our purpose, a sufficient definition of a *one-way function* is as follows. Let $f : X \mapsto Y$, we call f a one-way function if it is efficiently computable $\forall x \in X$ and for almost all $y \in Y$ it is computationally infeasible to find a preimage x.

There are essentially two dual ways to build a one-way function based on the hardness of the SDP. The first, due to the Robert McEliece [McE78], is fully described by the generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$. It works by scrambling a codeword $\mathbf{c}$ with an error vector $\mathbf{e}$ to an erroneous codeword $\mathbf{x} = \mathbf{c} + \mathbf{e}$.

$$f_{\mathbf{G}} \colon \mathbb{F}_2^k \times \mathcal{W}_\omega^n \to \mathbb{F}_2^n \qquad \text{and} \qquad f_{\mathbf{H}} \colon \mathcal{W}_\omega^n \to \mathbb{F}_2^{n-k}$$
$$(\mathbf{m}, \mathbf{e}) \mapsto \mathbf{m}^\intercal \mathbf{G} + \mathbf{e}^\intercal, \qquad \mathbf{e} \mapsto \mathbf{H}\mathbf{e}.$$

The latter method by Niederreiter [Nie86] is fully described by a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ and works by mapping an error $\mathbf{e}$ to the syndrome $\mathbf{s} = \mathbf{H}\mathbf{e}$. It should be noted that a one-way function can be obtained directly by instantiating one of the two constructions defined, through the selection of a random linear code $\mathcal{C}$. Throughout this work, we will use the Niederreiter approach for describing the problem to be attacked, as it has wider adaption due to its efficiency.

In addition to the normal one-way function, a *trapdoor one-way function* allows for efficient inversion by using some auxiliary *trapdoor* information. Hence, one has a public description of $f \colon X \mapsto Y$ and a secret description of f^{-1} such that one can efficiently compute $f^{-1}(f(x)) = x, \forall x \in X$. Consequently, one cannot use a random linear code anymore to instantiate the trapdoor one-way function but needs to use a structured one. The idea is then to *hide* the structure of the chosen code via linear algebra. Thus, choosing a random permutation $\mathbf{P} \in \mathbb{F}_2^{n \times n}$ and a basis transformation $\mathbf{T} \in \mathbb{F}_2^{(n-k) \times (n-k)}$ and applying them to the parity check matrix $\mathbf{H}$ to compute a public matrix $\mathbf{H}' = \mathbf{THP}$. The trapdoor then consists of the construction of $\mathbf{H}$ as well as the transformation matrices $\mathbf{T}$ and $\mathbf{P}$.

The (Classic) McEliece trapdoor one-way function is based on binary Goppa codes. However, this additional structure opens the door for further (structural) attacks. History has shown that replacing Goppa codes with alternative codes often leads to successful attacks. For example Generalized Reed-Solomon codes were broken by Sidelnikov and Shestakov [SS92], and Reed-Muller codes were broken by Minder and Shokrollahi [MS07].

So far, the family of binary Goppa codes resists all known structural attacks, which leads to the following assumption.

Assumption. *A randomly transformed parity check matrix $\mathbf{H}'$ of a binary $[n, k]$ Goppa code is computationally indistinguishable from a random parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$.*

Therefore, the only available approach to attack code-based encryption schemes relying on the Niederreiter one-way function construction is generic decoding.

Generic Decoding. All decoding algorithms solving the *Syndrome Decoding Problem* have a runtime $T(n, k, d)$ as a function in three parameters n, k, d. Given any random linear code, we can use the Gilbert-Varshamov distance from Section 2.2, which shows that d itself is a function in n, k. This simplifies the runtime formula to $T(n, k)$. Furthermore, we mostly consider worst-case runtimes $T(n)$ over all code rates $\frac{k}{n}$. As of today, *Information Set Decoding* algorithms are the most efficient way to decode random linear codes. However, these run in exponential time of the form $T(n) = 2^{\vartheta n}$, where ϑ is a constant. In literature, this coefficient is used as a metric to compare different algorithms asymptotically.

Information Set In the previous section, we mentioned that *Information Set Decoding* is currently considered to be the most efficient way to decode random binary linear codes. To better understand this, we need the following definitions.

Definition 4. Information Set (Generator Matrix): Given a linear $[n, k]$ code $\mathcal{C}$ with generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$, an index-set $\mathcal{I} \subset [n]$ of size k is called an *Information Set* if the sub-matrix $\mathbf{G}' \in \mathbb{F}_2^{k \times k}$, that arises from the columns of $\mathbf{G}$ selected by $\mathcal{I}$ has full rank, i.e. $\text{rank}(\mathbf{G}') = k$.

If $\mathbf{G}'$ has full rank, it means that its column vectors span the k-dimensional vector space $\mathcal{C}$ and can be used to represent any non-redundant linear operation on any message $\mathbf{m}$. Consequently, the $\mathcal{I}$ indexed entries are of an error-free codeword $\mathbf{c}$ and are sufficient to recover the message $\mathbf{m}$. Hence, correcting linear codes and therefore retrieving the original message can be described as the problem of finding an error free information set. Adapting this to the parity check matrix yields the following.

Definition 5. Information Set (Parity Check Matrix): Given a linear $[n, k]$ code $\mathcal{C}$ with parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$, an index-set $\mathcal{I} \subset [n]$ of size k is called *Information Set* if the sub matrix $\mathbf{H}' \in \mathbb{F}_2^{(n-k) \times (n-k)}$, emerged from the columns of $\mathbf{H}$ selected by $[n] \setminus \mathcal{I}$ has full rank, i.e. $\text{rank}(\mathbf{H}') = n - k$.

The fundamental concept behind the class of algorithms known as *information set decoding* is to randomly select an information set $\mathcal{I}$ and then attempt to guess or calculate the errors in this particular set of coordinates using a predetermined method. The specific method used distinguishes the various algorithms.

Information Set Decoding

In this section, we introduce the most important algorithms used to attack code-based cryptography. The goal of this chapter is to provide an overview of the state-of-the-art cryptanalysis techniques used in code-based cryptography, in particular, applications in practical cryptanalysis. This is necessary background information for Chapter 3 and Chapter 4.

We will assume that we are given an instance of the Syndrome Decoding Problem (SDP), which consists of a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$, a syndrome $\mathbf{s}$, and a weight ω for all subsequent algorithms. Our objective is to find the error vector $\mathbf{e}$ that corresponds to this instance.

The Brute Force Attack is the simplest of all, but never the less important algorithm. The idea is to consider all vectors $\mathbf{e} \in W_\omega^n$, which takes $\mathcal{O}(|W_\omega^n|) = \mathcal{O}(\binom{n}{\omega}) = \mathcal{O}(2^{H(w)n})$ computations. However, this algorithm is crucial as it is often used as a subroutine in subsequent algorithms to solve some kind of smaller *Syndrome Decoding Problem*. Although the algorithm itself is simple, it is important to pay attention to details, such as the method used to compute $\mathbf{He}$. A naive matrix-vector multiplication can take up to n^2 operations, whereas in Section 2.2 a method is shown to reduce this to ω operations. Lastly, if the weight ω is unknown, one can enumerate every $\omega \in \{1, \ldots, n\}$. This increases the total runtime at most by a factor of n.

The Meet in the Middle (MitM) is the second algorithm we will consider, which represents the first time-memory trade-off that can be utilized. Rather than enumerating the whole search space, it is split into two halves. First, create a list that can be efficiently

searched and contains pairs $(\mathbf{He}_1, \mathbf{e}_1)$, where $\mathbf{e_1} \in \mathcal{W}_{\omega/2}^{n/2} \times \mathbf{0}^{n/2}$ enumerates the weight $\omega/2$ on the first $n/2$ coordinates. In a subsequent step enumerate $\mathbf{He}_2 + \mathbf{s}$, with $\mathbf{e}_2 \in \mathbf{0}^{n/2} \times \mathcal{W}_{\omega/2}^{n/2}$, and attempt to find a collision within the list. Consequently, a collision leads to the solution

$$\begin{aligned} \mathbf{He}_1 &= \mathbf{He}_2 + \mathbf{s} \\ \mathbf{s} &= \mathbf{He}_1 + \mathbf{He}_2 \\ \mathbf{s} &= \mathbf{H}(\mathbf{e}_1 + \mathbf{e}_2) = \mathbf{He} \end{aligned}$$

Algorithm 1: Meet-in-the-Middle

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, \omega$
Output: $\mathbf{e} \in \mathbb{F}_2^n$ st. $\mathbf{He} = \mathbf{s}$ and $\mathrm{wt}(\mathbf{e}) \leq \omega$

```
1 S ← ∅
2 while S is empty do
3     choose random permutation matrix P
4     Set H_1 = PH
5     Compute L = {(H_1 e_1, e_1) | e_1 ∈ W_{ω/2}^{n/2} × 0^{n/2}}
6     for e_2 ∈ 0^{n/2} × W_{ω/2}^{n/2} do
7         if (H_1 e_2 + s, e_1) ∈ L then
8             S ← S ∪ {P^{-1}(e_1 + e_2)}
9 return S
```

Note that this algorithm succeeds with probability $\binom{n/2}{\omega/2}^2 \binom{n}{w}^{-1}$ because the weight has to distributed equally over the two enumeration halves. Due to this, an additional random column permutation $\mathbf{P}$ needs to be applied to the parity check matrix.

Furthermore, it is crucial to highlight that this algorithm is frequently utilized as a subroutine in the subsequent algorithms, where a smaller instance of the Syndrome Decoding Problem must be solved, which is expected to yield not just one, but multiple solutions. Therefore, the pseudocode is formulated to return all solutions. The runtime and space complexity is $\mathcal{O}(\mathcal{W}_{\omega/2}^{n/2}) = \mathcal{O}(\binom{n/2}{\omega/2}) = \mathcal{O}(2^{H(w)n/2})$.

Equipped with these two basic algorithms we can introduce the class of Information Set Decoding algorithms.

Prange The class of Information Set Decoding (ISD) algorithms was introduced in 1962 by Prange [Pra62]. Before presenting the algorithm specifics, we provide an overarching framework for all ISD algorithms in Algorithm 2.

All ISD algorithms proceed in a two-step manner. First, the provided parity check matrix is randomly permuted and afterward systematized via a Gaussian elimination, see line 3. In the second step, the algorithm tries to find an information set, which may yield a solution to the problem. If no such set is found, the process is repeated with a new randomly chosen permutation.

Note that after applying the random permutation the resulting matrix does not necessarily have full rank, resulting in a possible failure of the Gaussian elimination. In theory, we restart the algorithm by choosing a new random permutation, whereas in practice we permute a

Algorithm 2: General Framework of all Information Set Decoding Algorithms

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, \omega$
Output: $\mathbf{e} \in \mathbb{F}_2^n$ st. $\mathbf{He} = \mathbf{s}$ and $\mathrm{wt}(\mathbf{e}) \leq \omega$

1 choose p
2 **repeat**
3 choose random permutation matrix $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ ▷ Permutation Step
4 $\bar{\mathbf{H}} = \begin{pmatrix} \mathbf{I}_{n-k} & \mathbf{H}_1 \end{pmatrix} = \mathbf{GHP}$ in systematic form
5 $\mathbf{s}_1 = \mathbf{Gs}$
6 $L \leftarrow \texttt{Search}(\mathbf{H}_1, \mathbf{s}_1, p)$ ▷ Search Step
7 **for** $\mathbf{e}_2 \in L$ **do**
8 $\mathbf{e}_1 = \mathbf{H}_1\mathbf{e}_2 + \mathbf{s}_1$
9 **if** $\mathrm{wt}(\mathbf{e}_1) = \omega - p$ **then**
10 **return** $\mathbf{P}^{-1}(\mathbf{e}_1\, \mathbf{e}_2)$

previously unused column into the current position of the elimination step to archieve full rank. In the following, we also refer to this step as *Permutation Step*. This step is needed to permute the weight of the error vector $\mathbf{e}$ in such a way that it is useful for the underlying ISD algorithm. Our goal is to have weight p (or p erroneous positions) in the last k coordinates of $\mathbf{e}$, whereas the remaining weight of $\omega - p$ is positioned in the first $n - k$ coordinates. We refer to such a permutation as a *good* one. If this happens, we can search for partial solutions in the following step.

The second step is a *Search* phase, in which a smaller sub *Syndrome Decoding Problem* must be solved, where the algorithm finds all $\mathbf{e}_1 \in \mathcal{W}_p^k$ s.t. $\mathbf{H}_1\mathbf{e}_1 = \mathbf{s}_1$. Note that the (sub-) parity check matrix $\mathbf{H}_1 \in \mathbb{F}_2^{(n-k)\times k}$ is now much smaller and therefore many $\mathbf{e}_1$ fulfill the constraint. The various ISD algorithms differ in how they solve this step. In the following, we also refer to this step as *Search Step*.

Correctness First, one needs to see that a solution $\mathbf{e}$ found for a permuted parity check matrix $\mathbf{HP}$ holds a valid solution if one applies the inverse permutation, as in line 10, i. e. it holds $(\mathbf{HP}) \cdot (\mathbf{P}^{-1}\mathbf{e}) = \mathbf{He} = \mathbf{s}$. Secondly, as the Gaussian elimination, denoted by $\mathbf{G}$, only applies elementary row operations, the resulting parity check matrix $\bar{\mathbf{H}}$ is still a valid one for the given code. Note that the syndrome must be changed as well.

Runtime The runtime is a result of the combination of the two steps above. First, the expected numbers of random permutation needed to hit a *correct* one, which is $\mathcal{P} = \binom{n-k}{\omega-p}\binom{k}{p}\binom{n}{\omega}^{-1}$. Second, we assume that the search phase requires a runtime T_S. Hence, the expected runtime of every ISD algorithm can be described as

$$T = \mathcal{P}^{-1} \cdot T_s.$$

The precise runtime T_S depends on the *Search* subroutine used.

Prange's idea was to use the brute-force algorithm in the search step, thus enumerating every error vector $\mathbf{e}_1 \in \mathcal{W}_p^k$ [1] and hoping for a solution $\mathbf{H}_1\mathbf{e}_1 = \mathbf{s}_1$. If no such solution is

[1] At this point we ignore the additional runtime introduced by the matrix-vector multiplication and only state the runtime as the size of the search space. Note that there are more efficient ways to enumerate all e_1 than by simple matrix-vector operations, see Section 2.2

found, the algorithm restarts. The runtime of the *Search* subroutine is therefore $\mathcal{O}(\binom{k}{p})$. Thus, the overall runtime can be denoted by $\mathcal{O}\left(\frac{\binom{n}{\omega}\binom{k}{p}}{\binom{n-k}{\omega-p}\binom{k}{p}}\right) = \mathcal{O}\left(\frac{\binom{n}{\omega}}{\binom{n-k}{\omega-p}}\right)$. Numerical optimization of p yields $p = 0$, meaning nothing is enumerated. To put it another way, Prange's algorithm is designed to solve the case where there are no errors in the information set.

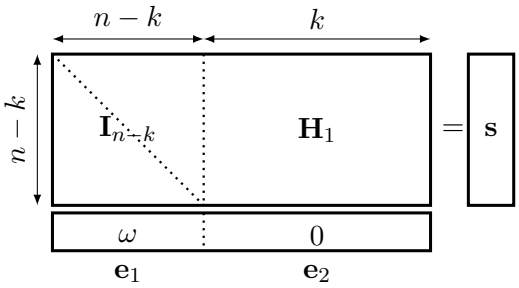


Figure 2.1: Schematic view of Prange's ISD algorithm, in which all the ω error positions are permuted into e_1.

Therefore, Prange's algorithm 2.1 finds solutions of the form

$$\mathbf{He} = \mathbf{I}_{n-k}\mathbf{e}_1 + \mathbf{H}_1\mathbf{e}_2 = \mathbf{e}_1 = \mathbf{s}. \tag{2.1}$$

Finally, we can check for a correct permutation if the resulting syndrome $\mathbf{s}_1$ has low weight, i.e. $\mathrm{wt}(\mathbf{s}_1) = \omega$. Note that this is only possible because we assume $\omega \ll n - k$, as it is given for codes with sub-linear weight.

Algorithm 3: Prange

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, \omega$
Output : $\mathbf{e} \in \mathbb{F}_2^n$ st. $\mathbf{He} = \mathbf{s}$ and $\mathrm{wt}(\mathbf{e}) \leq \omega$

repeat
  choose random permutation matrix $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ ▷ Permutation Step
  $\bar{\mathbf{H}} = \begin{pmatrix}\mathbf{I}_{n-k} & \mathbf{H}_1\end{pmatrix} = \mathbf{GHP}$ in systematic form
  $\mathbf{s}_1 = \mathbf{Gs}$
  if $\mathrm{wt}(\mathbf{s}_1) = \omega$ **then**
    return $\mathbf{e} = P^{-1}(\mathbf{s}_1\, \mathbf{0}^k)$ ▷ Search Step

Note the following things:

- Even though we represent the permutation matrix P in all our pseudocodes as a matrix, for simplicity and readability, it is sufficient to store the permutation in a vector. This reduces the memory from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.
- Again, for simplicity alone, we represent the systematization step via a matrix $\mathbf{G}$, which can be computed in $\mathcal{O}(n^3)$. There are faster ways to achieve this, see [BLP08, Hob12, AB21] for more details. In summary, these optimizations reduce the runtime of elimination from $\mathcal{O}(n^3)$ to $\mathcal{O}(\frac{(n-k)^3}{\log(n-k)})$ via the Method of the four Russians [Bar07, AB21] and clever pivoting. In the following chapters we refer to the runtime of the elimination step as $T_{\mathbf{G}}$.

In summary, the algorithm by Prange has runtime $T_{Prange} = \mathcal{O}\left(\frac{\binom{n}{\omega}}{\binom{n-k}{\omega-p}} \cdot T_{\mathbf{G}}\right)$ and a memory complexity of $\mathcal{O}(n^2)$. While this only reflects the asymptotic runtime, an implementation

of the algorithm is dominated by the runtime of the Gaussian elimination. Thus, many subsequent developments of ISD algorithms are focused on how to reduce the cost of this step [LB88, Leo88, CC94, Cha95, CC98, BLP08].

Lee-Brickell Up to this point, we analyzed Prange's algorithm asymptotically, ignoring the runtime of the Gaussian elimination, which is the dominating factor of an implementation. The idea of the next algorithm by Lee and Brickell [LB88] is to relax the constraint $p = 0$, thus enumerating a little weight, to amortize the cost of the Gaussian elimination, even though this is not optimal asymptotically. Consequently, error vectors with the weight distribution of $\omega - p$ in the first $n - k$ and weight p in the last k coordinates, thus allowing for an erroneous information set, can be recovered. Therefore, the algorithm finds solutions of the form:

$$\begin{aligned} \mathbf{s} &= \mathbf{I}_{n-k}\mathbf{e_1} + \mathbf{H}_1\mathbf{e}_2 \\ \Leftrightarrow \mathbf{e}_1 &= \mathbf{H}_1\mathbf{e}_2 + \mathbf{s} \end{aligned}$$

As in the algorithm by Prange, such a solution can be detected efficiently by checking for

$$\mathrm{wt}(\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}_1) = \omega - p.$$

Algorithm 4: Lee Brickell

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, \omega$
Output : $\mathbf{e} \in \mathbb{F}_2^n$ st. $\mathbf{He} = \mathbf{s}$

1 Choose p
2 **repeat**
3 choose random permutation matrix $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ ▷ Permutation Step
4 $\begin{pmatrix} I_{n-k} & \mathbf{H}_1 \end{pmatrix} = \mathbf{GHP}^{-1}$ in systematic form
5 $\mathbf{s}_1 = \mathbf{Gs}$
6 **for** $\mathbf{e}_1 \in \mathcal{W}_p^k$ **do**
7 **if** $\mathrm{wt}(\mathbf{H}_1\mathbf{e}_1 + \mathbf{s}_1) = \omega - p$ **then**
8 **return** $\mathbf{e} = P^{-1}(\mathbf{H}_1\mathbf{e}_1 + \mathbf{s}_1\,\mathbf{e}_1)$ ▷ Search Step

Thus, the cost of each iteration increases to $T_{\mathbf{G}} + \binom{k}{p}$. Given the required amount of iterations is $\frac{\binom{n}{\omega}}{\binom{n-k}{\omega-p}\binom{k}{p}}$, the overall runtime can be stated as

$$T_{LB} = \frac{\binom{n}{\omega}}{\binom{n-k}{\omega-p}\binom{k}{p}} \cdot \left(T_{\mathbf{G}} + \binom{k}{p}\right) = \frac{\binom{n}{\omega}}{\binom{n-k}{\omega-p}} \cdot \left(1 + \frac{T_{\mathbf{G}}}{\binom{k}{p}}\right) > \frac{\binom{n}{\omega}}{\binom{n-k}{\omega-p}} > \frac{\binom{n}{\omega}}{\binom{n-k}{w}} = \frac{T_{Prange}}{T_{\mathbf{G}}}$$

This implies that the algorithm proposed by Lee and Brickell can never outperform Prange's algorithm by more than a polynomial factor. The exact value of this factor is dependent on the set of optimizations applied in the implementation, such as the reuse of error vector $\mathbf{e}_1$ additions in line 7 of the algorithm, as described in Section 2.2, or the early termination of the computation of $\mathbf{H}_1\mathbf{e}_1 + \mathbf{s}_1$ when the weight has already exceeded a certain threshold for a subset of computed operations. For an in-depth analysis of the bit operations involved in this algorithm, we refer the reader to [Hob12].

Leon In 1988, Leon proposed an idea [Leo88] that served as the foundation for all subsequent ISD algorithms. He observed that the approach by Lee and Brickell could be extended by introducing a new parameter ℓ which guesses additional zeros in the error vector. In other words, one could reduce the number of columns to be systematized from $n-k$ to $n-k-\ell$, because zeros were guessed in the selected ℓ coordinates within the error vector $\mathbf{e}$. Since this method is utilized in Stern's algorithm, we move the concrete analysis to the next section.

Stern — Permutation Based ISD In the subsequent algorithm, we leap forward to the year 1989 and the next era of ISD algorithms, to the point when Stern presented his work. The central concept of his algorithm is again aimed at reducing the cost of Gaussian elimination, combining and extending the ideas of Lee, Brickell and Leon. More specifically, instead of brute-forcing the correct weight-p error vector, a more sophisticated Meet-in-the-Middle (MitM) strategy is employed, as described in Section 2.2.

To understand Stern's idea, it is necessary to reconsider equation (2.1)

$$\mathbf{He} = \mathbf{I}_{n-k}\mathbf{e}_1 + \mathbf{H}_1\mathbf{e}_2 = \mathbf{s}.$$

Instead of brute forcing $\mathbf{H}_1\mathbf{e}_2$, $\forall \mathbf{e}_2 \in \mathcal{W}_p^k$, we construct $\mathbf{e}_2$ in a MitM manner $\mathbf{e}_2 = \mathbf{e}_{21} + \mathbf{e}_{22}$, with $\mathbf{e}_{21} \in \mathcal{W}_{p/2}^{k/2} \times \mathbf{0}^{k/2}, \mathbf{e}_{22} \in \mathbf{0}^{k/2} \times \mathcal{W}_{p/2}^{k/2}$. While only applying a meet-in-the-middle technique would not improve that much over Lee-Brickell's algorithm, Stern's second idea was to use Leon's [Leo88] ℓ parameter to be able to control the number of possible $\mathbf{e}_2$ which are constructed as the sum of underlying $\mathbf{e}_{21}, \mathbf{e}_{22}$.

For a better understanding of the algorithm, it is recommended that the reader closely examines Figure 2.2. In contrast to previous algorithms, Gaussian elimination in Stern's algorithm is performed on the first $n-k-\ell$ rows. It is noteworthy that the $\ell \times n-k-\ell$ sub matrix below the identity matrix can still be eliminated, which is critical to the correctness of the algorithm. We refer to a matrix in this form as *semi-systematic*.

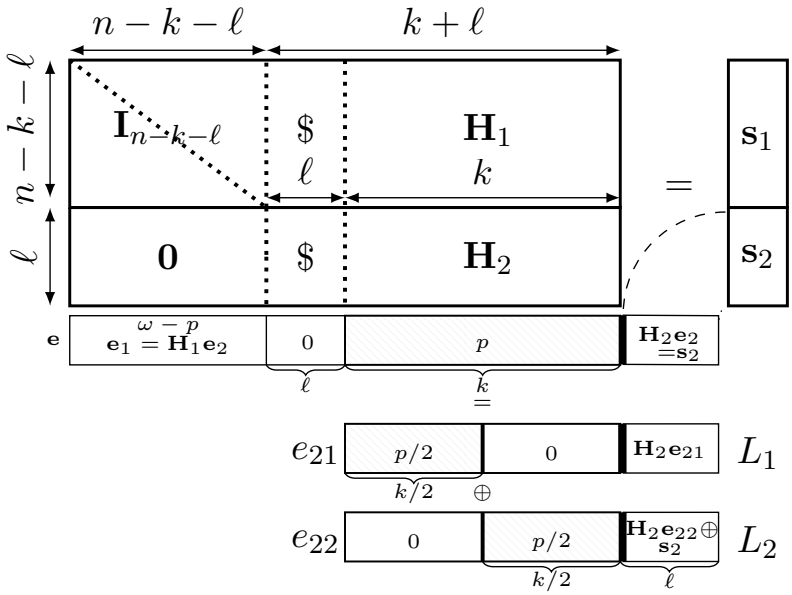


Figure 2.2: Schematic description of Stern's algorithm. The target syndrome $\mathbf{s}_2$ is added into list L_2. Thus, each collision between L_1 and L_2 holds the syndrome on the last ℓ coordinated.

Additionally, we split the matrix $\mathbf{H} = (\mathbf{H}_1, \mathbf{H}_2) \in \mathbb{F}_2^{(n-k-\ell)\times(k+\ell)} \times \mathbb{F}_2^{\ell\times(k+\ell)}$ as well as the syndrome $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{\ell}$. With these, we are now able to formulate the correct MitM step as proposed by Stern.

First, we enumerate all $\mathbf{e}_{21} \in \mathcal{W}_{p/2}^{k/2} \times \mathbf{0}^{k/2}$, ($\mathbf{e}_{22} \in \mathbf{0}^{k/2} \times \mathcal{W}_{p/2}^{k/2}$) and save the tuples $(\mathbf{e}_{21}, \mathbf{H}_2\mathbf{e}_{21})$ in a list L_1 ($(\mathbf{e}_{22}, \mathbf{H}_2\mathbf{e}_{22} + \mathbf{s}_2)$ in list L_2 respectively). As $\mathbf{s}_2$ is included in the second list, the algorithm then can search for pairs $(\mathbf{e}_{21}, \mathbf{e}_{22})$, where

$$\mathbf{H}_2\mathbf{e}_{12} = \mathbf{H}_2\mathbf{e}_{22} + \mathbf{s}_2 \tag{2.2}$$

holds. Such $(\mathbf{e}_{21}, *) \in L_1, (\mathbf{e}_{22}, *) \in L_2$ can be found by looking for equality in the second component of the lists. These *collisions* $(\mathbf{e}_{21}, \mathbf{e}_{22})$ are stored in the final list L.

Consequently, the size of the lists L_1 and L_2 is $\binom{k/2}{p/2}$ and the final list has expected size $|L| = \frac{\binom{k/2}{p/2}^2}{2^\ell}$. The latter follows from the fact that we can find at most $\binom{k/2}{p/2}^2$ equal elements in the two base lists, whereas any two elements are equal with a probability $2^{-\ell}$. Therefore, the overall cost of Stern's *Search Step* is

$$T_S = 2(|L_1| + |L|) = 2\left(\binom{k/2}{p/2} + \frac{\binom{k/2}{p/2}^2}{2^\ell}\right).$$

Note that the algorithm needs to enumerate the last list twice, once to build it, and a second time to check whether one of the constructed error vectors leads to a solution.

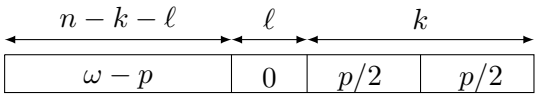


Figure 2.3: Weight distribution needed by Stern's algorithm.

In this algorithm the parameter ℓ fulfills two purposes: it determines both the number of elements in the final list and the number of permutations required until we can expect to find a *good* permutation that fulfills our demands. The algorithm expects weight $\omega - p$ in the first $n-k-\ell$ coordinates of the error vector, followed by ℓ zeros and finally twice the enumerate weight $p/2$ in each $k/2$ coordinates halves. Thus, Stern's algorithm requires in expectation $\frac{\binom{n}{\omega}}{\binom{n-k-\ell}{\omega-p}\binom{k/2}{p/2}^2}$ permutations. The weight distribution is depicted in Figure 2.3.

It should be noted that we can find all collisions between the two lists in $\mathcal{O}(|L_1|+|L_2|+|L|)$, without introducing any factors for sorting. In practice, this can be archieved via a hashmap M that at index $\mathbf{H}_2\mathbf{e}_{21}$ stores the value $\mathbf{e}_{21}$ in one of the required 2^ℓ many buckets. As $\mathbf{H}_2$ is random, we expect $\frac{\binom{k/2}{p/2}}{2^\ell}$ many elements per bucket. If we now choose $\ell \approx \log\binom{k/2}{p/2}$, only a constant amount of elements are stored in each bucket. More concrete implementation details can be found in the work by Bernstein, Lange and Peters [BLP08].

The overall runtime of Stern's *Search Step* is $T_G + T_S = \left(T_G + 2\binom{k/2}{p/2} + 2\frac{\binom{k/2}{p/2}^2}{2^\ell}\right)$. The second term represents the cost of enumerating the base lists, while the third term represents the construction of the final list and the verification of whether a solution has been found.

Algorithm 5: Stern

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, \omega$
Output : $\mathbf{e} \in \mathbb{F}_2^n$ st. $\mathbf{He} = \mathbf{s}$

choose optimal ℓ, p
repeat
  $\begin{pmatrix} \mathbf{I}_{n-k-\ell} & \mathbf{T}_1 & \mathbf{H}_1 \\ 0 & \mathbf{T}_2 & \mathbf{H}_2 \end{pmatrix} = \mathbf{GHP}^{-1}$ in semi-systematic form ▷ Permutation Step
  $\mathbf{H}_1 \in \mathbb{F}_2^{(n-k-\ell)\times k}, \mathbf{H}_2 \in \mathbb{F}_2^{\ell\times k}, \mathbf{T}_1 \in \mathbb{F}_2^{(n-k-\ell)\times \ell}, \mathbf{T}_2 \in \mathbb{F}_2^{\ell\times\ell}$
  $\bar{\mathbf{s}} = (\mathbf{s}_1, \mathbf{s}_2) = \mathbf{Gs} \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{\ell}$
  $L = \texttt{MitM}(\mathbf{H}_2, \mathbf{s}_2, p)$
  for $\mathbf{e}_2 \in L$ **do**
    if $\mathrm{wt}(\mathbf{H}_1\mathbf{e}_2) = \omega - p$ **then**
      return $\mathbf{e} = \mathbf{P}^{-1}(\mathbf{H}_1\mathbf{e}_2\, \mathbf{0}^{\ell}\, \mathbf{e}_2)$ ▷ Search Step

Therefore, the total runtime is given by:

$$T_{\text{Stern}} = \mathcal{O}\left(\frac{\binom{n}{\omega}}{\binom{n-k-\ell}{\omega-p}\binom{k/2}{p/2}^2} \cdot \left(T_G + \binom{k/2}{p/2} + \frac{\binom{k/2}{p/2}^2}{2^\ell}\right)\right)$$
$$= \mathcal{O}\left(\frac{\binom{n}{\omega}}{\binom{n-k-\ell}{\omega-p}} \cdot \left(\frac{1}{\binom{k/2}{p/2}} + \frac{1}{2^\ell}\right)\right)$$

Note that the parameters $p = p(n), \ell = \ell(n)$ are both functions in n. In most cases, the asymptotic optimization yields the minimum of the formula for $\ell = \log(\binom{k/2}{p/2})$, as this term balances the two addends.

The memory complexity of the algorithm is $\mathcal{O}\left(\binom{k/2}{p/2}\right)$, because an implementation only needs to store one of the base lists, as the remaining lists can be computed on the fly due to the ability to immediately disregard each element in the final list when it is not a solution.

Dumer – Permutation Based ISD Dumer's algorithm [Dum91] is based on Stern's algorithm as well as the observation that the zero window in the final error vector $\mathbf{e}$ is not optimal in terms of needed permutations. To overcome this Dumer's algorithms allows the base lists to enumerate on length $(k + \ell)/2$ instead of only $k/2$ as shown in Figure 2.4.

The resulting weight distribution of the error vector changes only slightly, as shown in Figure 2.5.

Compared to Stern's algorithm the expected number of permutations changes to

$$\frac{\binom{n}{\omega}}{\binom{n-k-\ell}{\omega-p}\binom{(k+l)/2}{p/2}^2}.$$

However, the size of the base list increases from $\binom{k/2}{p/2}$ to $\binom{(k+l)/2}{p/2}$.

MMT/BJMM – Enumeration Based ISD In this paragraph, we enter the most advanced algorithms in the field of ISD algorithms, thus making a jump into the year 2011. The core

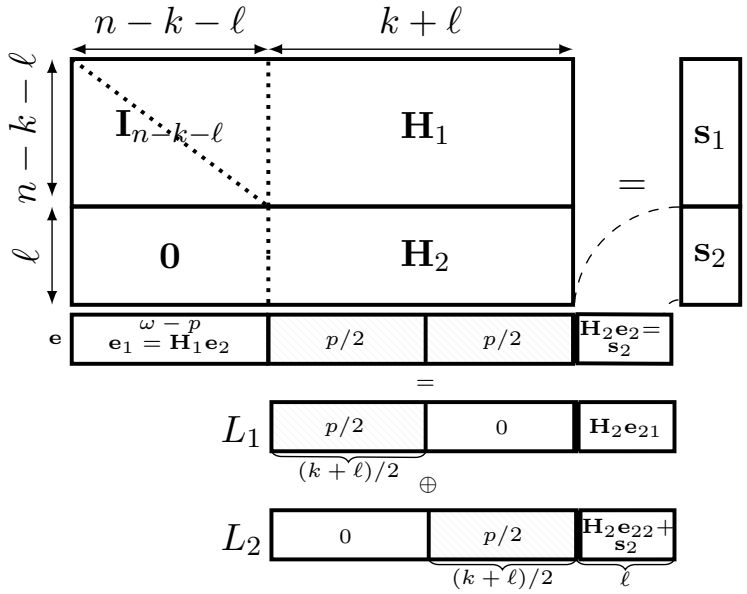


Figure 2.4: Schematic view of Dumer's algorithm. The error vector $\mathbf{e_2}$ is build as the sum of two vectors $\mathbf{e}_{21}$ and $\mathbf{e}_{22}$ which are enumerated in a Meet-in-the-Middle fashion. Each entry of the lists L_1, L_2 consists of the enumerated error vector and the matrix-vector product e.g. $\mathbf{H}_2\mathbf{e}_{21}$, which is matched on $\mathbf{s}_2$

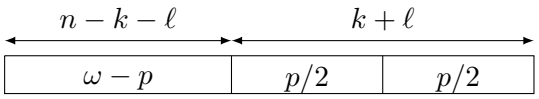


Figure 2.5: Weight distribution needed for Dumer's algorithm.

idea of the two algorithms invented by May, Meurer, Thomae [MMT11] and Becker, Joux, May, Meurer [BJMM12] is to use the representation technique invented by Howgrave-Graham and Joux [HJ10] in 2010 in the context of the Subset Sum problem (see Section 2.3) and one year later further improved by Becker, Coron and Joux [BCJ11]. In a way, the representation technique is based on a relaxation of the construction of the lists in Dumer's algorithm, by allowing the enumeration of $k + l$ coordinates (instead of $(k + l)/2$) in the error vector. This introduces some redundancy into the search space because each solution vector $\mathbf{e}_2 \in \mathcal{W}_p^{k+l}$ can now be *represented* in multiple ways as a sum of two $\mathbf{e}_2 = \mathbf{e}_{21} + \mathbf{e}_{22}$. Thus, $(\mathbf{e}_{21}, \mathbf{e}_{22})$ are called representations.

This is best visualized with an example. Let $\mathbf{x} = (10110010)$ be the solution. Thus, $\mathbf{x}$ can be represented as

$$(10100000) + (00010010),\ (10010000) + (00100010),\ (10000010) + (00110000)$$

$$(00110000) + (10000010),\ (00100010) + (10010000),\ (00010010) + (10100000)$$

We can choose 2 out of 4 of the 1-entries, so there exist $\binom{4}{2} = 6$ many representations, in general $\binom{n/2}{n/4} = 2^{n/2}$. For more detailed information on the representation technique, we recommend reading [HJ10, BCJ11, MMT11, BJMM12] or the dissertation by Meurer [Meu12].

Assume that there are 2^r many such representations for $\mathbf{e}_2$. Consequently, the two lists L_1 and L_2 are bloated by this factor. However, we are not interested in all representations

Algorithm 6: Dumer

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, \omega$
Output: $\mathbf{e} \in \mathbb{F}_2^n$ st. $\mathbf{He} = \mathbf{s}$

choose optimal ℓ, p
repeat
  $\begin{pmatrix} \mathbf{I}_{n-k-\ell} & \mathbf{H}_1 \\ 0 & \mathbf{H}_2 \end{pmatrix} = \mathbf{GHP}^{-1}$ in semi-systematic form
  $\mathbf{H}_1 \in \mathbb{F}_2^{(n-k-\ell)\times(k+\ell)}, \mathbf{H}_2 \in \mathbb{F}_2^{\ell\times(k+\ell)}$
  $\bar{\mathbf{s}} = (\mathbf{s}_1, \mathbf{s}_2) = \mathbf{Gs} \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{\ell}$
  $L = \texttt{MitM}(\mathbf{H}_2, \mathbf{s}_2, p)$
  for $\mathbf{e}_2 \in L$ **do**
    if $\mathrm{wt}(\mathbf{H}_1\mathbf{e}_2) = \omega - p$ **then**
      return $\mathbf{e} = \mathbf{P}^{-1}(\mathbf{H}_1\mathbf{e}_2\, \mathbf{e}_2)$

of the error vector, since it is sufficient if only one of those representations is in the final list. Therefore, the goal is to construct a 2^r-fraction of the two lists instead. This can be done, by forcing $\ell_1 = r < \ell$ coordinates of the two lists to be $\mathbf{0}^{l_1}$, e.g. be redefining $L_1 = \{(\mathbf{e}_{21}, \mathbf{H}_2\mathbf{e}_{21}) \,|\, \mathbf{e}_{21} \in \mathcal{W}_{p/2}^{k+l}, \pi_{\ell_1}(\mathbf{H}_2\mathbf{e}_{21}) = \mathbf{0}^{\ell_1}\}$.

While enumerating $\mathcal{W}_{p/2}^{k+\ell}$ for the two lists would result in the same runtime as the full search space, we could construct the lists themselves in MitM fashion, see Figure 2.6. For additional theoretical analysis and proofs we refer to [MMT11, BJMM12, Meu12] as well as to Section 3.2 and Section 4.5 in this work.

The full algorithm therefore works as follows: First, we start by enumerating the four base lists in a meet-in-the-middle manner, i.e.

$$L_i = \{(\mathbf{e}^i, \mathbf{H}_2\mathbf{e}^i) \,|\, \mathbf{e}^i \in \mathcal{W}_{p/2}^{(k+\ell)/2} \times \mathbf{0}^{(k+\ell)/2}\},$$
$$L_{i+1} = \{(\mathbf{e}^{i+1}, \mathbf{H}_2\mathbf{e}^{i+1}) \,|\, \mathbf{e}^{i+1} \in \mathbf{0}^{(k+\ell)/2} \times \mathcal{W}_{p/2}^{(k+\ell)/2}\},\ i = 0, 2.$$

Note that we add $(\mathbf{s}_2, \mathbf{0}^{\ell_1})$ into the second base list and $\mathbf{s_3}$ into the last base list. This is required to be able to match on zeros in the first level of the search tree. Thus, if the algorithm starts to construct the right intermediate list $L_2^{(1)}$, finding all $(\mathbf{e}^3, \mathbf{e}^4) \in L_3 \times L_4$, s.t. $\pi_{\ell_1}(\mathbf{y}_3 + \mathbf{y}_4) = \mathbf{0}^{\ell_1}$, by construction for all $(\mathbf{e}^3, \mathbf{e}^4) \in L_3 \times L_4$ it holds that $\pi_{\ell_1}(\mathbf{H}_2\mathbf{e}_3 + \mathbf{H}_2\mathbf{e}_4) = \mathbf{s}_3$. Accordingly, this step is repeated for the left intermediate list $L_1^{(1)}$.

The same happens in the next step constructing the output list L as the merge result of the two intermediate lists $L_1^{(1)}$ and $L_2^{(1)}$, where we search for elements $((\mathbf{e}_{21}, \mathbf{x}_1), (\mathbf{e}_{22}, \mathbf{x}_2)) \in L_1^{(1)} \times L_1^{(2)}$ s.t. $\pi_\ell(\mathbf{x}_1 + \mathbf{x}_2) = \mathbf{0}^\ell$. For each of those collisions, we are checking if they yield a full solution

$$\begin{aligned}\mathrm{wt}(\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}_1) &= \mathrm{wt}(\mathbf{H}_1(\mathbf{e}_{21} + \mathbf{e}_{22}) + \mathbf{s}_1) \\ &= \mathrm{wt}(\mathbf{H}_1(\mathbf{e}^1 + \mathbf{e}^2 + \mathbf{e}^3 + \mathbf{e}^4) + \mathbf{s}_1) = \omega - p.\end{aligned}$$

Additionally, in Section 4.5, a more sophisticated time-memory trade-off is proposed for the MMT and BJMM algorithm. This trade-off is based on the idea of reusing already built lists many times. Assume that we choose $\ell_1 > r$, reducing the probability of a representation

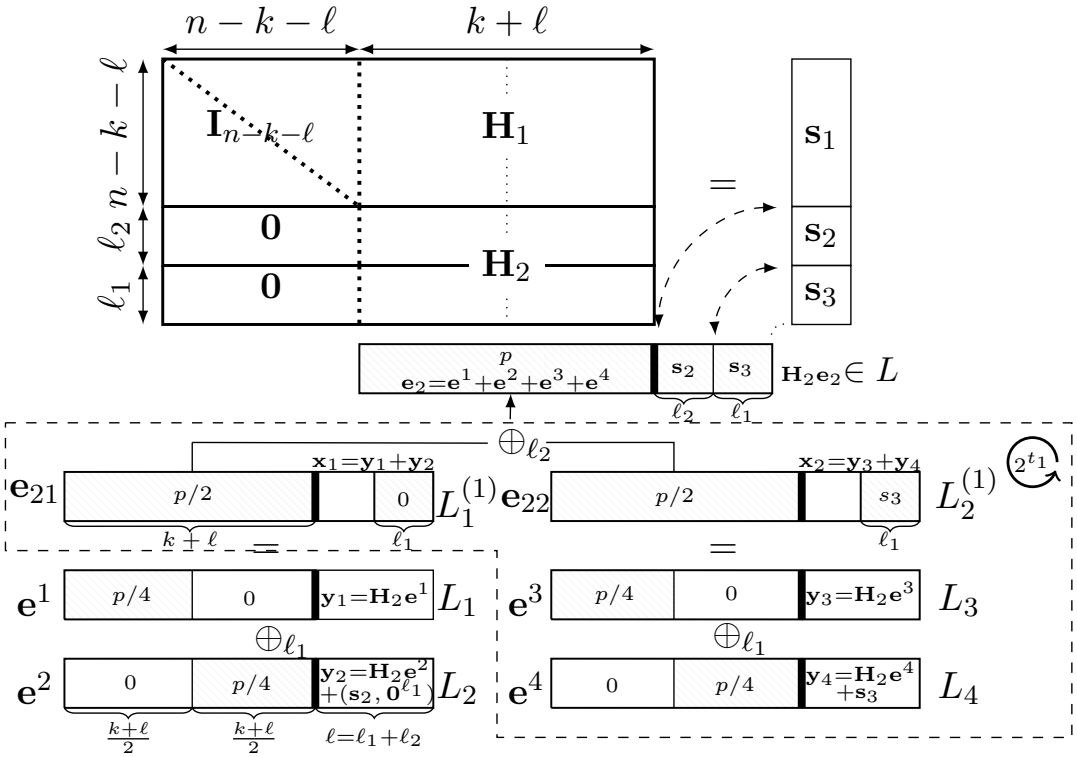


Figure 2.6: Search tree construction of the MMT algorithm in depth 2, with the improvement of reusing parts of the search tree. Here once the base list L_1 is built up we are reusing it 2^{t_1} times, each time we rebuild the right hand side of the tree.

to be detected as such to $\frac{1}{2^{\ell_1-r}}$. In this case Section 4.4 shows that it is possible to compensate for this by repeatedly rebuilding the rest of the search tree $2^{t_1} = 2^{\ell_1-r}$ many times. We show that in terms of expected runtime, we could not improve, but the memory complexity is noticeably reduced. In Section 3.2 and Section 4.5, we present a more elaborate analysis of the MMT and BJMM algorithm, thus we now focus on the practical aspects of the implementation and the results.

As mentioned at the beginning of this chapter the implementation [2] of the MMT/BJMM algorithm is one of the main results of this work. Therefore, let us describe the details of the implementation compared to the theoretical description of the algorithm. First, notice that the lists L_1 and L_3 are the same by definition, so there is no need to build them twice. Secondly, notice that L_2 and L_4 only differ in the syndrome we add to the lists. Thus, we build up a plain list without the syndromes and only add to it while iterating over the lists. Furthermore, we replace the lists L_1 and L_3 with a hashmap. While merging two lists consists of sorting one list and a subsequent binary search for each element in the other list, it is considerably faster to first hash one list into a hashmap (which is algorithmically the same as sorting if the key of the hashmap is the value itself) and afterward doing a single lookup for each element of the other list in the hashmap.

Enumerating vectors of low weight Enumerating Vectors in $\mathcal{W}_p^k$, or to be more precise, the set $\mathcal{W}_p^{(k+\ell)/2}$, is crucial for all ISD algorithms, be it the enumeration-based ones like Stern's

[2] publicly available under `https://github.com/FloydZ/decoding`

or Dumer's, or the representation-based ones like MMT, BJMM. For the best performance, the enumeration algorithm needs to fulfill the following constraints.

- It is not on the error vector $\mathbf{e}_i$ itself, but rather on the modifications made to move from $\mathbf{e}_i$ to its successor $\mathbf{e}_{i+1}$ we are interested in. If we have such a *change sequence* we can simply compute $\mathbf{He}_{i+1}$ from $\mathbf{He}_i$ by adding and subtracting the changed columns. To initiate this sequence, we once compute the first element $\mathbf{He}_1$, with $\mathbf{e}_1 = (1, \dots, 1, 0, \dots, 0)$ via a matrix-vector multiplication.
- The changes between successive elements should be minimal, ideally involving only two bit positions. Moreover, it is highly beneficial if these changes occur nearby, meaning that they are only one or two bit positions apart. This significantly reduces the memory span required by the algorithm, thereby minimizing the occurrence of cache misses.

The Chase Sequence (described in Exercise 45 of [Knu11]) satisfies all of these requirements and is therefore an ideal choice. In our implementation, we precompute this *change list* and save it in a list of size $\binom{(k+\ell)/2}{p/2} \cdot p$.

Exact parameters For our record computations, we used the configurations shown in Table 2.1. Only the optimal parameters are shown, where the optimization process bruteforces over p, ℓ, ℓ_1 and the hashmap meta parameters. The optimal ℓ, ℓ_1 that were found in this way is normally only one dimension bigger than the theoretic optimization predicted. We assume that this is due to our stream-join implementation favoring smaller lists. Notice that all our records were made with $p = 1$, and consequently, the final error vector in list L has weight 4, respectively 3 in case of QC, because the last list was replaced with the list of syndromes, see Section 3.5. A detailed analysis of the cost of memory and where the break-even points lay to switch to *high-memory* configurations is made in Section 3.4.2. Whereby, with *high-memory* configurations a bigger p, e.g. $p = 3$ is meant. Additionally, a detailed description of the hardness of each instance can be found Section 3.4.1.

	Instance				HashMap 1		HashMap 2		Lists (log)		
	n	ℓ	ℓ_1	p	BS	NB	BS	NB	$\lvert L_1 \rvert$	$\lvert L_1^{(1)} \rvert$	$\lvert L \rvert$
McEliece	1223	17	2	1	150	2^2	5	2^{15}	8.9	15.9	16.8
	1284	17	2	1	146	2^2	4	2^{15}	9.0	16.0	17.1
QC	2118	20	1	1	300	2^1	2	2^{19}	9.1	17.1	15.3
	2306	20	1	1	320	2^1	2	2^{19}	9.2	17.4	15.8
	2502	21	1	1	340	2^1	2	2^{20}	9.3	17.6	15.2
	2706	21	1	1	360	2^1	2	2^{20}	9.4	17.8	15.7
	2918	21	1	1	360	2^1	2	2^{20}	9.5	18.0	16.1
	3138	23	7	1	10	2^7	1	2^{16}	9.6	12.3	8.5

Table 2.1: Used parameters for the record computations. BS describes the number of elements each bucket can contain, and NB descibes the number of buckets each hashmap contains.

The last quasi-cyclic record, QC-3138, was computed with the new time-memory trade-off. The chosen parameters result in much smaller lists. The final list, whose computation denotes the majority of the runtime is roughly 2^8 times smaller than the final list computed by the algorithm that does not utilize time-memory trade-offs. On the other side, the new algorithm needs to recompute the final list $2^{t_1} = 2^6 = 64$ times per iteration. Overall, this still yields a runtime improvement factor of 3.2.

What didn't work During the implementation process, a lot of testing was conducted on different ideas that led to runtime improvements. In this section, we list a couple of ideas that did not improve the underlying algorithm:

- In [BLP08], a lot of optimization effort was put into reducing the cost of Gaussian elimination. Instead of permuting n columns (and therefore permuting $(n-k)/2$ non-unity vectors into the first $n-k$ columns in expectation), the idea is to only permute $c \ll n$, i.e. $c = 8$ columns. In this case, the cost of systematizing the first $n-k$ columns is dramatically reduced. However, on the other side, we do not apply a random permutation anymore. As a result, have to pick more permutations overall to hit a *good* one. In the end, we were unable to improve with these methods due to two reasons: First, the Gaussian elimination holds only a share of about 5% of the overall runtime, meaning any improvement would not significantly reduce the total runtime. Secondly, the bookkeeping requirements of the algorithm are too high for a significant speed-up to be reached. Consequently, all of our algorithms and records were made with an implementation of the Method of the 4 Russians.

- Our servers are equipped with AMD EPYCs of the second generation, meaning they are x86 processors whose word width is 64 bits. Because of this, the implementation needs to allocate 2 words for a row of the parity check matrix containing 65 variables. Therefore, we waste a total of 63 bits for each of the last words of a row. Thus, it could be beneficial to introduce a new parameter d which removes as many as d columns from the parity check matrix, such that each row fits perfectly into a multiple of the word size. Unfortunately, our experiments did not show any significant improvement.

- Recent advancements in the field of ISD algorithms, as [MO15,BM18], have incorporated nearest neighbour search techniques to accelerate the search-tree computation. An introduction to these techniques can be found in Section 2.4. In our implementation, we have also integrated these LSH (Locality-Sensitive Hashing) techniques. However, our experiments did not yield any practical runtime improvements when using these techniques. Further investigation revealed that this was due to our implementation's preference for low-memory configurations. It should be noted that the nearest neighbour search allows for savings of permutations of the algorithm, at the cost of an increased effort to compute the final list. Therefore, the savings are realized in the Gaussian elimination, base list construction, and matching to the final list. Our benchmark results indicate that these procedures collectively account for only around 10-15% of the total running time when using our optimal low-memory configuration. Thus, the nearest neighbour approach is not well-suited for our streaming design, as the level-one lists need to be available in memory to fully leverage the nearest neighbour gain. As a result, the nearest neighbour strategy did not contribute to our record computations.

Results With the implementation described in Section 2.2, we were able to break two McEliece challenges ($n = 1284, 1223$, more details about the hardness in Section 3.4), six quasi-cyclic BIKE, HQC challenges (from $n = 2118$ up to $n = 2918$, more details about the hardness in Section 3.5, and $n = 3138$ with a more memory-optimized version of the algorithm, more details in Section 4.5.3) and five in the large weight ternary syndrome decoding setting. Thus, we gathered more and better data points for step 2 in our extrapolation methodology.

The last step of the extrapolation methodology is shown in Table 2.2 for McEliece and Table 2.3 for BIKE and HQC, where we extrapolated from our record computations to

cryptographic sized instances. In order to provide a better understanding of the resulting runtimes, we relate their security level to the corresponding security of AES-128, -192 and -256. To estimate the time complexity of breaking AES on our cluster, we benchmarked the number of AES encryptions that our cluster can perform per second. Using this information, we were able to calculate the expected time to break AES with respective key sizes. The resulting bit-difference is presented in both tables.

McEliece	Category 1 $n = 3488$	Category 3 $n = 4608$	Category 5a $n = 6688$	Category 5b $n = 6960$	Category 5c $n = 8192$
constant:					
unlimited	−0.82	−26.10	−24.04	−24.73	5.14
$M \leq 2^{80}$	0.29	−23.95	−13.65	−13.36	21.18
$M \leq 2^{60}$	2.63	−19.85	− 9.07	− 8.58	26.43
logarithmic:					
unlimited	0.84	−24.15	−21.60	−22.26	7.85
$M \leq 2^{80}$	1.54	−22.65	−12.40	−12.11	22.47
$M \leq 2^{60}$	3.46	−19.18	− 8.32	− 7.81	27.21
cube-root:					
	8.94	−13.79	− 1.57	− 0.72	35.43

Table 2.2: Bit-difference in security of Classic McEliece and AES with respective key-length considering different memory access cost.

As in Chapter 1 described, extrapolation inherits uncertainties. One of these uncertainties is related to the correct modeling of memory accesses. To address this issue, we are using three different memory penalization models to account for the large memory usage. These models multiply the runtime by either a logarithmic, cube-root, or square-root factor, resulting in a runtime function of $T_d \cdot \log M$, $T_d \cdot \sqrt[3]{M}$ or $T_d \cdot \sqrt{M}$, respectively. A detailed analysis is conducted in Section 3.4.2.

Summarizing this work, in the contexts of BIKE and HQC, the security claims made by the authors were validated across all memory models. However, in the McEliece setting, significant discrepancies were identified between the desired security levels and the extrapolated ones. For example, assuming unlimited memory, McEliece-4608 showed a difference of up to 26 bits. In a more realistic setting, where the runtime is penalized based on memory usage and the total memory is limited to 2^{60}, a difference of 19 bits was still observed. This work demonstrates the superior performance of advanced Information Set Decoding algorithms at the implementation level. Consequently, these algorithms must be considered when estimating the security level of code-based cryptographic schemes.

Quasi-Cyclic	Category 1	Category 3	Category 5
constant:			
BIKE	0.73	−1.07	2.13
HQC	−1.84	1.19	−0.86
logarithmic:			
BIKE	1.24	−0.10	3.50
HQC	−1.51	1.61	−0.38
cube-root:			
BIKE	1.89	0.79	4.60
HQC	−0.67	2.71	0.90

Table 2.3: Bit-difference in security of BIKE/HQC and AES with respective key-length considering different memory access cost.

2.3 Subset Sum Problem

In Chapter 4, a new time-memory trade-off for the Syndrome Decoding Problem is introduced. This is done in the light of the well-known Subset Sum problem, which is defined as follows:

Definition 6. Let $\mathbf{a} := (a_1, \ldots, a_n) \in \mathbb{Z}_{2^n}^n$ be drawn uniformly at random. For a random $\mathbf{e} \in \{0,1\}^n$ with $\mathrm{wt}(\mathbf{e}) = \frac{n}{2}$, let $t := \langle \mathbf{a}, \mathbf{e} \rangle$. The *(random) subset sum problem* is: given $(\mathbf{a}, t)$ find any $\mathbf{e}' \in \{0,1\}^n$ satisfying $\langle \mathbf{a}, \mathbf{e}' \rangle = t$. We call any such $\mathbf{e}'$ a *solution* and $(\mathbf{a}, t)$ an *instance*.

It should be noted that our consideration of the subset sum problem is different from the general subset sum problem in that we focus on the modular version over $\mathbb{Z}_{2^n}$, which is motivated by practical applications, unlike the general subset sum problem, where the size of the elements of $\mathbf{a}$ is unbounded.

Moreover, our focus is on analyzing the *random* Subset Sum problem, where the vector $\mathbf{a}$ is chosen uniformly at random from $\mathbb{Z}_{2^n}^n$, as opposed to worst-case instances where $\mathbf{a}$ can be chosen arbitrarily. This choice is motivated by the cryptanalytic nature of our work, as cryptographic problems usually involve random instances. In the definition of the random Subset Sum problem, the modulus $M := 2^n$ is considered a function of the problem dimension n, leading to a commonly used measure called *density*. For a random Subset Sum instance $(\mathbf{a}, t) \in (\mathbb{Z}_M)^{n+1}$, the density d is defined as $d := \frac{n}{\log M}$. It is worth noting that the density is directly related to the expected number of solutions, which is given by $E[\#\text{ Solutions}] = \frac{2^n}{M} = M^{d-1}$. Thus, a higher density corresponds to a greater number of expected solutions, while a lower density relates to fewer solutions. In the realm of cryptography, the objective is typically to achieve a unique solution, such as a unique key. Consequently, in the early stages of constructing cryptographic systems, low-density instances were primarily used. However, Coster et al. [CLOS91] (in conjunction with independent research by Joux and Stern [JS91]) demonstrated that instances with a density of $d < 0.94$ could be reduced to solving the shortest vector problem in lattices. While computing shortest vectors in lattices is generally NP-hard, practical analysis have shown that these instances can be solved efficiently.

We are only interested in instances with density $d = 1$, for which no similar result could be obtained. The best-known algorithms to solve them require an exponential amount of time and a few of them even require exponential memory. Although known algorithms capable of solving the Subset Sum problem can work with arbitrary weight ω of the solution vector $\mathbf{e}$, we focus specifically on instances where $\mathrm{wt}(\mathbf{e}) = \frac{n}{2}$. This is motivated by the intuition that instances of this form are the hardest, as the entire search space $\binom{n}{\omega}$ is maximized when $\omega = \frac{n}{2}$.

The relationship between the Subset Sum Problem and the Syndrome Decoding Problem can be understood by viewing the SDP as a vectorized form of the Subset Sum Problem. This can be achieved by mapping each column, $\mathbf{h}_i$, of the parity check matrix $\mathbf{H}$ into the ring $\mathbb{Z}_{2^{n-k}}$ through the canonical isomorphism. As such, each new algorithm developed for the Subset Sum setting can be applied to the Syndrome Decoding Problem.

2.4 Nearest Neighbour Problem

Historically, the Nearest Neighour problem was defined by giving a list L_1 and a *query point*. The objective is then to locate an element in L_1 which is *close* under a certain metric to the given query point. Nowadays, the most common way to define the problem is to find the *solution* or *closest pair* $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$, e.g. in two inputs lists. The problem is formally defined as follows.

Definition 7 ((Bichromatic) Closest Pair Problem)**.** Let $n \in \mathbb{N}$, $\omega \in \left[0, \frac{1}{2}\right]$ and $\lambda \in (0, 1]$. Let $L_1 = (\mathbf{v}_i)_{i \in 2^{\lambda n}}, L_2 = (\mathbf{w}_i)_{i \in 2^{\lambda n}} \in (\mathbb{F}_2^n)^{2^{\lambda n}}$ be two lists containing elements uniformly drawn at random, together with a distinguished pair $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ with $\mathrm{wt}(\mathbf{x} + \mathbf{y}) = \omega n$. The *Closest Pair Problem* $\mathcal{CP}_{d,\lambda,\omega}$ asks to find this *closest pair* $(\mathbf{x}, \mathbf{y})$ given L_1, L_2 and the weight parameter ω. We call $(\mathbf{x}, \mathbf{y})$ the *solution* of the $\mathcal{CP}_{d,\lambda,\omega}$ problem.

The *nearest neighbour problem* in extension to the *closest pair problem* now asks to find **all** *solutions* $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ which are close together. This problem, ever since the first definition by Minsky and Papert [MP69] as well as Knuth [Knu98], led to several interesting results for the *euclidean* or *hamming* metric. The latter is of special interest in this work because it is the key algorithmic ingredient in modern ISD algorithms.

The latest ISD algorithms, proposed by May-Ozerov [MO15] and Both-May [BM18], use nearest neighbour algorithms to accelerate the search tree construction. However, the nearest neighbour algorithm proposed by May and Ozerov incurs a high polynomial overhead, making it impractical to implement. Therefore, in Chapter 5, we take the first step towards using nearest neighbour algorithms as a subroutine in ISD algorithms by introducing a novel, implementable algorithm that can optimally solve the problem. A proof-of-concept implementation of this algorithm can be found at `https://github.com/FloydZ/NNAlgorithm`.

2.5 Legendre Pseudorandom Function

In the last chapter of this work, the *Legendre pseudorandom function* is analyzed. For our purpose, a sufficient definition of a *pseudorandom function* (PRF) is as follows. Let $f_k : X \mapsto Y$ be, s.t. f_k is efficiently computable $\forall x \in X$ and computationally indistinguishable from a randomly chosen function $f : X \mapsto Y$.

Before proceeding, it is important to provide a brief review of the definition of the Legendre symbol for the reader's convenience. The Legendre symbol is defined as follows: Let p be a

prime, and let $x \in \mathbb{F}_p$. Then

$$\left(\frac{x}{p}\right) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x \text{ is a quadratic residue} \\ -1 & \text{else.} \end{cases}$$

This leads to the following definition.

Definition 8 (Legendre Function)**.** Let $k \in \mathbb{F}_p$ be a secret key. The *Legendre PRF* is defined as $L_k : \mathbb{F}_p \to \{-1, 0, 1\}$,

$$L_k(x) := \left(\frac{x+k}{p}\right).$$

This PRF construction was already proposed in 1987 by Damgård [Dam88]. Given oracle access to this function, the problem at hand is to find the secret key k. Note that $L_k(x) = 0$ if and only if $x = p - k$, so it is easy to check whether $x \in \mathbb{F}_p$ is the secret key.

The Legendre PRF has recently attracted renewed attention due to its potential use in various applications, including multi-party computation [GRR+16] and quantum secure signature schemes [BD20]. Additionally, the Ethereum 2.0 protocol [Fou20a, Fou20b], which has been designed to increase transaction throughput and achieve competitiveness with modern credit card systems, relies on the Legendre PRF as an efficient and fast pseudorandom function in their Proof-of-Stake mechanism. However, if the Legendre PRF fails to provide sufficient security, a malicious user could potentially steal the stake of an honest user, making it crucial to assess the security of this newly proposed PRF. Moreover, given the large user base of the Ethereum blockchain, an attacker might be interested in stealing the secret keys of a few users without any preference for which ones. Hence, it is natural to inquire whether the attack can be accelerated with access to multiple keys. This work presents three distinct attacks on the Legendre PRF. The first two leverage precomputation, whether an attacker wants to recover a single key or multiple keys. The last attack of the chapter focuses on the multi key setting with no precomputation. This is realized via the method of *distinguished points* by van Oorschot and Wiener [vW99]. All the algorithms discussed have been implemented and are publicly available at `https://github.com/FloydZ/prep-legendre`.

3 McEliece Needs a Break – Solving McEliece-1284 and Quasi-Cyclic-2918 with Modern ISD

With the recent shift to post-quantum algorithms it becomes increasingly important to provide precise bit-security estimates for code-based cryptography such as McEliece and quasi-cyclic schemes like BIKE and HQC. While there has been significant progress on information set decoding (ISD) algorithms within the last decade, it is still unclear to which extent this affects current cryptographic security estimates.

We provide the first concrete implementations for representation-based ISD, such as May-Meurer-Thomae (MMT) or Becker-Joux-May-Meurer (BJMM), that are parameter-optimized for the McEliece and quasi-cyclic setting. Although MMT and BJMM consume more memory than naive ISD algorithms like Prange, we demonstrate that these algorithms lead to significant speedups for practical cryptanalysis on medium-sized instances (around 60 bit). More concretely, we provide data for the record computations of McEliece-1223 and McEliece-1284 (old record: 1161), and for the quasi-cyclic setting up to code length 2918 (before: 1938).

Based on our record computations we extrapolate to the bit-security level of the proposed BIKE, HQC and McEliece parameters in NIST's standardization process. For BIKE/HQC, we also show how to transfer the Decoding-One-Out-of-Many (DOOM) technique to MMT/BJMM. Although we achieve significant DOOM speedups, our estimates confirm the bit-security levels of BIKE and HQC.

For the proposed McEliece round-3 192 bit and two out of three 256 bit parameter sets, however, our extrapolation indicates a security level overestimate by roughly 20 and 10 bits, respectively, i.e., the high-security McEliece instantiations may be a bit less secure than desired.

The content of this Chapter is the result of a collaboration with Andre Esser and Alexander May. It previously appeared as McEliece Needs a Break – Solving McEliece-1284 and Quasi-Cyclic-2918 with Modern ISD in EuroCrypt 2022 and is reproduced here with permission.

3.1 Introduction

For building trust in cryptographic instantiations it is of utmost importance to provide a certain level of real-world cryptanalysis effort. Code-based cryptography is usually build on the difficulty of correcting errors in binary linear codes. Let C be a binary linear code of length n and dimension k, i.e., C is a k-dimensional subspace of $\mathbb{F}_2^n$. We denote by $H \in \mathbb{F}_2^{(n-k)\times n}$ a parity-check matrix of C, thus we have $H\mathbf{c} = \mathbf{0}$ for all $\mathbf{c} \in C$.

Let $\mathbf{x} = \mathbf{c} + \mathbf{e}$ be an erroneous codeword with error $\mathbf{e}$ of small known Hamming weight $\omega = \mathrm{wt}(\mathbf{e})$. Let $\mathbf{s} := H\mathbf{x} = H\mathbf{e}$ denote the syndrome of $\mathbf{x}$. Then decoding $\mathbf{x}$ is equivalent to the recovery of the weight-ω error vector from $H\mathbf{e} = \mathbf{s}$.

Permutation-Dominated ISD – Prange. Let $P \in \mathbb{F}_2^{n\times n}$ be a permutation matrix. Then $(HP)(P^{-1}\mathbf{e}) = \bar{H}\bar{\mathbf{e}} = \mathbf{s}$ is another weight-ω decoding instance with permutated solution $\bar{\mathbf{e}} = P^{-1}\mathbf{e}$.

Assume that $\bar{\mathbf{e}} = (\mathbf{e}_1, \mathbf{e}_2)$ with $\mathbf{e}_2 = 0^k$. An application of Gaussian elimination $G \in \mathbb{F}_2^{(n-k)\times(n-k)}$ on the first $n-k$ columns of $\bar{H}$ yields

$$G\bar{H}\bar{\mathbf{e}} = (I_{n-k}H')\bar{\mathbf{e}} = \mathbf{e}_1 + H'\mathbf{e}_2 = \mathbf{e}_1 = G\mathbf{s}. \tag{3.1}$$

Thus, from $\mathrm{wt}(G\mathbf{s}) = \omega$ we conclude that $\bar{\mathbf{e}} = (G\mathbf{s}, 0^k)$ and $\mathbf{e} = P\bar{\mathbf{e}}$.

In summary, if we apply the correct permutation P that sends all weight ω to the first $n-k$ coordinates, then we decode correctly in polynomial time. This is why the first $n-k$ coordinates are called an *information set*, and the above algorithm is called *information set decoding* (ISD). This ISD algorithm, due to Prange [Pra62], is the main tool for estimating the security of code-based cryptography such as McEliece and BIKE/HQC.

We would like to stress that the complexity of Prange's algorithm is mainly dominated by finding a proper permutation P, which takes super-polynomial time for cryptographic instances. All other steps of the algorithm are polynomial time. This is why we call Prange a *permutation-dominated* ISD algorithm. A permutation-dominated ISD performs especially well for small weight errors $\mathbf{e}$ and large co-dimension $n-k$. More precisely, we have to find a permutation P that sends all weight ω to the size-$(n-k)$ information set, which happens with probability

$$\Pr[P \text{ good}] = \frac{\binom{n-k}{\omega}}{\binom{n}{\omega}} = \frac{(n-k)(n-k-1)\dots(n-k-\omega+1)}{n(n-1)\dots(n-\omega+1)}.$$

Let $\omega = o(n)$, and let us denote C's rate by $R = \frac{k}{n}$. Then Prange's permutation-based ISD takes up to polynomial factors expected running time

$$T = \frac{1}{\Pr[P \text{ good}]} \approx \left(\frac{1}{1-R}\right)^{\omega}. \tag{3.2}$$

Modern Enumeration-Dominated ISD – MMT/BJMM. The core idea of all ISD improvements since Prange's algorithm is to allow for some weight $p > 0$ outside the information set. Thus we allow in Equation (3.1) that $\mathrm{wt}(\mathbf{e}_2) = p$, and have to *enumerate* $H'\mathbf{e}_2$. However, the cost of enumerating $H'\mathbf{e}_2$ may be well compensated by the larger success probability of a good permutation

$$\Pr[P \text{ good}] = \frac{\binom{n-k}{\omega-p}\binom{k}{p}}{\binom{n}{\omega}}.$$

Indeed, modern ISD algorithms like MMT [MMT11] and BJMM [BJMM12] use the *representation technique* to heavily speed up enumeration. In the large weight regime $\omega = \Theta(n)$, parameter optimization of MMT/BJMM yields that these ISD algorithm do not only balance the cost of permutation and enumeration, but their enumeration is so efficient that it eventually almost completely dominates their runtime. This is why we call these algorithms *enumeration-dominated* ISD.

The large error regime $\omega = \Theta(n)$ is beneficial for MMT/BJMM, since for large-weight errors $\mathbf{e}$ it becomes hard to send all weight to the information set, and additionally a large-weight $\mathbf{e}$ introduces a large number of representations. From a cryptographic perspective

however it remains unclear if MMT/BJMM also offer speedups for concrete cryptographic instances of interest.

Main question: How much improve modern enumeration-based ISD algorithms cryptanalysis of code-based crypto in practice (if at all)?

What makes this question especially hard to answer is that as opposed to permutation-based ISD, all enumeration-based ISD algorithms require a significant amount of memory. Thus, even if enumeration provides significant speedups it is unclear if it can compensate for the introduced memory access costs. As a consequence, the discussion of enumeration-based ISD in the NIST standardization process of McEliece already led to controversial debates [Var21b, Var21a]. We would like to stress that up to our work, all decoding records on decodingchallenge.org have been achieved either using Prange's permutation-based ISD, or Dumer's first generation enumeration-based ISD [Dum91].

In the asymptotic setting, Canto-Torres and Sendrier [TS16a] showed that all enumeration-dominated ISD approaches offer in the small-weight setting $\omega = o(n)$ only a speedup from T in Equation (3.2) to $T^{1-o(1)}$, i.e., the speedup asymptotically vanishes. While this is good news for the overall soundness of our cryptographic constructions, it tells us very little about the concrete hardness of their instantiations.

Recently, Esser and Bellini [EB22] pursued a more practice-oriented approach by providing a concrete *code estimator*, analogous to the successfully applied *lattice estimators* [APS15]. Their code estimator also serves us as a basis for optimizing our ISD implementations. However, such an estimator certainly fails to model realistic memory access costs.

3.1.1 Our Contributions

Fast enumeration-dominated ISD implementation. We provide the first efficient, freely available implementation of MMT/BJMM, i.e., a representation-based enumeration-dominated ISD. Our implementation uses depth 2 search-trees, which seems to provide best results for the cryptographic weight regime. For the cryptographic instances that we attack we used weight $p = 4$ for McEliece with code length up to 1284, and $p = 3$ for BIKE/HQC. However, our benchmarking predicts that McEliece with code length larger than 1350 should be attacked with significantly larger weight $p = 8$. Our code is publicly available on GitHub.[1]

In comparison to other available implementations of (first-generation) enumeration-based ISD algorithms, our implementation performs significantly faster. Our experimental results demonstrate that in cryptanalytic practice even moderately small instances of McEliece can be attacked faster using modern enumeration-based ISD.

So far, our efforts to additionally speed up our enumeration-based ISD implementations with localilty-sensitive hashing (LSH) techniques [MO15, BM18] did not succeed. We discuss the reasons in Section 3.3.3.

Real-world cryptanalysis of medium-sized instances. For building trust in the bit-security level of cryptographic instances, it is crucial to solve medium-sized instances, e.g. with 60 bit security. This gives us stable data points from which we can more reliably extrapolate to high security levels. An example of good cryptanalysis practice is the break of RSA-768 [KAF+10] that allows us to precisely estimate the security of RSA-1024.

Before our work, for McEliece the record code length $n = 1161$ on decodingchallenge.org was reported by Narisada, Fukushima, Kiyomoto with an estimated bit-security level of 56.0.

[1] `https://github.com/FloydZ/decoding`

We add two new records McEliece-1223 and McEliece-1284 with estimated bit-security levels of 58.3 and 60.7, respectively. These record computations took us approximately 5 CPU years and 22 CPU years.

As a small technical ingredient to further speed up our new MMT/BJMM implementation, we show how to use the parity of ω to increase the information set size by 1, which saved us approximately 9% of the total running time.

For the quasi-cyclic setting we improved the previously best code length 1938 of Bossard [ALL19] with 6400 CPU days to the five new records 2118, 2306, 2502, 2706, and 2918. The last has a bit-security level of 58.6, and took us (only) 1700 CPU days.

As a technical contribution for the quasi-cyclic setting, we show how to properly generalize the Decoding-One-Out-of-Many (DOOM) strategy to the setting of tree-based enumeration-dominated ISD algorithms. Implementing our DOOM strategy gave us roughly a $\sqrt{n-k}$ experimental speedup, where $n-k=n/2$ is the co-dimension in the quasi-cyclic setting. This coincides with our theoretical analysis, see Section 3.5.1.

Our real-world cryptanalysis shows that memory access certainly has to be taken into account when computing bit-security, but it might be less costly than suggested. More precisely, our ISD implementations support the so-called *logarithmic cost model*, where an algorithm with time T and memory M has cost $T \cdot \log_2 M$.

Solid bit-security estimations for McEliece and BIKE/HQC. Based on our record computations and further extensive benchmarking for larger dimensions, we extrapolate to the proposed round-3 McEliece and BIKE/HQC instances. To this end, we also estimate via benchmarking the complexity of breaking AES-128 (NIST Category 1), AES-192 (Category 3) and AES-256 (Category 5) on our hardware.

For McEliece, we find that in the logarithmic cost model the Category 1 instance `mceliece348864` achieves quite precisely the desired 128-bit security level, whereas the Category 3 instance (`mceliece460896`) and two out of three Category 5 instances (`mceliece6688128` and `mceliece6960119`) fail to reach their security level by roughly 20 and 10 bit, even when restricting our attacks to a memory upper limit of $M \leq 2^{80}$. Hence, these instances seem to overestimate security.

For BIKE/HQC, our extrapolation shows that the proposed round-3 instances achieve their desired bit-security levels quite accurately.

Discussion of our results. In our opinion, the appearance of a small security gap for McEliece and no security gap for BIKE/HQC is due to the different weight regimes. Whereas BIKE/HQC use small weight $\omega = \sqrt{n}$, McEliece relies on Goppa codes with relatively large weight $\omega = \Theta(n/\log n)$,

Both the BIKE/HQC and McEliece teams use the asymptotic formula from Equation (3.2) to analyze their bit-security, which is the more accurate the smaller the weight ω. Hence, while in the BIKE/HQC setting the speedups that we achieve from enumeration-dominated ISD in practice are compensated by other polynomial factors (e.g. Gaussian elimination), in McEliece's (large) weight regime the speedups are so significant that they indeed lead to measurable security losses.

Comparison to previous security estimates. Baldi et al. [BBC+19] and more recently Esser and Bellini [EB22] already provide concrete bit security estimates for code-based NIST candidates. Further, Esser and Bellini introduce new variations of the BJMM and MMT

algorithm based on nearest neighbor search, which however did not result in practical gains for our implementation (see Section 3.3.3 for details).

Both works [BBC+19, EB22] take into account memory access costs. While [BBC+19] uses a logarithmic cost model, [EB22] considers three models (constant, logarithmic, and cube-root). As opposed to our work, [BBC+19, EB22] both solely rely on the computation of runtime formulas.

Our work extends and specifies these estimates in the following way. For the first time, we establish with our record computations solid experimental data points for the hardness of instances with roughly 60 bits security. Moreover, our implementation for the first time allows us to identify a proper memory access model that closely matches our experimental observations. Based on our data points, we extrapolate to NIST parameters of cryptographic relevance, using an estimator like [EB22] with the proper memory access model choice. This eventually allows for a much more reliable security estimate.

3.2 The MMT/BJMM Algorithm

Let us briefly recap the MMT and BJMM algorithm. From an algorithmic point of view both algorithms are the same. The benefit from BJMM over MMT comes from allowing a more fine-grained parameter selection. In our practical experiments, we mainly used the simpler MMT parameters. Therefore, we refer to our implementation as MMT in the simple parameter setting, and as BJMM in the fine-grained parameter setting.

Main idea. Let $H\mathbf{e} = \mathbf{s}$ be our syndrome decoding instance with parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, unknown error $\mathbf{e} \in \mathbb{F}_2^n$ of known Hamming weight ω, and syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$.

As usual in information set decoding, we use some permutation matrix $P \in \mathbb{F}_2^{n\times n}$ to send most of the weight ω to the information set. Let $\bar{H} = HP$ and $\bar{\mathbf{e}} = P^{-1}\mathbf{e}$. Then, obviously $\mathbf{s} = \bar{H}\bar{\mathbf{e}}$.

MMT/BJMM now computes the *semi-systematic form* as originally suggested by Dumer [Dum91]. To this end, fix some parameter $\ell \leq n-k$. Let $\bar{\mathbf{e}} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{k+\ell}$, and assume for ease of exposition that the first $n-k-\ell$ columns of $\bar{H}$ form a full rank matrix. Then we can apply a Gaussian elimination $G \in \mathbb{F}_2^{(n-k)\times(n-k)}$ that yields

$$\bar{\mathbf{s}} := G\mathbf{s} = G\bar{H}\bar{\mathbf{e}} = \begin{pmatrix} I_{n-k-\ell} & H_1 \\ 0 & H_2 \end{pmatrix} = (\mathbf{e}_1 + H_1\mathbf{e}_2, H_2\mathbf{e}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{\ell}. \tag{3.3}$$

Let $\bar{\mathbf{s}} = (\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{\ell}$. From Equation (3.3) we obtain the identity $\mathbf{s}_2 = H_2\mathbf{e}_2$. MMT/BJMM constructs $\mathbf{e}_2$ of weight p satisfying $\mathbf{s}_2 = H_2\mathbf{e}_2$. Notice that for the correct $\mathbf{e}_2$ we directly obtain from Equation (3.3) that

$$\mathbf{e}_1 = \mathbf{s}_1 + H_1\mathbf{e}_2. \tag{3.4}$$

Since we know that $\mathrm{wt}(\mathbf{e}_1) = \omega - p$, MMT/BJMM checks for correctness of $\mathbf{e}_2$ via $\mathrm{wt}(\mathbf{s}_1 + H_1\mathbf{e}_2) \stackrel{?}{=} \omega - p$.

Tree-based recursive construction of $\mathbf{e}_2$ using representations. For the tree-based construction of $\mathbf{e}_2$ the reader is advised to closely follow Figure 3.1. Here, we assume at least some reader's familiarity with the representation technique, otherwise we refer to [HJ10, MMT11] for an introduction.

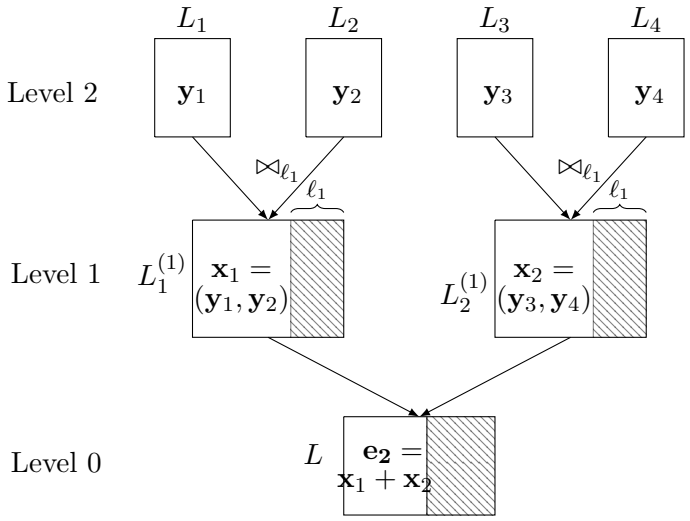


Figure 3.1: Search tree of the MMT algorithm. Striped areas indicate matching of the last coordinates of $H\mathbf{x}_i$ or $H(\mathbf{x}_1 + \mathbf{x}_2)$ with some predefined values.

We write $\mathbf{e}_2$ as a sum $\mathbf{e}_2 = \mathbf{x}_1 + \mathbf{x}_2$ with $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{F}_2^{k+\ell}$ and $\mathrm{wt}(\mathbf{x}_1) = \mathrm{wt}(\mathbf{x}_2) = p_1$. In MMT we choose $p_1 = p/2$, whereas in BJMM we allow for $p_1 \geq p/2$ s.t. a certain amount of one-coordinates in $\mathbf{x}_1, \mathbf{x}_2$ has to cancel in their $\mathbb{F}_2$-sum.

The number of ways to represent the weight-p $\mathbf{e}_2$ as a sum of two weight-p_1 $\mathbf{x}_1, \mathbf{x}_2$, called the number of *representations*, is

$$R = \binom{p}{p/2}\binom{k+\ell-p}{p_1-p/2}.$$

However, it suffices to construct $\mathbf{e}_2$ from a single representation $(\mathbf{x}_1, \mathbf{x}_2)$. Recall from Equation (3.4) that we have

$$H_2\mathbf{x}_1 = H_2\mathbf{x}_2 + \mathbf{s}_2 \in \mathbb{F}_2^{\ell}.$$

Notice that we do not know the value of $H_2\mathbf{x}_1$. Let us define $\ell_1 := \lfloor \log_2(R) \rfloor$. Since there exist R representations $(\mathbf{x}_1, \mathbf{x}_2)$ of $\mathbf{e}_2$, we expect that for any fixed random target vector $\mathbf{t} \in \mathbb{F}_2^{\ell_1}$ and any projection $\pi : \mathbb{F}_2^{\ell} \to \mathbb{F}_2^{\ell_1}$ on ℓ_1 coordinates (e.g. the last ℓ_1 bits), there is on expectation at least one representation $(\mathbf{x}_1, \mathbf{x}_2)$ that satisfies

$$\pi(H_2\mathbf{x}_1) = \mathbf{t} = \pi(H_2\mathbf{x}_2 + \mathbf{s}_2).$$

We construct all $\mathbf{x}_1$ satisfying $\pi(H_2\mathbf{x}_1) = \mathbf{t}$ in a standard Meet-in-the-Middle fashion. To this end, we enumerate vectors of length $\frac{k+\ell}{2}$ and weight $p_2 := \frac{p_1}{2}$ in baselists L_1, L_2. Analogously, we find all $\mathbf{x}_2$ that satisfy $\pi(H_2\mathbf{x}_2 + \mathbf{s}_2)$ via a Meet-in-the-Middle from baselists L_3, L_4, see Figure 3.1.

The resulting MMT/BJMM algorithm is described in Algorithm 7.

Runtime analysis. For every permutation P, MMT/BJMM builds the search tree from Figure 3.1. P has to send weight $\omega - p$ to the information set of size $n - k - \ell$ which happens with probability

$$q := \Pr[P \text{ good}] = \frac{\binom{n-k-\ell}{\omega-p}\binom{k+\ell}{p}}{\binom{n}{\omega}} \quad . \tag{3.5}$$

Algorithm 7: MMT Algorithm

Input : $H \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, w \in \mathbb{N}$
Output : $\mathbf{e} \in \mathbb{F}_2^n, H\mathbf{e} = \mathbf{s}$

begin
  Choose optimal ℓ, p, p_2
  Set $\ell_1 = \lfloor \binom{p}{p/2}\binom{k+\ell-p}{p_1-p/2} \rfloor$ and $p_1 = 2p_2$
  repeat
    choose random permutation matrix P
    $\bar{H} = \begin{pmatrix} I_{n-k-\ell} & H_1 \\ 0 & H_2 \end{pmatrix} = GHP$ in semi-systematic form
    $\bar{\mathbf{s}} = (\mathbf{s}_1, \mathbf{s}_2) = G\mathbf{s}$
    Compute

$$L_1 = L_3 = \{(\mathbf{y}_1, H_2\mathbf{y}_1) \mid \mathbf{y}_1 \in \mathbb{F}_2^{(k+\ell)/2} \times 0^{(k+\ell)/2}, \mathrm{wt}(\mathbf{y}_1) = p_2\}$$
$$L_2 = \{(\mathbf{y}_2, H_2\mathbf{y}_2) \mid \mathbf{y}_2 \in 0^{(k+\ell)/2} \times \mathbb{F}_2^{(k+\ell)/2}, \mathrm{wt}(\mathbf{y}_2) = p_2\}$$
$$L_4 = \{(\mathbf{y}_2, H_2\mathbf{y}_2 + \mathbf{s}_2) \mid \mathbf{y}_2 \in 0^{(k+\ell)/2} \times \mathbb{F}_2^{(k+\ell)/2}, \mathrm{wt}(\mathbf{y}_2) = p_2\}$$

    Choose some random $t \in \mathbb{F}_2^{\ell_1}$
    Compute

$$L_1^{(1)} = \{(\mathbf{x}_1, H_2\mathbf{x}_1) \mid \pi(H_2\mathbf{x}_1) = \mathbf{t}, \mathbf{x}_1 = \mathbf{y}_1 + \mathbf{y}_2\} \text{ from } L_1, L_2$$
$$L_2^{(1)} = \{(\mathbf{x}_2, H_2\mathbf{x}_2 + \mathbf{s}_2) \mid \pi(H_2\mathbf{x}_2 + \mathbf{s}_2) = \mathbf{t}, \mathbf{x}_2 = \mathbf{y}_1 + \mathbf{y}_2\} \text{ from } L_3, L_4$$

    Compute $L = \{\mathbf{e}_2 \mid H_2\mathbf{e}_2 = \mathbf{s}_2, \mathbf{e}_2 = \mathbf{x}_1 + \mathbf{x}_2\}$ from $L_1^{(1)}, L_2^{(2)}$
    for $\mathbf{e}_2 \in L$ **do**
      $\mathbf{e}_1 = H_1\mathbf{e}_2 + \mathbf{s}_1$
      if $\mathrm{wt}(\mathbf{e}_1) \leq \omega - p$ **then**
        return $P^{-1}(\mathbf{e}_1, \mathbf{e}_2)$
    end
  end
end

The tree construction works in time T_{list}, which is roughly linear in the maximal list size in Figure 3.1. Let $|L_i|$ denote the common list base size. Then it is not hard to see that the overall expected runtime can be bounded by

$$T = q^{-1} \cdot \tilde{\mathcal{O}}\left(T_{\text{list}}\right), \text{ where } T_{\text{list}} = \max\left\{|L_i|, \frac{|L_i|^2}{2^{\ell_1}}, \frac{|L_i|^4}{2^{\ell+\ell_1}}\right\}.$$

Part of the strength of our MMT/BJMM implementation in the subsequent section is to keep the polynomial factors hidden in the above $\tilde{\mathcal{O}}(\cdot)$-notion small, e.g. by using a suitable hash map data structure.

Locality-Sensitive Hashing (LSH). Most recent improvements to the ISD landscape [MO15,BM18] use nearest neighbor search techniques to speed-up the search-tree computation. We also included LSH techniques in our implementation. However for the so far benchmarked

code dimensions, LSH did not (yet) lead to relevant speedups. See Section 3.3.3 for further discussion on LSH.

3.3 Implementing MMT/BJMM Efficiently

In Section 3.3.1 we introduce an elementary, but at least for McEliece practically effective decoding trick. We then detail our MMT/BJMM implementation in Section 3.3.2

3.3.1 Parity Bit Trick

Let us introduce a small technical trick to speed up ISD algorithms, whenever the weight of the error vector is known. Known error weight is the standard case in code-based cryptography. The trick is so elementary that we would be surprised if it was missed in literature so far, but we failed to find a reference, let alone some proper analysis.

Let $H\mathbf{e} = \mathbf{s}$ be our syndrome decoding instance, where ω is the known error weight of $\mathbf{e}$. Then certainly

$$\langle 1^n, \mathbf{e} \rangle = \omega \mod 2.$$

Thus, we can initially append to the parity-check matrix the row vector 1^n, and append to $\mathbf{s}$ the parity bit $\omega \bmod 2$.

Notice that this *parity bit* trick increases the co-dimension by 1, and therefore also the size of the information set. For Prange's permutation-dominated ISD this results in a speedup of

$$\frac{\binom{n}{\omega}}{\binom{n-k}{\omega}} \cdot \frac{\binom{n-k+1}{\omega}}{\binom{n}{\omega}} = \frac{\binom{n-k+1}{\omega}}{\binom{n-k}{\omega}} \ .$$

The speedup for Prange with *parity bit* is the larger the smaller our co-dimension $n-k$ is. For McEliece with small co-dimension and our new record instance ($n = 1284, k = 1028, \omega = 24$) we obtain more than a 10% speedup, and for the proposed round-3 McEliece parameter sets it is in the range 8-9 %. If instead of Prange's algorithm we use the MMT/BJMM variant that performed best in our benchmarks then the speedup is still in practice a remarkable 9% for the $n = 1284$ instance, and 6-7% for the round-3 parameter sets.

For BIKE and HQC with large co-dimension $n - k = \frac{n}{2}$ and way bigger n, the speedup from the *parity bit* goes down to only 0.5-1%.

3.3.2 Implementation

Parameter Selection and Benchmarking.

As seen in Section 3.3 and Algorithm 7, the MMT/BJMM algorithm —even when limited to depth 2 search trees— still has to be run with optimized parameters for the weights on all levels of the search tree, and an optimized ℓ. We used an adapted formula based on the syndrome decoding estimator tool by Esser and Bellini [EB22] that precisely reflects our implementation to obtain initial predictions for those parameters on concrete instances. We then refined the choice experimentally.

To this end, we measure the number of iterations per second our cluster is able to process for a specific parameter configuration. We then calculate the expected runtime to solve the instance as the number of expected permutations q^{-1} (from Equation (3.5)) divided by the number of permutations per second. We then (brute-force) searched for an optimal

configuration in a small interval around the initial prediction that minimizes the expected runtime.

For instances with McEliece code length $n \leq 1350$ we find optimality of the most simple non-trivial MMT weight configuration with weight $p_2 = 1$ for the baselists $L_1, \ldots, L_4$ on level 2, weight $p_1 = 2$ in level 1, and eventually weight $p = 4$ on level 0 in Figure 3.1. We refer to the weight configuration $p_2 = 1$ in the baselists as the *low-memory* configuration. Recall that for $p_2 = 0$ MMT becomes Prange's algorithm, and therefore is a memory-less algorithm.

We call configurations with $p_2 \in \{2, 3\}$ *high-memory* configurations. The choice $p_2 = 3$ already requires roughly 40 gigabytes of memory. Increasing the weight to $p_2 = 4$ would increase the memory consumption by another factor of approximately 2^{11}.

Gaussian Elimination.

For the Gaussian elimination step we use an open source version [AB21] of the *Method of the four Russians for Inversion* (M4RI), as already proposed by Bernstein et al. and Peters [BLP08,Pet10]. According to [Bar07] the M4RI algorithm is preferable to other advanced algorithms like Strassen [Str69] up to matrices of dimension six-thousand. We extended the functionality of [AB21] to allow for performing a transformation to semi-systematic form, without fully inverting the given matrix. Even for small-memory configurations the permutation and Gaussian elimination step together only account for roughly 2-3% of our total computation time. Therefore we refrain from further optimizations of this step, as introduced in [BLP08, Pet10].

Search Tree Construction.

To save memory, we implemented the search tree from Figure 3.1 in a streaming fashion, as already suggested by Wagner in [Wag02]. See Figure 3.2 for an illustration showing that we have to store only two baselists and one intermediate list.

Our implementation exploits that $L_1 = L_3$, and L_2 and L_4 only differ by addition of $\mathbf{s}_2$ to the label $H_2\mathbf{y}_2$. To compute the join of L_1, L_2 to $L_1^{(1)}$ we hash list L_1 into a hashmap H_{L_1} using $\pi(H_2\mathbf{y}_1)$ as an index. Then we search each label $\pi(H_2\mathbf{y}_2)$ of list L_2 in H_{L_1}, and store all resulting matches in another hashmap $H_{L_1^{(1)}}$ using the remaining $\ell - \ell_1$ bits of label $H_2\mathbf{x}_2$.

For the right half of the tree we reuse the hashmap H_{L_1} and the list L_2, to which we add $\mathbf{s}_2$. The resulting matches from $L_2^{(1)}$ are directly processed on-the-fly with $H_{L_1^{(1)}}$, producing $\mathbf{e}_2 \in L$.The candidates $\mathbf{e}_2$ are again processed on the fly, and checked whether they lead to the correct counterpart $\mathbf{e}_1$.

For speed optimization we worked with a single 64-bit register computation of our candidate solutions throughout all levels of the tree. Even eventually falsifying incorrect $\mathbf{e}_1$ can be performed within 64 bit most of the time. For construction of the baselists L_1, L_2 we used a Gray-code type enumeration.

Parallelization of Low- and High-Memory Configuration.

Recall that ISD algorithms consist of a permutation and an enumeration part. In the low-memory regime, we only perform a light enumeration with $p_2 = 1$. The algorithmic complexity is in this configuration dominated by the number of permutations. Therefore, we choose to fully parallelize permutations, i.e., each thread computes its own permutation, Gaussian elimination, and copy of the search tree.

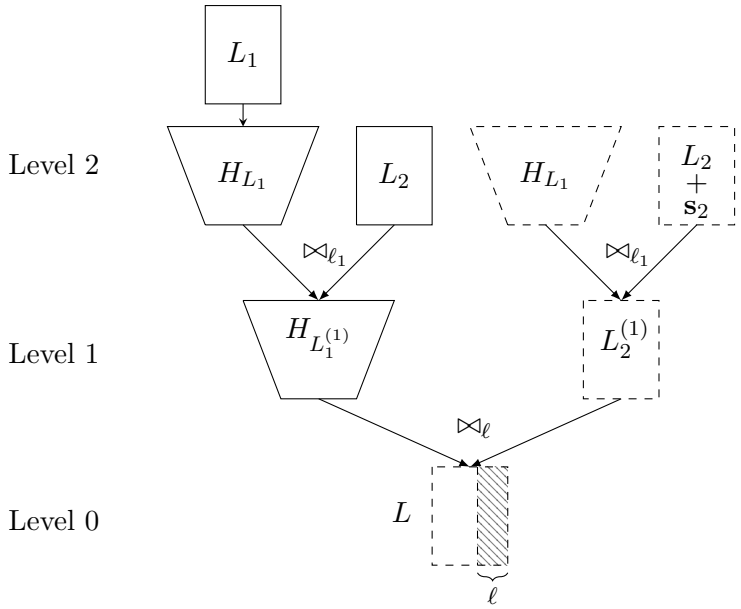


Figure 3.2: Streaming implementation of the MMT / BJMM search tree in depth two using two physically stored lists and hashmaps. H_{L_1} and $H_{L_1^{(1)}}$ denote the hashmaps, while dashed lists and hashmaps are processed on the fly.

In the high memory regime however, the number of permutations is drastically reduced at the cost of an increasing enumeration complexity. Therefore for the $p_2 = 2, 3$ configurations we choose to parallelize the search tree construction. To this end, we parallelize among N threads by splitting the baselist into N chunks of equal size. To prevent race conditions, every bucket of a hashmap is also split in N equally sized partitions, where only thread number i can insert into partition i.

3.3.3 Other Benchmarked Variants – Depth 3 and LSH

It is known that asymptotically, and in the high error regime, an increased search tree depth and the use of LSH techniques [MO15, BM18] both yield asymptotic improvements. We implemented these techniques, but for the following reasons we did not use them for our record computations.

The estimates for depth 2 and 3 complexities are rather close, not giving clear favour to depth 3. This explains why in practice the overhead of another tree level outweighs its benefits.

LSH allows to save on some permutations at the cost of an increased complexity of computing L from the level-one lists $L_1^{(1)}, L_2^{(2)}$. Accordingly, the LSH savings lie in the Gaussian elimination, the base list construction and the matching to level one. Our benchmarks reveal that these procedures together only account for 10-15% of the total running time in the low-memory setting. Moreover, LSH is not well compatible with our streaming design. Therefore, LSH did not yet provide speedups for our computations, but this will likely change for future record computations, see the discussion in Section 3.4.2.

3.4 McEliece Cryptanalysis

In this section we give our experimental results on McEliece instances. Besides giving background information on our two record computations, we discuss how good different memory cost models fit our experimental data.

Moreover, we show that MMT reaches its asymptotics *slowly from below*, which in turn implies that purely asymptotic estimates tend to overestimate bit security levels. We eloborate on how to properly estimate McEliece bit security levels in Section 3.7.

For our computations we used a cluster consisting of two nodes, each one equipped with 2 AMD EPYC 7742 processors and 2 TB of RAM. This amounts for a total of 256 physical cores, allowing for a parallelization via 512 threads.

3.4.1 Record Computations

Table 3.1 states the instance parameters of our records we achieved in the McEliece-like decoding category of decodingchallenge.org.

n	k	ω	time (days)	CPU years	bit complexity
1223	979	23	2.45	1.71	58.3
1284	1028	24	31.43	22.04	60.7

Table 3.1: Parameters of the largest solved McEliece instances, needed wallclock time, CPU years and bit complexity estimate.

McEliece-1223.

We benchmarked an optimal MMT parameter choice of $(\ell, \ell_1, p, p_2) = (17, 2, 4, 1)$. With this low-memory configuration our computing cluster processed $2^{33.32}$ permutations per day, which gives an expected computation time of 8.22 days, since in total we expect $2^{36.36}$ permutations from Equation (3.5). We solved the instance in 2.45 days, only 30% of the expected running time. If we model the runtime as a geometrically distributed random variable with parameter $q = 2^{-36.36}$, then we succeed within 30% of the expectation with probability 26%.

McEliece-1284.

Our benchmarks identified the same optimal parameter set $(\ell, \ell_1, p, p_2) = (17, 2, 4, 1)$ as for McEliece-1223. For this configuration our estimator formula yields an expected amount of $2^{38.49}$ permutations. We benchmarked a total performance of $2^{33.26}$ permutations per day, leading to an expected 37.47 days. We solved the challenge within 31.43 days which is about 84% of the expected running time, and happens with probability about 57%.

Experimental Results and Discussion.

In Figure 3.3, we plot our record computations as squares. Before we performed our record computations, we heavily tested our implementation with smaller instances $n < 1000$. As before, we computed the expected running time for every value of n, denoted as larger open diamonds in Figure 3.3, via the quotient of expected permutations and permutations per

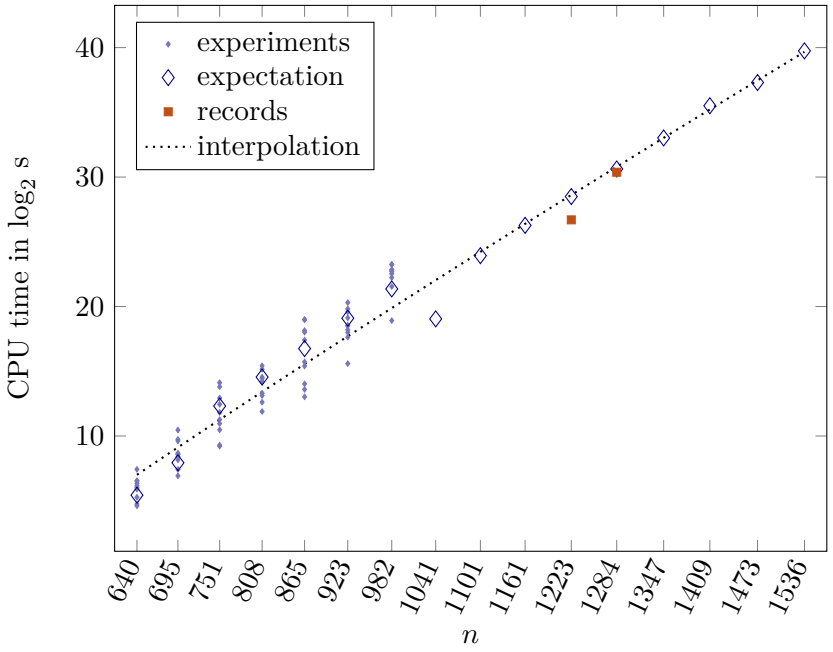


Figure 3.3: Running time of experiments and records as well as interpolation for McEliece.

second on our cluster. The small diamonds depict the actual data points which cluster around their expectation, as desired.

The runtime jumps from $n = 695$ to $n = 751$ and from $n = 982$ to $n = 1041$ can be explained by the instance generation method. For every choice of n the parameters k and ω are derived on decodingchallenge.org as (see [ALL19]) $k = \left\lceil \frac{4n}{5} \right\rceil$ and $\omega = \left\lceil \frac{n}{5\lceil \log n \rceil} \right\rceil$.

For most consecutive instances ω increases by one, but for $n = 695$ to $n = 751$ there is an increase of 2, whereas for $n = 982$ to $n = 1041$ there is a decrease of 1. Besides these jumps, the instance generation closely follows the Classic McEliece strategy.

Comparison with other Implementations.

We also compare our implementation to those of Landais [Lan12] and Vasseur [Vas]. These implementations were used to break the previous McEliece challenges, with the only exception of the $n = 1161$ computation by Narisada, Fukushima, and Kiyomoto that uses non-publicly available code. We find that our implementation performs 12.46 and 17.85 times faster on the McEliece-1284 challenge and 9.56 and 20.36 times faster on the McEliece-1223 instance than [Lan12] and [Vas], respectively.

3.4.2 The Cost of Memory

Not very surprising, our experimental results show that large memory consumption leads to practical slowdown. This is in line with the conclusion of the McEliece team [CCU+20] that a constant memory access cost model, not accounting for any memory costs, underestimates security. However, this leaves the question how to properly penalize an algorithm with running time T for using memory M. Most prominent models use logarithmic, cube-root or square-root penalty factors, i.e. costs of $T \cdot \log M$, $T \cdot \sqrt[3]{M}$ or $T \cdot \sqrt{M}$, respectively.

In [EB22] it was shown that logarithmic costs do not heavily influence parameter selection of enumeration-based ISD algorithms, whereas cube-root costs let the MMT advantage

deteriorate. Thus, it is crucial to evaluate which cost model most closely matches experimental data.

Break-Even Point for High-Memory Regime.

Using our estimator formula we find that under cube-root memory access costs the point where the low-memory configuration $p_2 = 1$ becomes inferior lies around $n = 6000$, falling in the 256-bit security regime of McEliece. In contrast, the logarithmic cost model predicts the break even point at $n \geq 1161$.

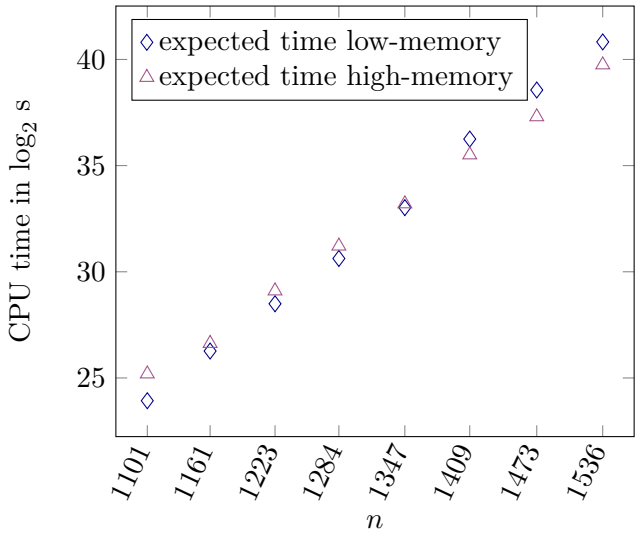


Figure 3.4: Estimated running times for low- and high-memory configurations.

By benchmarking the running time of our implementation in the range $n = 1101$ to 1536 for different choices of p_2, see Figure 3.4, we experimentally find a break even point at $n \approx 1400$. For $n \geq 1400$ the choice $p_2 = 3$ performs best. The configuration $p_2 = 2$ was experimentally always inferior to $p_2 = 1$ and $p_2 = 3$ (which is consistent with our estimation). The reason is that as opposed to $p_2 = 2$ the configuration $p_2 = 3$ does allow for a BJMM parameter selection with $p = 8 < 4p_2$, and also leads to a better balancing of list sizes in the search tree.

In conclusion, the experimentally benchmarked break-even point is way closer to the theoretical point of $n = 1161$ in the logarithmic cost model than to $n = 6000$ in the cube-root model. This already supports the use of logarithmic costs, especially when we take into account that many of our implementation details heavily reward the use of low-memory configurations, such as:

- *Large L3 Caches.* Our processors have an exceptionally large L3 cache of 256 MB that is capable of holding our complete lists in low-memory configurations.

- *Use of Hashmaps.* As indicated in Section 3.3.2, our parallelization is less effective e.g. for hashmaps in the large-memory regime.

- *Communication complexity.* As opposed to low-memory configurations the high-memory regime requires thread communication for parallelization.

3.4.3 McEliece Asymptotics: From Above and from Below

It was analyzed in [TS16a], that asymptotically all ISD algorithms converge for McEliece instances to Pranges complexity bound

$$\left(1 - \frac{k}{n}\right)^{\omega}, \text{ see Equation (3.2).}$$

Since we have rate $\frac{k}{n} = 0.8$ for the decodingchallenge.org parameters, we expect an asymptotic runtime of

$$T(n) = 2^{2.32\frac{n}{\log n}}. \tag{3.6}$$

This asymptotic estimates supresses polynomial factors. Thus, in Prange's algorithm we have rather $2^{2.32(1+o(1))\frac{n}{\log n}}$, and the algorithm converges to Equation (3.6) from *above*. For other advanced ISD algorithms the asymptotics suppresses polynomial runtime factors as well as second order improvements. Thus, they have runtime $2^{2.32(1\pm o(1))\frac{n}{\log n}}$, and it is unclear whether they converge from above or below.

Let us take the interpolation line from our data in Figure 3.3, where we use for the runtime exponent the model function $f(n) = a \cdot \frac{n}{5 \log n} + b$. The interpolation yields

$$a = 2.17 \text{ and } b = -22.97,$$

where the negative b accounts for instances which can be solved in less than a second. The small slope a experimentally demonstrates that the convergence is clearly from *below*, even including realistic memory cost.

However, we still want to find the most realistic memory cost model. To this end, we used our estimator for all instances from Figure 3.3 in the three different memory access models, constant, logarithmic and cube-root. The resulting bit complexities are illustrated in Figure 3.5 in a range $n \in [640, 1536]$ for which in practice we have optimal $p_2 \leq 3$. For each model we computed the interpolation according to $f(n) = a \cdot \frac{n}{5 \log n} + b$. For a constant access cost we find $a = 2.04$, for a logarithmic $a = 2.13$, and for the cube-root model we find $a = 2.24$. Hence, again a logarithmic access cost most accurately models our experimental data.

Cryptographic Parameters.

So far, we considered only instances with $n \leq 1536$. However, the current round 3 McEliece parameters reach up to code length $n = 8192$. Thus, we also used our estimator to check the slopes a in this cryptographic regime. We compared the ISD algorithms of Prange, Stern and our MMT/BJMM variant. For all algorithms we imposed logarithmic memory access costs $T \cdot \log M$ and considered the three cases of unlimited available memory, as well as 2^{80}-bit and 2^{60}-bit as memory limitation for M. The results for the exponent model $f(n) = a \cdot \frac{n}{5 \log n} + b$ are given in Table 3.2.

We observe that Prange does not quickly converge to Equation (3.2) from above. For Stern and MMT however we are even in the most restrictive memory setting below the exponent from Equation (3.2). This clearly indicates an overestimate of McEliece security using Equation (3.2). We eloborate on this more qualitatively in Section 3.7.

3.5 The Quasi-Cyclic Setting: BIKE and HQC

The proposals of BIKE and HQC —both alternate finalists of the NIST PQC competition— use double circulant codes with code rate $\frac{1}{2}$, i.e., $n = 2k$. It has been shown by Sendrier [Sen11]

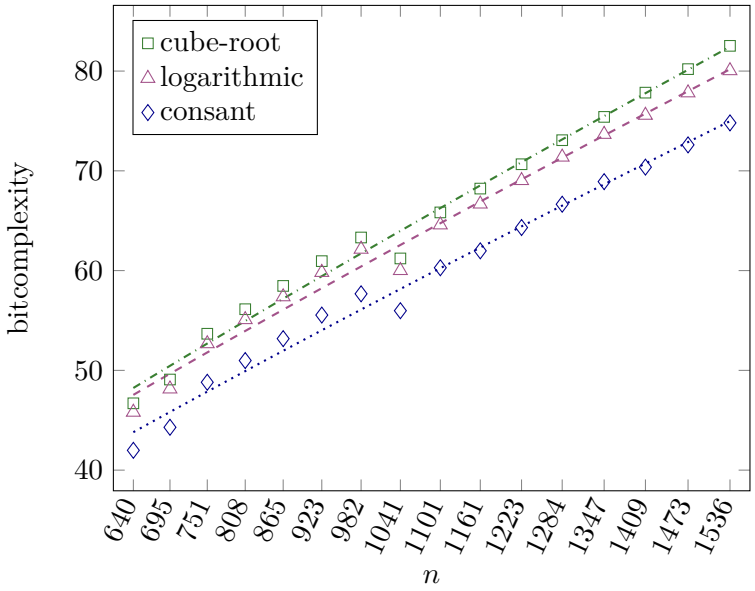


Figure 3.5: Estimated bitcomplexities for different memory access cost models and corresponding interpolations.

	Prange	Stern	MMT
unlimited	2.438	2.297	2.075
$M \leq 2^{80}$	2.438	2.299	2.207
$M \leq 2^{60}$	2.438	2.308	2.287

Table 3.2: Slope of interpolation of bitcomplexities under logarithmic memory access costs considering instances with $n \leq 8192$ according to the model function $f(n) = a \cdot \frac{n}{5 \log n} + b$.

that these codes allow for a speedup of Stern's ISD algorithm by a factor of up to $\sqrt{k}$. The basic observation is that the cyclicity immediately introduces k instances of the syndrome decoding problem, where a solution to any of the k instances is a cyclic rotation of the original solution. Thus, this technique is widely known as *Decoding One Out of Many* (DOOM).

Let $H_1, H_2 \in \mathbb{F}_2^{k \times k}$ be two circulant matrices satisfying

$$\begin{pmatrix} H_1 & H_2 \end{pmatrix} (\mathbf{e}_1, \mathbf{e}_2) = \mathbf{s}$$

with $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{F}_2^k$. Let us denote by $\mathrm{rot}_i(\mathbf{x})$ the cyclic left rotation of $\mathbf{x}$ by i positions. Then for any $i = 0, \ldots, k-1$ we have

$$\begin{pmatrix} H_1 & H_2 \end{pmatrix} (\mathrm{rot}_i(\mathbf{e}_1), \mathrm{rot}_i(\mathbf{e}_2)) = \mathrm{rot}_i(\mathbf{s}) =: \mathbf{s}_i.$$

This implies that a solution to any of the k instances $(H_1 H_2, \mathbf{s}_i, \omega)$ yields $(\mathbf{e}_1, \mathbf{e}_2)$.

Note that in the special case of $\mathbf{s} = \mathbf{0}$, thus, when actually searching for a small codeword the instances are all the same, meaning there simply exist k different solutions $\mathbf{e}$. In this case any ISD algorithm obtains a speedup of k.

For $\mathbf{s} \neq \mathbf{0}$ one usually assumes a speedup of $\sqrt{k}$ in the quasi-cyclic setting, referring to Sendrier's DOOM result [Sen11]. However, [Sen11] only analyzes Stern's algorithm.

In the following section we adapt the idea of Sendrier's DOOM to the MMT/BJMM algorithm in the specific setting of double circulant codes, achieving speedups slightly larger than $\sqrt{k}$ both in theory and practical experiments.

3.5.1 Decoding one out of k (DOOM_k)

To obtain a speedup from the k instances we modify the search tree of our MMT/BJMM variant such that in every iteration all k syndromes are considered. To this end, similar to Sendrier, we first enlarge list L_4 (compare to Figure 3.2) by exchanging every element $(\mathbf{x}, H\mathbf{x}) \in L_4$ by $(\mathbf{x}, H\mathbf{x} + \bar{\mathbf{s}}_i)$ for all $i = 1 \dots k$, where $\bar{\mathbf{s}}_i := G\mathbf{s}_i$ denotes the i-th syndrome after the Gaussian elimination. This results in a list that is k times larger than L_4. To compensate for this increased list size we enumerate in L_4 initially only vectors of weight $p_2 - 1$ rather than p_2.

This simple change already allows for a speedup of our MMT/BJMM algorithm of order $\sqrt{k}$, as shown in the following lemma.

Lemma 1 (DOOM_k speedup)**.** *A syndrome decoding instance with double circulant parity-check matrix, code rate $\frac{k}{n} = \frac{1}{2}$ and error weight $\omega = \Theta(\sqrt{k})$ allows for a speedup of the MMT/BJMM algorithm by a factor of $\Omega(\sqrt{k})$.*

Proof. First note, that since list L_4 is duplicated for every syndrome $\mathbf{s}_i$ by the correctness of the original MMT algorithm our modification is able to retrieve any of the rotated solutions if permutation distributed the weight properly.

Let us first analyze the impact of our change on the size of the list L_4. The decrease of the weight of the vectors in L_4 from p_2 to $p_2 - 1$ decreases the size by a factor of

$$\delta_{\mathrm{L}} := \frac{\binom{(k+\ell)/2}{p_2}}{\binom{(k+\ell)/2}{p_2-1}} = \frac{\frac{k+\ell}{2} - p_2 + 1}{p_2} \approx \frac{k}{2 \cdot p_2},$$

since $\ell, p_2 \ll k$. Thus, together with the initial blowup by k for every syndrome we end up with a list that is roughly $2p_2$ times as large as the original list. Next let us study the effect on the probability of a random permutation distributing the error weight properly for anyone of the k error vector rotations, which is

$$\delta_P := \frac{\binom{n-k-\ell}{\omega-p+1}\binom{k+\ell}{p-1} \cdot k / \binom{n}{\omega}}{\binom{n-k-\ell}{\omega-p}\binom{k+\ell}{p} / \binom{n}{\omega}} = \frac{\binom{n-k-\ell}{\omega-p+1}\binom{k+\ell}{p-1} \cdot k}{\binom{n-k-\ell}{\omega-p}\binom{k+\ell}{p}} \tag{3.7}$$

$$= \frac{(k-\ell-\omega+p) \cdot p \cdot k}{(\omega-p+1)(k+\ell-p+1)} = \Omega(\sqrt{k}). \tag{3.8}$$

Here the denominator states the probability of a permutation inducing the correct weight distribution on any of the k syndromes, while the numerator is the probability of success in any iteration of the MMT algorithm (compare to Equation (3.5)). Observe that the last equality follows from the fact, that $\omega = \Theta(\sqrt{k})$ and $p \ll \omega$ as well as $\ell \ll k$.

So far we showed, that our modification increases the list size of L_4 by a small factor of $2p_2$, while we enhance the probability of a good permutation for any of the given k instances by a factor of $\Omega(\sqrt{k})$. While in the case of Sterns' algorithm this is already enough to conclude that the overall speedup in this setting is $\Omega(\sqrt{k})$, as long as $p_2 \ll k$, for MMT/BJMM we also need to consider the reduced amount of representations. Note that the amount of representations decreases from an initial R to R_k, i.e., by a factor of

	Instance		$\log\left(\sqrt{k}\right)$	Speedup	
	k	ω		Stern	MMT
Challenge-1	451	30	4.41	4.88	4.96
Challenge-2	883	42	4.89	5.39	5.43
QC-2918	1459	54	5.26	5.77	5.77
BIKE-1	12323	134	6.79	7.58	7.47
BIKE-3	24659	199	7.29	8.00	7.55
BIKE-5	40973	264	7.66	8.32	8.06
HQC-1	17669	132	7.05	8.14	8.00
HQC-3	35851	200	7.56	8.55	8.39
HQC-5	57637	262	7.91	8.83	8.66

Table 3.3: Estimated DOOM_k speedups for Stern and MMT in the quasi-cyclic setting with double circulant codes ($n = 2k$).

$$\begin{aligned}\delta_R := \frac{R_k}{R} &= \frac{\binom{p-1}{p/2}\binom{k+\ell-p+1}{p_1-p/2}}{\binom{p}{p/2}\binom{k+\ell-p}{p_1-p/2}} \\ &= \frac{(k+\ell-p+1)\cdot p/2}{(k+\ell-p/2-p_1+1)\cdot p} = \frac{(k+\ell-p+1)}{2(k+\ell-p+1-\varepsilon)} \approx \frac{1}{2},\end{aligned}$$

Here $\varepsilon = p_1 - p/2$ is the amount of 1-entries added by BJMM to cancel out during addition, which is usually a small constant. Hence $\ell_1 := \log R$ in Algorithm 7 decreases by one. This in turn increases the time for computing the search-tree by a factor of at most two.

In summary, we obtain a speedup of $\delta_{\text{P}} = \Omega(\sqrt{k})$ on the probability while losing a factor of at most $\frac{\delta_{\text{L}}}{\delta_R} = 4p_2$ in the construction of the tree. Hence, for MMT/BJMM with $p_2 \ll \sqrt{k}$ this yields an overall speedup of $\Omega(\sqrt{k})$. □

We included the DOOM_k improvement in our estimator formulas for Stern as well as MMT. Table 3.3 shows the derived estimated speedups. As a result both algorithms Stern and MMT achieve comparable DOOM_k speedups slightly larger than $\sqrt{k}$. Additionally, we performed practical experiments on the instances listed as *Challenge-1* and *Challenge-2* to verify the estimates. Therefore, we solved these instances with MMT with and without the DOOM_k technique. Averaged over ten executions we find speedups of 4.99 and 5.40 respectively (closely matching 4.96 and 5.43 from Table 3.3).

3.6 Quasi-Cyclic Cryptanalysis

In the quasi-cyclic setting we obtained five new decoding records on decodingchallenge.org with our MMT implementation [ALL19], see Table 3.4. Instances are defined on [ALL19] for every ω using parameters $n = \omega^2 + 2$ and $k = \frac{n}{2}$, closely following the BIKE and HQC design.

QC-2918.

The largest instance we were able to solve has parameters $(n, k, w) = (2918, 1459, 54)$, and took us 3.33 days on our cluster. The optimal identified parameter set is $(\ell, \ell_1, p, p_2) = (21, 1, 3, 1)$, for which we estimated $2^{31.9}$ permutations. We were able to perform $2^{29.31}$ permutations per

n	k	ω	time (days)	CPU years	bit complexity
2118	1059	46	0.08	0.05	50.5
2306	1153	48	0.22	0.15	52.5
2502	1459	50	0.30	0.21	54.6
2706	1353	52	1.18	0.83	56.6
2918	1459	54	3.33	2.33	58.6

Table 3.4: Parameters of the largest solved BIKE/HQC instances, needed wallclock time, CPU years and bit complexity estimates.

day, resulting in an expected running time of 6.02 days. Our computation took only 55% of the expected time, which happens with probability 42%.

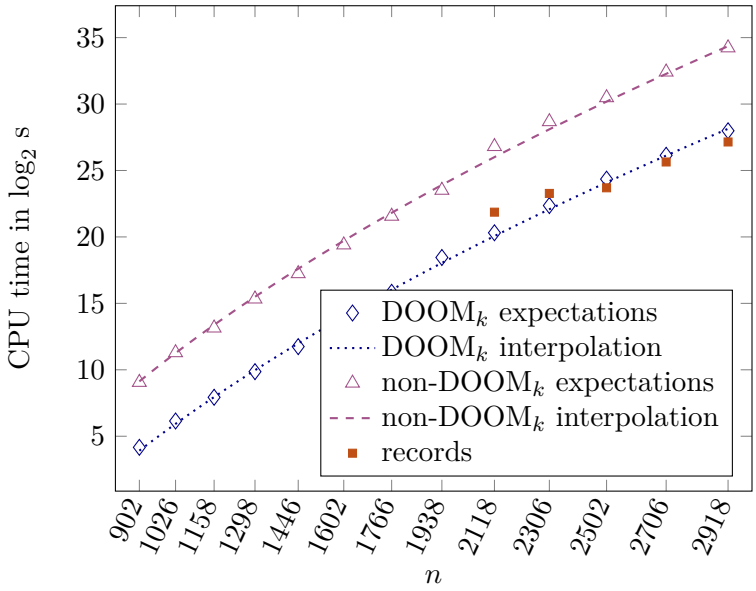


Figure 3.6: Estimated running times and interpolations for low- and high-memory configurations.

Interpolation.

In Figure 3.6 we give the expected running times that we obtained via benchmarking, both with (diamonds) and without (triangles) our DOOM$_k$ result from Section 3.5.1. Our five record computations are depicted as squares. All record computations where achieved in a runtime closely matching the expected values.

In the quasi-cyclic setting with rate $\frac{k}{n} = \frac{1}{2}$ and $\omega = \sqrt{n}$ Prange's runtime formula from Equation (3.2) gives $2^{\sqrt{n}}$. An interpolation of our experimental data points using the model $f(n) = a\sqrt{n} + b$ yields a best fit for

$$f(n) = 1.01\sqrt{n} - 26.42. \tag{3.9}$$

The slope $a = 1.01$ shows how accurately our MMT implementation matches the asymptotics already for medium sized instances, i.e., our MMT advantage and the polynomial runtime factors almost cancel out.

Concrete vs Asymptotic.

Similar to the McEliece setting in Section 3.4.3 and in Table 3.2, we also performed for BIKE/HQC an interpolation of estimated bit complexities in the logarithmic cost model using the algorithms of Prange, Stern and our MMT variant. We included instances up to code length $120,000$, reflecting the largest choice made by an HQC parameter set. As opposed to Section 3.4.3 we do not need addtional memory limitations, since none of the optimal configurations exceeds 2^{60}-bit of memory.

The interpolation with $f(n) = a\sqrt{n} + b$ gave us slopes of 1.054, 1.019 and 1.017 for Prange, Stern and MMT, respectively, i.e., all slopes are slightly above the asymptotic prediction of $a = 1$. Thus, as opposed to the McEliece setting our MMT benefits are canceled by polynomial factors.

Verification of the DOOM$_k$ Speedup.

From Figure 3.6, we can also experimentally determine the speedup of our DOOM$_k$ technique inside MMT. Lemma 1 predicts a speedup of $\sqrt{k} = \sqrt{n/2}$. Let $f(n) = 1.01\sqrt{n} - 26.42$ as before, and in addition take the model $f(n) + c \cdot \frac{\log(n/2)}{2}$ for non-DOOM$_k$. The new model should fit with $c = 1$.

The interpolation of our experimental non-DOOM$_k$ data, see the dashed line in Figure 3.6, yields $c = 1.17$. Thus, in practice we obtain a DOOM$_k$ speedup of $k^{0.58}$, slightly larger than $\sqrt{k}$.

3.7 Estimating Bit-Security for McEliece and BIKE/HQC

Based on our record computations, let us extrapolate to the hardness of breaking round-3 McEliece, BIKE and HQC. To provide precise statements about the security levels of proposed parameter sets, we also need to compare with the hardness of breaking AES. Recall that NIST provides five security level categories, where the most frequently used categories 1, 3, and 5 relate to AES. Category 1, 3, and 5 require that the scheme is as hard to break as AES-128, AES-192, and AES-256, respectively.

For AES we benchmarked the amount of encryptions per second on our cluster using the openssl benchmark software. The results for different key-lengths are listed in Table 3.5. For AES-192 and AES-256, we increased the blocklength from 128 to 256 bit, such that on expectation only a single key matches a known plaintext-ciphertext pair.

	AES-128	AES-192	AES-256
10^9enc/sec	2.16	0.96	0.83

Table 3.5: Number of AES encryptions per second performed by our cluster.

From Table 3.5 we extrapolate the running time to break AES-128, AES-192, and AES-256 on our hardware.

Extrapolation for McEliece and BIKE/HQC.

Let us detail our extrapolation methodology. We take as starting points the real runtimes of $22.04 = 2^{4.46}$ CPU years for McEliece-1284 and 2.33 CPU years for QC-2918.

Then we estimate by which factor it is harder to break the round-3 instances, and eventually compare the resulting runtime to the hardness of breaking AES.

Let us give a numerical example for McEliece-4608. Assume that we take 2^{60}-bit memory limitation for M, and we are in the most realistic logarithmic memory cost model. In this setting our estimator (without LSH) gives for $n = 4608$ a bit complexity of 187.72, and for $n = 1284$ a bit complexity of 65.27. Thus, it is a factor of $2^{122.45}$ harder to break McEliece-4608 than to break our record McEliece-1284. Therefore, we conclude that a break of McEliece-4608 on our hardware would require $2^{4.46} \cdot 2^{122.45} = 2^{126.91}$ CPU years.

In contrast, from Table 3.5 we conclude that breaking AES-192 on our hardware requires $2^{145.24}$ CPU years. Thus, from our extrapolation McEliece-4608 is a factor of $2^{18.33}$ *easier to break* than AES-192. This is denoted by -18.33 in Table 3.6.

McEliece Slightly Overestimates Security.

For completeness, we consider in Table 3.6 all three different memory-access cost models, constant, logarithmic and cube-root, even though we identified the *logarithmic* model as most realistic (compare to Section 3.4.2). Recall that in these models an algorithm with memory M suffers either no penalty (constant), a multiplicative factor of $\log M$ (logarithmic) or even a $\sqrt[3]{M}$ factor penalty (cube-root).

Moreover, we also provide memory limitations for the constant and logarithmic models. This is unnecessary in the cube-root model, in which no optimal parameter configuration exceeds a memory bit complexity of 60.

McEliece		Category 1 $n = 3488$	Category 3 $n = 4608$	Category 5a $n = 6688$	Category 5b $n = 6960$	Category 5c $n = 8192$
	unlimited	0.09	−24.86	−23.18	−23.80	6.10
constant	$M \leq 2^{80}$	1.54	−21.52	−11.67	−10.87	23.37
	$M \leq 2^{60}$	4.80	−19.12	− 3.86	− 3.80	32.70
	unlimited	1.77	−23.11	−20.70	−21.29	8.84
logarithmic	$M \leq 2^{80}$	2.86	−20.41	−10.46	− 9.63	24.64
	$M \leq 2^{60}$	5.55	−18.33	− 3.46	− 3.40	33.16
cube-root		10.37	−12.27	0.82	1.38	38.22

Table 3.6: Difference in bit complexity of breaking McEliece and corresponding AES instantiation under different memory access cost.

Let T_{McEliece} denote the extrapolated McEliece runtime, and let T_{AES} be the extrapolated AES runtime in the respective security category. Then Table 3.6 provides the entries $\log_2(\frac{T_{\text{McEliece}}}{T_{\text{AES}}})$. Thus, a negative x-entry indicates that this McEliece instance is x bits easier to break than its desired security category.

Whereas the Category 1 instance McEliece-3488 meets its security level in all memory models, the Category 3 instance McEliece-4608 misses the desired level by roughly 20 bits for constant/logarithmic costs. Even for cube-root costs McEliece-4608 is still 12 bits below the required level.

The Category 5a and 5b McEliece instances are in the realistic logarithmic model with 2^{80}-bit memory also 10 bits below their desired security level, whereas the Category 5c

McEliece instance is independent of the memory model above its security level.

BIKE / HQC			Category 1	Category 3	Category 5
constant	BIKE	message	2.44	2.50	3.49
		key	3.88	2.13	5.87
	HQC		1.24	4.28	2.23
logarithmic	BIKE	message	2.86	3.04	4.10
		key	4.42	3.11	6.74
	HQC		1.72	4.87	2.90
cube-root	BIKE	message	4.47	5.20	6.68
		key	5.77	5.00	9.03
	HQC		3.62	7.34	5.75

Table 3.7: Difference in bit complexity of breaking BIKE/HQC and corresponding AES instantiation under different memory access cost.

BIKE/HQC Accurately Matches Security.

In Table 3.7 we state our results for BIKE and HQC. As opposed to the McEliece setting we do not need memory limitations here, since none of the estimates exceeded 2^{60}-bit of memory.

Note that for BIKE we need to distinguish an attack on the key and an attack on a message. That is because recovering the secret key from the public key corresponds to finding a low-weight codeword, whereas recovering the message from a ciphertext corresponds to a syndrome decoding instance, where the syndrome is usually not the zero vector. Both settings allow for different speedups as outlined in Section 3.5.

We observe that the BIKE as well as the HQC instances precisely match their claimed security levels already in the conservative setting of constant memory access costs. Introducing memory penalties only leads to slight increases in the security margins.

4 New Time-Memory Trade-Offs for Subset Sum–Improving ISD in Theory and Practice

We propose new time-memory trade-offs for the random subset sum problem defined on $(a_1, \ldots, a_n, t)$ over $\mathbb{Z}_{2^n}$.

Our trade-offs yield significant running time improvements for every fixed memory limit $M \geq 2^{0.091n}$. Furthermore, we interpolate to the running times of the fastest known algorithms when memory is not limited. Technically, our design introduces a pruning strategy to the construction by Becker-Coron-Joux (BCJ) that allows for an exponentially small success probability. We compensate for this reduced probability by multiple randomized executions. Our main improvement stems from the clever reuse of parts of the computation in subsequent executions to reduce the time complexity per iteration.

As an application of our construction, we derive the first non-trivial time-memory trade-offs for Information Set Decoding (ISD) algorithms. Our new algorithms improve on previous (implicit) trade-offs asymptotically as well as practically. Moreover, our optimized implementation also improves on *running time*, due to reduced memory access costs. We demonstrate this by obtaining a new record computation in decoding quasi-cyclic codes (QC-3138). Using our newly obtained data points we then extrapolate the hardness of suggested parameter sets for the McEliece, BIKE and HQC schemes, lowering previous estimates by up to 6 bits.

The content of this chapter is the result of a collaboration with Andre Esser. It previously appeared as New Time-Memory Trade-Offs for Subset Sum–Improving ISD in Theory and Practice in EuroCrypt 2023 and is reproduced here with permission.

4.1 Introduction

For the ongoing NIST PQC standardisation process to be successful, large cryptanalytic efforts analysing the involved primitives are required. This includes theoretical studies of the asymptotically best attacks as well as experiments on a meaningful scale to safely extrapolate the hardness of cryptographic-sized instances. This methodology, combining theory and practice, is well established for conventional (number-theoretic) cryptographic systems and has found its adaptation to post-quantum secure schemes in recent years [EKM17, ADH+19, DSv21, UV21, EMZ22].

The best attacks on post-quantum schemes often suffer from high memory demands [BKW00, BDGL16, BJMM12, BBC+20, Din21]. This either leads to an immense slowdown of the algorithm due to physical access times or, in the worst case, prevents its application entirely. In practice, both cases usually lead to a fallback to more memory-efficient but asymptotically inferior procedures. In these cases *time-memory trade-offs* for the best algorithms are needed which allow to tailor their memory consumption to any given amount while (only slightly) increasing their running time.

For post-quantum secure candidates, especially from code- and lattice-families, several of the known attacks are build on techniques initially introduced in the context of the (random) subset sum problem [MMT11, BCDL19, May21, BDGL16]. This is because the underlying problems can usually be formulated as (vectorial) variants of subset sum, as it is the case for LPN / LWE, SIS or the syndrome decoding problem.

The subset sum problem defined on $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{2^n}$ asks to find a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = t \bmod 2^n$. For this problem time-memory trade-offs are actually well studied [DDKS12, AKKM13, Din18, HJ10]. However, the translations of those trade-offs to the aforementioned applications are mostly missing. The reason is the very diverse landscape of optimal trade-offs for subset sum, i.e., for different memory limitations there exist different optimal trade-offs. Furthermore, these trade-offs often do not match the design of the fastest subset sum algorithm used in the original application, which implies a separate translation effort for each algorithm.

In this work we construct new improved time-memory trade-offs for the subset sum problem. In contrast to previous works, our constructions follow the design by Becker-Coron-Joux (BCJ) [BCJ11], which is the basis for the fastest known algorithms. This allows for an easy adaptation of our trade-off to known applications of the BCJ algorithm. Further, our trade-offs reduce the running time of previous approaches for any fixed memory significantly. Only for very small available memory a trade-off based on a memory-less algorithm by Esser and May [EM20] becomes favourable. In total this reduces the trade-off landscape to only two algorithms.

We illustrate the potential of our trade-off by formalizing its application to the syndrome decoding problem, whose hardness forms the basis of code-based cryptography. Informally, the problem asks to find a low Hamming weight solution $\mathbf{e} \in \mathbb{F}_2^n$ to the matrix-vector equation $\mathbf{He} = \mathbf{s}$, where $\mathbf{H} \in \mathbb{F}_2^{r \times n}$ and $\mathbf{s} \in \mathbb{F}_2^r$. Moreover, it allows for a direct translation to a vectorial subset sum variant. Denote by $\mathbf{h}_i$ the columns of $\mathbf{H}$, then $(\mathbf{h}_1, \ldots, \mathbf{h}_n, \mathbf{s})$ defines a subset sum instance over $\mathbb{F}_2^n$, i.e., we are looking for a small subset of the $\mathbf{h}_i$ that sums to $\mathbf{s}$ over $\mathbb{F}_2^n$.

Information Set Decoding (ISD) algorithms now solve this problem by first applying a dimension reduction technique, which yields an instance with decreased n, r and smaller solution weight. Then an adaptation of the BCJ subset sum algorithm over $\mathbb{F}_2$ is applied to solve this reduced instance. Since the dimension reduction technique, in contrast to the subset sum algorithm, does not require any memory, every ISD algorithm inherits a naive time-memory trade-off. That is, reduce the instance size sufficiently so that the latter applied BCJ algorithm does not exceed the given memory. So far this simple interpolation to a full dimension-reduction based ISD algorithm proposed by Prange in 1962 [Pra62], was the best known trade-off strategy. Our adaptation now yields the first time-memory trade-offs for advanced ISD algorithms improving their performance asymptotically as well as in practice.

4.1.1 Related Work

Subset Sum. Any subset sum instance can be solved in time and memory $\tilde{\mathcal{O}}\left(2^{\frac{n}{2}}\right)$ via a meet-in-the-middle algorithm [HS74]. Schroeppel and Shamir [SS81] then showed how to reduce the memory complexity to $\tilde{\mathcal{O}}\left(2^{\frac{n}{4}}\right)$. Later, their technique formed the basis for a series of advanced time-memory trade-offs [DDKS12, Din18, DEM19].

The second key-ingredient for most subset sum trade-offs [EM20, DEM19, BCJ11, HJ10] is the so-called *representation technique* introduced by Howgrave-Graham and Joux (HGJ) in [HJ10]. In their work they constructed the first algorithm breaking the $2^{\frac{n}{2}}$ time bound for *random* subset sum instances by achieving running time $2^{0.337n}$. In the cryptographic setting

we usually encounter random instances, i.e., the vector $\mathbf{a} := (a_1, \ldots, a_n)$ is chosen uniformly at random and the target is set to $t = \langle \mathbf{a}, \mathbf{e} \rangle$ for a randomly chosen solution vector $\mathbf{e} \in \{0,1\}^n$ of Hamming weight $\frac{n}{2}$. Howgrave-Graham and Joux then split the solution $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ with $\mathbf{e}_i \in \{0,1\}^n$ of weight $n/4$. Now, there exist multiple, namely $\binom{n/2}{n/4}$, such *representations* of $\mathbf{e}$, i.e., different combinations $\mathbf{e}_1, \mathbf{e}_2$ that sum to $\mathbf{e}$. The core observation is that it suffices to find a single of these representations to recover the solution. This representation is then constructed using a search-tree imposing restrictions on the exact form of the solution (similar to Wagners k-tree algorithm [Wag02]) so that in expectation one representation satisfies all restrictions. Becker, Coron and Joux (BCJ) [BCJ11] improved the running time to $2^{0.291n}$ by choosing $\mathbf{e}_i \in \{-1, 0, 1\}^n$ to increase the amount of representations. Later Bonnetain, Bricout, Schrottenloher and Shen (BBSS) [BBSS20] further extended the digit set to $\mathbf{e}_i \in \{-1, 0, 1, 2\}^n$ yielding a time and memory complexity of $2^{0.283n}$.

Information Set Decoding. ISD algorithms are the fastest known algorithms to solve general instances of the syndrome decoding problem and form the basis in assessing the security of code-based schemes. Introduced originally by Prange [Pra62], the class was extended by several improved algorithms over the years [Dum91, Ste88, MMT11, BJMM12, MO15]. All these works improve the running time by using more advanced subset sum techniques to solve the reduced instance after dimension reduction, which simultaneously increases the memory requirements. Surprisingly, there has been very limited work on time-memory trade-offs for ISD algorithms. Karpman and Lefevre [KL22] recently constructed advanced time-memory trade-offs for the special case of decoding ternary codes based on a subset sum trade-off strategy known as Dissection [DDKS12]. Further, a work by Wang et al. [WL15] extends an early ISD algorithm from Stern [Ste88] by the Dissection approach. However, this trade-off is entirely outperformed by the previously mentioned implicit trade-offs of more advanced ISD procedures.

4.1.2 Our Contribution

Subset Sum. As a first contribution we give a generalized description of the BCJ algorithm, that combines previous interpretations from [BBSS20, EM20]. This description then forms the basis for one of our main contributions which are new time-memory trade-offs for the random subset sum problem. Our constructions yield significantly improved running times for every fixed memory $M \geq 2^{0.091n}$, which corresponds to more than two-thirds of the meaningful memory parameters. Recall that $M = 2^{0.283n}$ memory is sufficient to instantiate the fastest known algorithm with time complexity $T = M$. In Figure 4.1 we illustrate the performance of our new trade-offs in comparison to previous works. For example, if the memory is limited to $2^{0.17n}$, we improve the running time from $2^{0.51n}$ down to $2^{0.4n}$, corresponding to an improvement by a factor of $2^{0.11n}$.

From a technical side we allow the BCJ and BBSS construction to impose larger restrictions on the representation-space, yielding an exponentially small success probability. We then perform multiple randomized executions to compensate for the reduced probability. In this context we introduce a novel strategy of reusing lower levels of the search-tree in subsequent randomized executions to reduce the time complexity per iteration. In order to obtain instantiations for small memory parameters and to further reduce the time complexity, we integrate the Dissection framework [DDKS12] in our construction, inspired by the combination of Wagners k-tree and Dissection in [Din18].

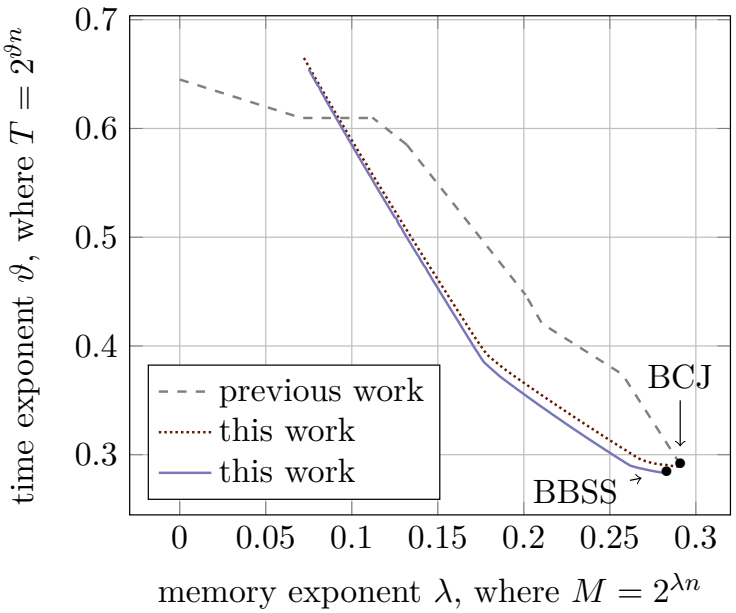


Figure 4.1: Our new subset sum trade-offs in comparison to the previously best known time-memory trade-offs. The solid line illustrates the minimum running time over the algorithms given in [HJ10, BCJ11, Ess20, EM20, DEM19, DDKS12]. For a memory larger than $2^{0.091n}$ ($2^{0.093n}$ resp.) our new trade-offs are superior to previous approaches.

Information Set Decoding. We give the first non-trivial time-memory trade-offs for advanced ISD algorithms by combining our trade-offs with the ISD algorithms by May-Meuer-Thomae (MMT) [MMT11] and Becker-Joux-May-Meurer (BJMM) [BJMM12]. Overall this yields asymptotic improved running times for every fixed memory. Moreover, for the MMT algorithm we are able to improve the memory, while maintaining its running time.

On the practical side, we extend the fastest implementation of the MMT / BJMM algorithm from [EMZ22] by our trade-off strategy observing memory and *time* improvements. Using our optimized implementation we obtain a new record computation in decoding quasi-cyclic codes (QC-3138) [ALL19] and re-break several old records, including the current record for McEliece-like decoding, consuming less resources, i.e., time and memory. Hence, our trade-off is the first asymptotic improvement of the MMT algorithm that transfers to the implementation level. Eventually, using our newly obtained data-points in combination with an estimation script [EB22] we extrapolate the hardness of suggested parameter sets for code-based NIST PQC (alternate) candidates McEliece, BIKE and HQC, resulting in reduced security estimates compared to previous works.

Outline. In Section 4.2 we set up necessary notation and cover some basics on the Dissection technique. Subsequently, in Section 4.3 we give the generalized description of the BCJ algorithm, which is then used as a basis to build our new trade-offs in Section 4.4. Eventually, in Section 4.5 we give the asymptotic and practical results of our decoding application including security estimates for code-based NIST PQC candidates.

4.2 Preliminaries

All logarithms are base 2. We define $H(x) := -x\log(x) - (1-x)\log(1-x)$ to be the binary entropy function with H^{-1} its inverse on $[0, \frac{1}{2}]$. Extending this definition, we also use the 2-way entropy function defined as $g(x, y) := -x\log(x) - y\log(y) - (1-x-y)\log(1-x-y)$. We simplify binomial and multinomial coefficients via Sterling's formula as

$$\binom{n}{\alpha n} \simeq 2^{nH(\alpha)} \text{ and } \binom{n}{\alpha n, \beta n, \cdot} \simeq 2^{ng(\alpha,\beta)},$$

where $\binom{n}{\alpha n, \beta n, \cdot} := \binom{n}{\alpha n, \beta n, (1-\alpha-\beta)n}$. We use standard landau notation, with $\tilde{\mathcal{O}}$-notation suppressing poly-logarithmic factors and write $A = \tilde{\mathcal{O}}(B)$ as $A \simeq B$. Our asymptotic complexity statements are all to be understood up to poly-logarithmic factors, even though we sometimes drop the $\tilde{\mathcal{O}}$ for convenience.

For a vector $\mathbf{x} \in \mathbb{F}_2^n$ we denote by $\mathrm{wt}(\mathbf{x})$ its Hamming weight. Additionally we denote by $\langle \mathbf{x}, \mathbf{y} \rangle$ the inner product of two vectors $\mathbf{x}, \mathbf{y}$.

All our algorithms target the random subset sum problem defined as follows, even if we might omit the term *random* sometimes.

Definition 9 (Random Subset Sum Problem)**.** Let $\mathbf{a} := (a_1, \ldots, a_n) \in \mathbb{Z}_{2^n}$ be drawn uniformly at random. For a random $\mathbf{e} \in \{0,1\}^n$ with $\mathrm{wt}(\mathbf{e}) = \frac{n}{2}$, let $t := \langle \mathbf{a}, \mathbf{e} \rangle$. The *random subset sum problem* is given $(\mathbf{a}, t)$ find any $\mathbf{e}' \in \{0,1\}^n$ satisfying $\langle \mathbf{a}, \mathbf{e}' \rangle = t$. We call any such $\mathbf{e}'$ a *solution.* and $(\mathbf{a}, t)$ an *instance.*

Our definition of the subset sum problem asks for a solution in $\{0,1\}^n$. However, algorithms like the BCJ algorithm approach the problem in a divide-and-conquer manner, which requires solving sub-instances with solutions in a different domain D. These sub-instances are usually solved via a meet-in-the-middle strategy, however, we employ more memory efficient strategies.

Schroeppel-Shamir and Dissection

While a standard meet-in-the-middle can solve a subset sum instance with solution in a set D in time and memory $|D|^{\frac{1}{2}}$ [HS74], the algorithm by Schroeppel and Shamir [SS81] achieves the same time complexity while improving the memory complexity to $|D|^{\frac{1}{4}}$. The Dissection framework introduced in [DDKS12] offers instantiations with less memory in form of a continues time-memory trade-off starting from the Schroeppel-Shamir algorithm. Besides the Schroeppel-Shamir algorithmm our constructions make use of another instantiation of this framework, a so-called 7-Dissection. A 7-Dissection runs in time $|D|^{4/7}$ and uses memory $|D|^{1/7}$. Moreover, with more memory its time complexity can be gradually decreased until it reaches the complexity of the Schroeppel-Shamir algorithm. We summarize this in the following lemma. For more details on the dissection framework the reader is referred to [DDKS12].

Lemma 2 (7-Dissection, [DDKS12])**.** *Let $\frac{1}{7} \leq \lambda \leq \frac{1}{4}$. The 7-Dissection algorithm finds all solutions $\mathbf{e} \in D$ to a random subset sum instance in expected time $|D|^{\frac{2(1-\lambda)}{3}}$ and expected memory $|D|^{\lambda}$.*

4.3 The generalized BCJ Algorithm

In this section we give a general description of the BCJ algorithm [BCJ11] for solving the random subset sum problem. This description forms the basis for our new trade-offs presented

in the following section. We advise the reader to follow Figure 4.2. In our exposition we assume a certain familiarity of the reader with the representation technique, otherwise we refer to [HJ10, BCJ11] for an introduction.

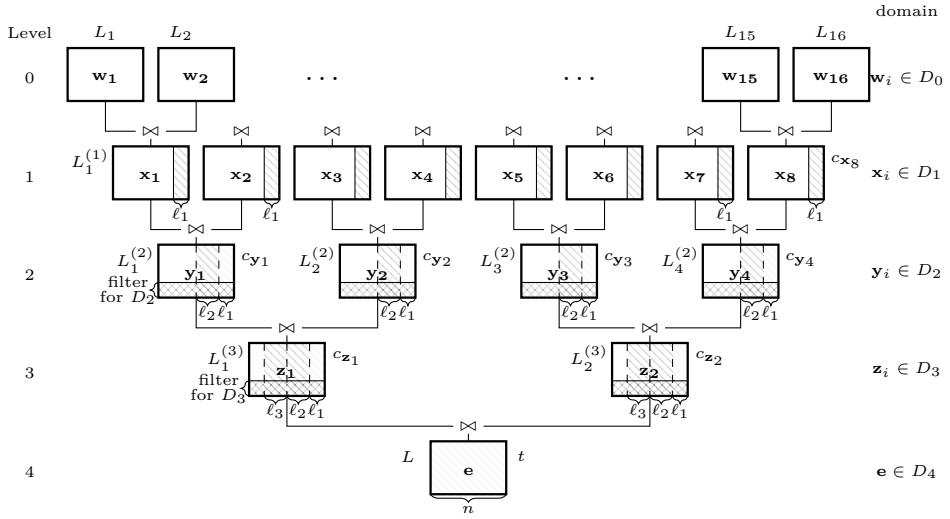


Figure 4.2: Generalized tree construction of the BCJ Algorithm in depth 4. Shaded areas on the right of a list L indicate that for all elements $\mathbf{v} \in L$ the inner product $\langle \mathbf{a}, \mathbf{v} \rangle$ matches a predefined value $c_{\mathbf{v}}$ (resp. t) on those bits.

Basic idea

To construct a solution $\mathbf{e}$ of the subset sum problem the BCJ algorithm splits $\mathbf{e}$ in the sum of two addends, i.e.,

$$\mathbf{e} = \mathbf{z}_1 + \mathbf{z}_2 \ .$$

Here the $\mathbf{z}_i$ are chosen from a set, such that there exist multiple different *representations* of the solution, i.e., different tuples that sum to $\mathbf{e}$. The goal is then to examine a respective fraction of the space of the $\mathbf{z}_1, \mathbf{z}_2$ to find one of these representations.

From $\langle \mathbf{a}, \mathbf{e} \rangle = \langle \mathbf{a}, \mathbf{z}_1 + \mathbf{z}_2 \rangle = t \mod 2^n$ we have by linearity

$$\langle \mathbf{a}, \mathbf{z}_1 \rangle = t - \langle \mathbf{a}, \mathbf{z}_2 \rangle \mod 2^n. \tag{4.1}$$

Note that the value of $\langle \mathbf{a}, \mathbf{z}_1 \rangle$ is not known. However, by considering only those $\mathbf{z}_1$ which fulfill $\langle \mathbf{a}, \mathbf{z}_1 \rangle = c_{\mathbf{z}_1} \mod 2^\ell$ for some fixed integer $c_{\mathbf{z}_1}$ we are able to impose a constraint on the search space. Here $\ell := \ell_1 + \ell_2 + \ell_3$ is an optimization parameter of the algorithm, with the ℓ_i's being positive integers. Moreover, since each representation of $\mathbf{e}$ fulfills Equation (4.1) the value of $c_{\mathbf{z}_2} := \langle \mathbf{a}, \mathbf{z}_2 \rangle = t - c_{\mathbf{z}_1} \mod 2^\ell$ is fully determined.

The construction of the $\mathbf{z}_1$ and $\mathbf{z}_2$ then works recursively. Therefore, they are split again in the sum of two addends

$$\mathbf{z}_1 = \mathbf{y}_1 + \mathbf{y}_2 \text{ and } \mathbf{z}_2 = \mathbf{y}_3 + \mathbf{y}_4 \ ,$$

and we fix the values $\langle \mathbf{a}, \mathbf{y}_1 \rangle$ and $\langle \mathbf{a}, \mathbf{y}_3 \rangle$ to some constraints $c_{\mathbf{y}_1} \bmod 2^{\ell_1+\ell_2}$ and $c_{\mathbf{y}_3} \bmod 2^{\ell_1+\ell_2}$. Note that this again determines the inner product of the remaining addends for any representation $(\mathbf{y}_1, \mathbf{y}_2)$ of $\mathbf{z}_1$ and $(\mathbf{y}_3, \mathbf{y}_4)$ of $\mathbf{z}_2$ as

$$c_{\mathbf{y}_2} := \langle \mathbf{a}, \mathbf{y}_2 \rangle = c_{\mathbf{z}_1} - c_{\mathbf{y}_1} \bmod 2^{\ell_1+\ell_2} \text{ and } c_{\mathbf{y}_4} := \langle \mathbf{a}, \mathbf{y}_4 \rangle = c_{\mathbf{z}_2} - c_{\mathbf{y}_3} \bmod 2^{\ell_1+\ell_2}$$

The recursion continues once more by splitting the $\mathbf{y}_i = \mathbf{x}_{2i-1} + \mathbf{x}_{2i}$ and introducing four additional modular constraints $c_{\mathbf{x}_{2i-1}} \bmod 2^{\ell_1}$. These modular constraints together with the $c_{\mathbf{y}_i}$'s determine the values of inner products $c_{\mathbf{x}_{2i}} := \langle \mathbf{a}, \mathbf{x}_{2i} \rangle \bmod 2^{\ell_1}$, since we have

$$\begin{aligned} c_{\mathbf{z}_2} &:= t - c_{\mathbf{z}_1} && \bmod 2^{\ell} \\ c_{\mathbf{y}_{2i}} &:= c_{\mathbf{z}_i} - c_{\mathbf{y}_{2i-1}} && \bmod 2^{\ell_1+\ell_2} && , i = 1, 2 \\ c_{\mathbf{x}_{2i}} &:= c_{\mathbf{y}_i} - c_{\mathbf{x}_{2i-1}} && \bmod 2^{\ell_1} && , i = 1, 2, 3, 4 \end{aligned} \tag{4.2}$$

Eventually, the $\mathbf{x}_i$'s are split in a meet-in-the-middle fashion, i.e.,

$$\mathbf{x}_i = (\mathbf{w}_{2i-1}, 0^{n/2}) + (0^{n/2}, \mathbf{w}_{2i}),$$

giving only a single representation of each $\mathbf{x}_i$.

The algorithm now starts by enumerating all possible values for the $\mathbf{w}_i$ in the base lists L_i. Then two lists are merged at a time in a new list by only considering those elements which fulfil the current constraint modulo 2^{ℓ_1}, $2^{\ell_1+\ell_2}$, 2^{ℓ} or 2^n respectively (compare to Figure 4.2). After the level-i list construction only those elements are kept whose coordinates follow a predefined distribution D_i, while all others are discarded. The choice of these distributions mainly determines the existing amount of representations and ultimately the performance of the algorithm. We give the pseudocode of the procedure in Algorithm 8.

Algorithm 8: BCJ Algorithm

Input : $\mathbf{a} \in (\mathbb{Z}_{2^n})^n, t \in \mathbb{Z}_{2^n}$
Output : $\mathbf{e} \in \mathbb{F}_2^n$ with $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$

1 Choose optimal ℓ_1, ℓ_2, ℓ_3 and D_i, $i = 0, 1, 2, 3$
2 Enumerate

$$L_{2i-1} = \{\mathbf{w}_{2i-1} \mid \mathbf{w}_{2i-1} \in D_0 \times 0^{n/2}\}$$
$$L_{2i} = \{\mathbf{w}_{2i} \mid \mathbf{w}_{2i} \in 0^{n/2} \times D_0\}, i = 1, \ldots, 8$$

3 Choose random $c_{\mathbf{z}_1} \in \mathbb{F}_2^{\ell}, c_{\mathbf{y}_1}, c_{\mathbf{y}_3} \in \mathbb{F}_2^{\ell_1+\ell_2}, c_{\mathbf{x}_1}, c_{\mathbf{x}_3}, c_{\mathbf{x}_5}, c_{\mathbf{x}_7} \in \mathbb{F}_2^{\ell_1}$
4 Set remaining constraints according to Equation (4.2)
5 Compute (and filter)

$$L_i^{(1)} = \{\mathbf{x}_i \mid \langle \mathbf{a}, \mathbf{x}_i \rangle = c_{\mathbf{x}_i} \bmod 2^{\ell_1}, \mathbf{x}_i = \mathbf{w}_{2i-1} + \mathbf{w}_{2i}\}$$
$$\text{from } L_{2i-1}, L_{2i},\ i = 1, \ldots, 8 \quad \text{, then filter such that } L_i^{(1)} \subseteq D_1$$
$$L_i^{(2)} = \{\mathbf{y}_i \mid \langle \mathbf{a}, \mathbf{y}_i \rangle = c_{\mathbf{y}_i} \bmod 2^{\ell_1+\ell_2}, \mathbf{y}_i = \mathbf{x}_{2i-1} + \mathbf{x}_{2i}\},$$
$$\text{from } L_{2i-1}^{(1)}, L_{2i}^{(1)},\ i = 1, \ldots, 4 \quad \text{, then filter such that } L_i^{(2)} \subseteq D_2$$
$$L_i^{(3)} = \{\mathbf{z}_i \mid \langle \mathbf{a}, \mathbf{z}_i \rangle = c_{\mathbf{z}_i} \bmod 2^{\ell}, \mathbf{z}_i = \mathbf{y}_{2i-1} + \mathbf{y}_{2i}\},$$
$$\text{from } L_{2i-1}^{(2)}, L_{2i}^{(2)},\ i = 1, 2 \quad \text{, then filter such that } L_i^{(3)} \subseteq D_3$$
$$L = \{\mathbf{e} \mid \langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n, \mathbf{e} = \mathbf{z}_{2i-1} + \mathbf{z}_{2i}\}$$
$$\text{from } L_1^{(3)}, L_2^{(3)} \quad \text{, then filter such that } L \subseteq \{0, 1\}^n$$

return $\mathbf{e} \in L$

Complexity

Let the expected list sizes *before* filtering on level i be $\mathcal{L}_i$ and let the probability of any element of a level-i list surviving the filter be q_i. Since the level-1 lists are constructed from the Cartesian product of the level-0 lists by enforcing a modular constrained on ℓ_1 bits we have

$$\mathcal{L}_1 = \frac{(\mathcal{L}_0)^2}{2^{\ell_1}}.$$

Analogously the level-2 lists are constructed from the *filtered* level-1 lists by enforcing a modular constrained on $\ell_1 + \ell_2$ bits. However, since the last ℓ_1 bits are already fixed to some value in the previous step we only enforce a new constraint on ℓ_2 bits, which results in

$$\mathcal{L}_2 = \frac{(q_1 \cdot \mathcal{L}_1)^2}{2^{\ell_2}}.$$

Analogously we obtain

$$\mathcal{L}_3 = \frac{(q_2 \cdot \mathcal{L}_2)^2}{2^{\ell_3}} \text{ and } \mathcal{L}_4 = \frac{(q_3 \cdot \mathcal{L}_3)^2}{2^{n-\ell}}.$$

The construction of each unfiltered list can be performed via hashing in time linear in the list's sizes giving an expected time complexity of

$$T = \max_i(\mathcal{L}_i) \ .$$

Since we need to store only filtered lists and the filtering can be performed on-the-fly the memory complexity becomes $M = \max_i(q_i \cdot \mathcal{L}_i)$.

Correctness

Obviously the constraint's sizes ℓ_1, ℓ_2 and ℓ_3 cannot be chosen arbitrarily large if one representation of the solution should survive all imposed constraints. On the other hand we need to ensure that multiple representations do not lead to the construction of duplicate elements in intermediate lists to ensure a proper list distribution. This leads to further restrictions on the size of ℓ_1, ℓ_2 and ℓ_3, called *saturation constraints* in [BBSS20] or simply lower bounds in [EM19].

In [BBSS20] this is formalized by ensuring that each list after filtering at every level is not larger than the size of the set filtered for, reduced by the total enforced constraint. Since by the randomness of the instance the elements are expected to distribute uniformly, it follows that the lists will not contain duplicate elements with high probability. The sets for which we filter on level i are D_i, $i = 1, 2, 3, 4$. Note that the choice of the sets D_i, $i \neq 4$ can be optimized, while the set D_4 has to describe the valid set of solutions, which is the set of binary vectors of length n.

Hence, to guarantee that there are no duplicates present in the level-1, level-2 and level-3 lists we need to ensure that

$$q_1 \cdot \mathcal{L}_1 \leq \frac{|D_1|}{2^{\ell_1}} \text{ and } q_2 \cdot \mathcal{L}_2 \leq \frac{|D_2|}{2^{\ell_1+\ell_2}} \text{ and } q_3 \cdot \mathcal{L}_3 \leq \frac{|D_3|}{2^{\ell}} \tag{4.3}$$

Next let us write the probabilities q_i in terms of representations and distributions. Therefore, let 2^{r_i} denote the amount of different representations of any element from D_{i+1} as the sum of two elements from D_i. Then we have

$$q_{i+1} = \frac{|D_{i+1}| \cdot 2^{r_i}}{|D_i|^2}, \tag{4.4}$$

describing the probability that a random sum of two elements from D_i forms a representation of any element from D_{i+1}.

Recall that we construct level-1 elements $\mathbf{x}_i = (\mathbf{w}_{2i-1}, \mathbf{w}_{2i-1}) \in D_0 \times D_0 = D_1$ in a meet-in-the-middle fashion from level-0 elements $\mathbf{w}_j$, which implies $\mathcal{L}_0 = \sqrt{|D_1|}$. As this gives only a single representation of any level-1 element, we have $r_0 = 0$, which leads to $q_1 = 1$, i.e., for this choice of D_0 there is no filtering on level one. It follows that the first saturation constraint from Equation (4.3) is always fulfilled since

$$q_1 \cdot \mathcal{L}_1 = \frac{(\mathcal{L}_0)^2}{2^{\ell_1}} = \frac{|D_1|}{2^{\ell_1}}.$$

The second constraint of Equation (4.3) gives

$$q_2 \cdot \mathcal{L}_2 = \frac{|D_2| \cdot 2^{r_1}}{|D_1|^2} \cdot \frac{(\mathcal{L}_0)^4}{2^{2\ell_1+\ell_2}} \overset{!}{\leq} \frac{|D_2|}{2^{\ell_1+\ell_2}} \Leftrightarrow r_1 \leq \ell_1.$$

Analogously we get from the last saturation constraint

$$q_3 \cdot \mathcal{L}_3 = q_3 \cdot \frac{(q_2)^2 \cdot (\mathcal{L}_1)^4}{2^{4\ell_1+2\ell_2+\ell_3}} = \frac{2^{2r_1+r_2} \cdot |D_3|}{2^{4\ell_1+2\ell_2+\ell_3}} \overset{!}{\leq} \frac{|D_3|}{2^{\ell}} \Leftrightarrow 2r_1 + r_2 \leq 3\ell_1 + \ell_2.$$

Eventually, to find exactly one representation of the solution in the final list we need to ensure that $q_4 \cdot \mathcal{L}_4 = 1$, which yields

$$\begin{aligned} q_4 \cdot \mathcal{L}_4 = \frac{q_4 \cdot (q_3)^2 \cdot (q_2)^4 (\mathcal{L}_1)^8}{2^{n+7\ell_1+3\ell_2+\ell_3}} &= \frac{2^{4r_1+2r_2+r_3} \cdot |D_4|}{2^{n+7\ell_1+3\ell_2+\ell_3}} \overset{!}{=} 1 \\ \Leftrightarrow \quad 4r_1 + 2r_2 + r_3 &= 7\ell_1 + 3\ell_2 + \ell_3, \end{aligned} \tag{4.5}$$

since we have $|D_4| = 2^n$, as D_4 is the set of binary vectors of length n.

Instantiation

The description of the general BCJ algorithm gives several degrees of freedom, including the choice of sets D_i, $i = 1, 2, 3$ and the size of the constraints ℓ_1, ℓ_2, ℓ_3. The original BCJ algorithm restricts all D_i's to include only vectors with coordinates in $\{0, \pm 1\}$. The purpose of including -1's is simply to increase the number of representations. Since the final goal is to construct a binary vector, minus one entries are supposed to cancel out with one entries in the addition. Thus, the distribution D_3 is chosen as vectors of length n with exactly $\omega_3 := n/4 + \alpha_3$ one entries and $m_3 := \alpha_3$ minus one entries for some small α_3, which has to be optimized. The distribution D_2 is then composed similarly as vectors of length n with $\omega_2 := \omega_3/2 + \alpha_2$ one entries and $m_2 := m_3/2 + \alpha_2$ minus one entries, where again α_2 minus ones are supposed to cancel out. Analogously the level-1 distribution is chosen as vectors of length n with $\omega_1 := \omega_2/2 + \alpha_1$ one entries and $m_1 := m_2/2 + \alpha_1$ minus one entries, expecting α_1 cancellations. An overview of this choice of distributions is given in Table 4.1. The size of these sets is

$$|D_i| = \binom{n}{\omega_i, m_i, \cdot} \simeq 2^{g(\frac{\omega_i}{n}, \frac{m_i}{n})n},$$

while the number of representations is given as

$$2^{r_{i-1}} = \binom{\omega_i}{\omega_i/2} \binom{m_i}{m_i/2} \binom{n-\omega_i-m_i}{\alpha_{i-1}, \alpha_{i-1}, \cdot} \simeq 2^{\omega_i + m_i + \rho_i},$$

		D_4	D_3	D_2	D_1
BCJ	ω_i	$\frac{1}{2}$	$\frac{1}{4}+\alpha_3$	$\frac{1}{8}+\frac{\alpha_3}{2}+\alpha_2$	$\frac{1}{16}+\frac{\alpha_3}{4}+\frac{\alpha_2}{2}+\alpha_1$
	m_i	0	α_3	$\frac{\alpha_3}{2}+\alpha_2$	$\frac{\alpha_3}{4}+\frac{\alpha_2}{2}+\alpha_1$
BBSS	ω_i	$\frac{1}{2}$	$\frac{1}{4}+\alpha_3-\gamma_3$	$\frac{1}{8}+\frac{\alpha_3-\gamma_3}{2}+\alpha_2-\gamma_2$	$\frac{1}{16}+\frac{\alpha_3-\gamma_3}{4}+\frac{\alpha_2-\gamma_2}{2}+\alpha_1-\gamma_1$
	m_i	0	α_3	$\frac{\alpha_3}{2}+\alpha_2$	$\frac{\alpha_3}{4}+\frac{\alpha_2}{2}+\alpha_1$
	c_i	0	γ_3	$\frac{\gamma_3}{2}+\gamma_2$	$\frac{\gamma_3}{4}+\frac{\gamma_2}{2}+\gamma_1$

Table 4.1: Choices of D_i made by BCJ and BBSS algorithm. The table states the proportion of coordinates equal to 1 (ω_i), -1 (m_i) and 2 (c_i). The proportion of zeros is $1-\omega_i-m_i-c_i$. Set D_0 has half the proportions of D_1.

where $\rho_i := g\left(\frac{\alpha_i}{n-\omega_i-m_i}, \frac{\alpha_i}{n-\omega_i-m_i}\right)(n-\omega_i-m_i)$.

Here the two binomial coefficients count the number of times the one and minus one entries of an element from D_i can be distributed equally over a sum of two elements. The multinomial coefficient then counts the number of possibilities how the minus one entries can cancel out.

Note that the algorithm splits D_1 into $D_0 \times D_0$, where D_0 is the set of vectors of length $n/2$ containing exactly $\omega_1/2$ ones and $m_1/2$ minus ones. This leads to all D_i only including balanced elements, i.e., elements which contain an equal amount of ones (resp. minus ones) on their first and second half of the coordinates. However, this affects the sizes of the D_i and the amount of representations only by a polynomial factor, which is subsumed in the landau notation.

Eventually, the BCJ algorithm chooses $\ell_1 = r_1$ and $\ell_2 = r_2 - r_1$, which yields $\ell_3 = r_3 - r_2$. A numerical optimization of the α_i results in a time complexity of $2^{0.291n}$ (originally reported as $2^{0.292n}$) for the BCJ configuration.

Bonnetain et al. [BBSS20] then showed that a more flexible choice of ℓ_1 and ℓ_2 and correspondingly adapted ℓ_3 allows to decrease the time complexity to $2^{0.289n}$. They also showed that extending the digit set of the D_i to $\{0, \pm 1, 2\}$ allows to further decrease the time complexity to $2^{0.283n}$, yielding the best known time complexity for the random subset sum problem.

The BCJ algorithm achieves optimal time complexity for a depth of the search-tree of four. However, in general the optimal depth varies with the application. We therefore give for completeness and later reference the complexity and saturation constraints for variable depth in Section 4.6.

4.4 Our new Subset Sum Trade-Off

The (generalized) BCJ algorithm from the previous section already inherits some time-memory trade-off potential. That is, one can try to optimize the choice of the ℓ_i with respect to the memory usage, since the larger the ℓ_i the smaller the list's sizes. However, the overall size of the ℓ_i's is bounded by the restriction that the last list should contain a representation of the solution.

On a high level our new trade-off works by relaxing this restriction, i.e. we do not require the last list to contain a solution. This allows to balance the lists more memory-friendly. We then perform multiple randomized executions of the algorithm to ensure that we find a solution overall. However, let alone this is not sufficient to obtain our improvements. The

main runtime advantage of our improved trade-off comes from our observation that we can reuse parts of the tree in subsequent randomized executions, reducing the cost per iteration. A second improvement stems from our use of the Dissection framework [DDKS12] for the construction of the level-1 lists.

Note that if we change some bit-constraints in the tree (the values of $c_{\mathbf{v}}$ in Figure 4.2) not necessarily all levels are affected. That means we do not need to re-compute all lists of the tree, but only those which depend on the changed constraints. Now, if the computation of each list had the same complexity, this strategy would not yield an improvement. However, by adapting parameters accordingly and exploiting the involved filtering, we can guarantee that the creation of frequently reconstructed lists (from already existing lists) is much cheaper than a reconstruction of the whole tree. This partial reconstruction strategy in combination with relaxing the correctness constraint from Equation (4.5) allows us to obtain significant improvements for rather high memory parameters $M \geq 2^{0.169n}$.

From there on the base lists, which are so far a meet-in-the-middle split of the first level domains start dominating the memory. The only possibility for the algorithm to decrease the size of those lists is to choose a set D_1 with smaller size on level 1. For the BCJ algorithm this means including less -1 entries, until ultimately no -1 entries are included in the enumeration. In this case the base lists require a memory of $\binom{n/2}{n/32} \simeq 2^{0.169n}$. From there the list sizes are as small as possible and we can not obtain instantiations for less memory. We circumvent this problem by exchanging the meet-in-the-middle strategy for exhaustive examination of the level-1 domain by the 7-Dissection algorithm. We find that apart from offering instantiations for smaller memory parameters this gives also time improvements as the optimization can choose a more optimal, usually larger set D_1 (implying larger D_0) without exceeding the memory limit.

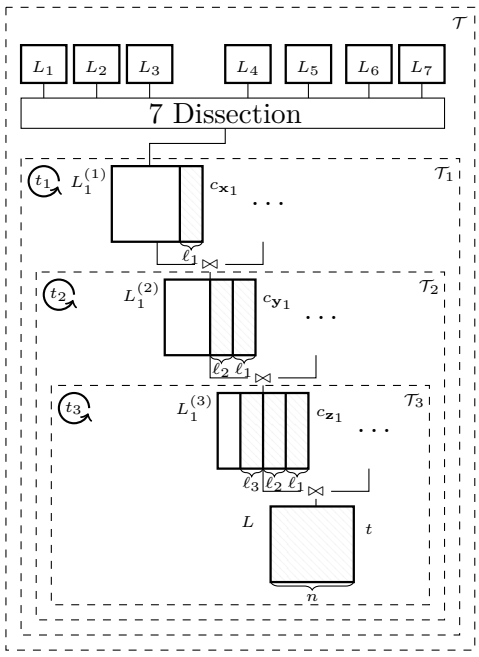


Figure 4.3: Our new trade-off in depth 4. Dashed boxes frame different subtrees $\mathcal{T}_i$, which are rebuild 2^{t_i} times. The level-1 lists are constructed using the 7-Dissection algorithm.

Adaptation of the BCJ Algorithm

We advise the reader to follow Figure 4.3. Let $\mathcal{T}$ be the full tree and $\mathcal{T}_i, i = 1, 2, 3$ the subtrees only including the lists from level i onwards. We denote by 2^{t_i} the number of times we rebuild the subtree $\mathcal{T}_i$ from the (already existing) lists of the previous level.

We start by changing only the upper ℓ_3 bits of the modular constraint $c_{\mathbf{z}_1}$ which requires recomputing only the subtree $\mathcal{T}_3$, since the level-i lists for $i \leq 2$ do not depend on these bits. Since there are only 2^{ℓ_3} choices for those bits we have $t_3 \leq \ell_3$. If 2^{ℓ_3} iterations are not sufficient to find the solution we start modifying the upper ℓ_2 bits of the modular constraints $c_{\mathbf{y}_1}$, $c_{\mathbf{y}_2}, c_{\mathbf{z}_1} \bmod 2^{\ell_2}$. This implies again that $t_2 \leq 3\ell_2$. Still, for every different choice of those bits we recompute the subtree $\mathcal{T}_3$ another 2^{t_3} times for different choices of the upper ℓ_3 bits. If $2^{3\ell_2+\ell_3}$ iterations are still not sufficient to find a solution, we eventually start modifying the lower ℓ_1 bits of the chosen modular constraints. Again for each choice of lower bits we reconstruct the tree $\mathcal{T}_2$ and $\mathcal{T}_3$ several times. Furthermore, as there are seven constraints that can be freely chosen we have $t_1 \leq 7\ell_1$

Finally, instead of computing the level-1 lists via a meet-in-the-middle algorithm we now use the 7-Dissection algorithm.

The pseudocode of our modified BCJ trade-off is given in Algorithm 9.

Complexity. The memory complexity stays as before with the only difference that the memory requirement of the base lists is now substituted by the memory requirement M_{7D} of the 7-Dissection algorithm, i.e.,

$$M = \max(M_{7D}, q_1\mathcal{L}_1, q_2\mathcal{L}_2, q_3\mathcal{L}_3, q_4\mathcal{L}_4).$$

To balance the memory requirement we instantiate the 7-Dissection algorithm with $M_{7D} = |D_1|^{\max\left(\frac{1}{7}, \lambda'\right)}$ memory where $|D_1|^{\lambda'} = \max_i(q_i L_i)$.

The analysis for the time complexity is also similar to before, with the essential difference that the three subtrees are now computed differently many times.

A single construction of subtree $\mathcal{T}_1$ can be performed in time

$$T_1 = \max(T_{7D}, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4),$$

where T_{7D} is the time it takes to compute the level-1 lists via the 7-Dissection algorithm. Recall that instantiated with $|D_1|^{\delta}$ memory, the 7-Dissection runs in time $T_{7D} = |D_1|^{\max\left(\frac{2(1-\delta)}{3}, \frac{1}{2}\right)}$ (compare to Lemma 2). The subtrees $\mathcal{T}_2$ and $\mathcal{T}_3$ can then be computed in time

$$T_2 = \max(q_1 \cdot \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4) \text{ and } T_3 = \max(q_2 \cdot \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4),$$

as they can be computed from the stored and already filtered level-1 respectively level-2 lists. Now the total time complexity becomes

$$T = \max(2^{t_1} \cdot T_1, 2^{t_1+t_2} \cdot T_2, 2^{t_1+t_2+t_3} \cdot T_3),$$

as subtree $\mathcal{T}_i$ is rebuild $2^{t_1+\ldots+t_i}$ many times.

Correctness. Most of the correctness follows from the correctness of the BCJ algorithm and the 7-dissection algorithm. Note that we instantiate the 7-Dissection with at least $|D_1|^{\frac{1}{7}}$ memory, which is the minimum requirement given by Lemma 2.

The main difference to before is that we relaxed the restriction given in Equation (4.5), such that the last list is not guaranteed to contain a solution anymore. However, we compensate

Algorithm 9: BCJ Trade-Off

Input : $\mathbf{a} \in (\mathbb{Z}_{2^n})^n, t \in \mathbb{Z}_{2^n}$
Output : $\mathbf{e} \in \{0,1\}^n$ with $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$

Choose optimal ℓ_1, ℓ_2, ℓ_3 and $D_i, i = 1,2,3$, let $r := r_3 + 2r_2 + 4r_1$
repeat $2^{t_1} := 2^{\max(7\ell_1 - r, 0)}$ **times**
  Choose random $c_{\mathbf{z}_1} \in \mathbb{F}_2^{\ell}, c_{\mathbf{y}_1}, c_{\mathbf{y}_3} \in \mathbb{F}_2^{\ell_1+\ell_2}, c_{\mathbf{x}_1}, c_{\mathbf{x}_3}, c_{\mathbf{x}_5}, c_{\mathbf{x}_7} \in \mathbb{F}_2^{\ell_1}$
  Set remaining constraints according to Equation (4.2)
  Compute

$$L_i^{(1)} = \{\mathbf{x}_i \mid \langle \mathbf{a}, \mathbf{x}_i \rangle = c_{\mathbf{x}_i} \bmod 2^{\ell_1}, \mathbf{x}_i \in D_1, \}, \quad \text{via 7-Dissection}, \quad i = 1, \ldots, 8$$

  repeat $2^{t_2} := 2^{\max(7\ell_1 + 3\ell_2 - r, 0) - t_1}$ **times**
    Choose randomly the upper ℓ_2 bits of $c_{\mathbf{z}_1}, c_{\mathbf{y}_1}, c_{\mathbf{y}_3} \bmod 2^{\ell_1+\ell_2}$
    Update $c_{\mathbf{z}_2}, c_{\mathbf{y}_2}, c_{\mathbf{y}_4}$ according to Equation (4.2)
    Compute

$$L_i^{(2)} = \{\mathbf{y}_i \mid \langle \mathbf{a}, \mathbf{y}_i \rangle = c_{\mathbf{y}_i} \bmod 2^{\ell_1+\ell_2}, \mathbf{y}_i \in D_2, \mathbf{y}_i = \mathbf{x}_{2i-1} + \mathbf{x}_{2i}\}, \quad \text{from } L_{2i-1}^{(1)}, L_{2i}^{(1)}, \quad i = 1, \ldots, 4$$

    repeat $2^{t_3} := 2^{\max(7\ell_1 + 3\ell_2 + \ell_3 - r, 0) - t_1 - t_2}$ **times**
      Choose randomly the upper ℓ_3 bits of $c_{\mathbf{z}_1}$
      Update $c_{\mathbf{z}_2}$ according to Equation (4.2)
      Compute

$$L_i^{(3)} = \{\mathbf{z}_i \mid \langle \mathbf{a}, \mathbf{z}_i \rangle = c_{\mathbf{z}_i} \bmod 2^{\ell}, \mathbf{z}_i \in D_3, \mathbf{z}_i = \mathbf{y}_{2i-1} + \mathbf{y}_{2i}\}, \quad \text{from } L_{2i-1}^{(2)}, L_{2i}^{(2)}, \quad i = 1, 2$$

$$L = \{\mathbf{e} \mid \langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n, \mathbf{e} \in D_4, \mathbf{e} = \mathbf{z}_{2i-1} + \mathbf{z}_{2i}\}, \quad \text{from } L_1^{(3)}, L_2^{(3)}$$

      if $|L| > 0$ **then**
        return $\mathbf{e} \in L$

for this by multiple randomized constructions of the final list. In contrast to completely independent executions of the algorithm, which would select all constraints uniformly at random, we only randomize the constraints affecting certain subtrees. However, note that under the standard assumption that the representations distribute independently and uniformly over all constraints, any set of constraints has the same independent probability of leading to a representation of the solution. Now, since we change at least one constraint for every reconstruction of the final list, we can treat the iterations as independent.

In order to ensure that over all iterations we find at least one representation, the final list's size accumulated over all its reconstructions must be at least one, which leads to (compare to Equation (4.5))

$$\begin{aligned} & q_4 \cdot \mathcal{L}_4 \cdot 2^{t_1+t_2+t_3} && \geq 1 \\ \Leftrightarrow\ & 4r_1 + 2r_2 + r_3 + t_1 + t_2 + t_3 && \geq 7\ell_1 + 3\ell_2 + \ell_3. \end{aligned}$$

Note that this constraint is fulfilled for our choice of

$$\begin{aligned} t_1 &= \max(7\ell_1 - r, 0) \\ t_2 &= \max(7\ell_1 + 3\ell_2 - r, 0) - t_1 \\ t_3 &= \max(7\ell_1 + 3\ell_2 + \ell_3 - r, 0) - t_1 - t_2, \end{aligned}$$

where $r := r_3 + 2r_2 + 4r_1$ and the maximum is needed since we need to build each subtree at least once.

Configuration of our Trade-Off

In terms of distributions we adapt the choice of the original BCJ algorithm, specified in Table 4.1. We then optimize the parameters $\alpha_i, \ell_i, i = 1, 2, 3$ numerically. We optimize such that the time is minimized, while simultaneously ensuring that the saturation constraints are satisfied and a given memory limit of $M = 2^{\lambda n}$ is not exceeded.

The resulting trade-off curve is depicted in Figure 4.1. We observe that our trade-off outperforms all existing approaches for $M \geq 2^{0.093n}$. Prior to our work, this interval was covered by a diverse landscape of different trade-offs including [HJ10, BCJ11, Ess20, EM20, DEM19, DDKS12]. For $M < 2^{0.093n}$ a trade-off given in [Ess20] based on a memory-free algorithm by Esser-May [EM20] becomes superior to our procedure.

Extending the digit set. We also adopted the choice of distributions made by the BBSS algorithm [BBSS20] (see Table 4.1). We find that the refined choice of the D_i gives an overall slight improvement, interpolating smoothly to their $2^{0.283n}$ algorithm. The resulting trade-off curve is depicted in Figure 4.1 as well, which remains superior to [EM20, Ess20] as long as $M \geq 2^{0.091n}$.

4.5 Application to Decoding Linear Codes

A linear code $\mathcal{C} \subset \mathbb{F}_2^n$ is a k-dimensional subspace of $\mathbb{F}_2^n$ and can be efficiently described via a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, such that $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n \mid \mathbf{Hc} = \mathbf{0}\}$. Decoding an error-prone codeword $\mathbf{y} := \mathbf{c} + \mathbf{e}$ to $\mathbf{c}$ is equivalent to recovering $\mathbf{e}$ from the so-called *syndrome* $\mathbf{s} := \mathbf{Hy} = \mathbf{H}(\mathbf{c} + \mathbf{e}) = \mathbf{He}$. This leads to the following definition of the syndrome decoding problem.

Definition 10 (Syndrome Decoding Problem)**.** Let $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ be the parity check matrix of a code of length n and dimension k, with constant *code-rate* $R := \frac{k}{n}$. Given a syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$ and an integer ω the *syndrome decoding problem* asks to find a vector $\mathbf{e} \in \mathbb{F}_2^n$ of Hamming weight $\mathrm{wt}(\mathbf{e}) = \omega$ satisfying $H\mathbf{e} = \mathbf{s}$.

Note that the problem admits a unique solution as long as $\omega \leq \lfloor \frac{d-1}{2} \rfloor$, where d is the minimum distance of the code, i.e., the minimum Hamming distance between two codewords. We call the setting with unique solution *half distance decoding*, while for $\omega \leq d$ we refer to *full distance decoding*. In general the time complexity increases with ω, such that we only consider the cases where ω is equal to those upper bounds. Further, random linear codes are known to achieve a minimum distance that is equal to the Gilbert-Varshamov bound of $d \approx H^{-1}(1 - \frac{k}{n})n$, i.e., the minimum distance is a function of the rate $R := \frac{k}{n}$ and the code-length n. We now maximize the complexity in our asymptotic analysis over all constant rates R to obtain a runtime formula which only depends on n.

The best known algorithms to solve the syndrome decoding problem are Information Set Decoding (ISD) algorithms. In the full and half distance setting these algorithms have exponential time and memory complexity of the form $\tilde{\mathcal{O}}\left(2^{cn}\right)$ for some constant c depending on the algorithm. On the other hand, cryptographic applications usually use a a sublinear weight, i.e., $\omega = o(n)$. In these cases the running time of ISD algorithms is subexponential of the form $\tilde{\mathcal{O}}\left(2^{c\omega}\right)$ for some constant c. Moreover, it was shown [TS16b] that in this case all known ISD algorithms converge to the same running time, i.e., they obtain the same constant c. However, in practical experiments advanced ISD algorithms were shown to provide significant speedups [BLP08, EMZ22].

We therefore first analyse our trade-offs in the full and half distance decoding setting, which allow to easily verify their superiority since they obtain improved constants c. We then study the practical effect of our trade-offs by providing an optimized implementation. Finally, we extrapolate the hardness of cryptographic schemes using our obtained data points.

Information Set Decoding

Information Set Decoding algorithms first apply a permutation matrix $\mathbf{P}$ to the columns of the parity check matrix. This allows to redistribute the weight on the error since the permuted instance $\mathbf{H}' := \mathbf{HP}$ has as valid solution $\mathbf{e}' := \mathbf{P}^{-1}\mathbf{e}$, since $\mathbf{HP}(\mathbf{P}^{-1}\mathbf{e}) = \mathbf{s}$. Then $\mathbf{H}'$ is transformed into semi-systematic form via Gaussian elimination modelled via the multiplication with an invertible matrix $\mathbf{Q}$

$$\mathbf{QH}'(\mathbf{P}^{-1}\mathbf{e}) = \begin{pmatrix} \mathbf{I}_{n-k-\ell} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} (\mathbf{e}_1, \mathbf{e}_2) = (\mathbf{e}_1 + \mathbf{H}_1\mathbf{e}_2, \mathbf{H}_2\mathbf{e}_2) = \mathbf{Qs} = (\mathbf{s}_1, \mathbf{s}_2), \tag{4.6}$$

where we write $\mathbf{e}' := \mathbf{P}^{-1}\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{k+\ell}$ with ℓ an optimization parameter of the algorithm. Let us further assume that the permutation distributes the weight on $\mathbf{e}'$ such that $\mathrm{wt}(\mathbf{e}_1) = \omega - p$ and $\mathrm{wt}(\mathbf{e}_2) = p$, for some value p to be optimized, too.

Now Equation (4.6) yields a (dimension) reduced syndrome decoding instance in form of the equation $\mathbf{H}_2\mathbf{e}_2 = \mathbf{s}_2$ with weight-p solution $\mathbf{e}_2 \in \mathbb{F}_2^{k+\ell}$. Usually, $\mathbf{e}_2$ is not a unique solution to this reduced instance. The algorithm therefore computes all solutions $\mathbf{x}$ to this smaller instance and checks if the corresponding $\mathbf{e}_1 = \mathbf{s}_1 + \mathbf{H}_1\mathbf{x}$ has weight $\omega - p$. In this case $\mathbf{P}(\mathbf{e}_1, \mathbf{x})$ forms a solution to the original syndrome decoding instance. If no solution is found, the algorithm is repeated for another random permutation.

Complexity. Let us briefly argue about the complexity of such a procedure. The probability of distributing the weight on $\mathbf{e}'$ as desired is

$$q := \frac{\binom{n-k-\ell}{\omega-p}\binom{k+\ell}{p}}{\binom{n}{\omega}}. \tag{4.7}$$

Hence, we expect that after q^{-1} random permutations one of them distributes the weight as desired. If now the cost to retrieve all weight-p solutions to the reduced instance for any of those permutations is T_{S}, the total complexity becomes

$$T = \tilde{\mathcal{O}}\left(q^{-1} \cdot T_{\mathrm{S}}\right).$$

In a nutshell different ISD algorithms differentiate in how they retrieve the solutions to the reduced instance. Usually they consider the reduced instance as a vectorial subset sum

instance, where the solution encodes a size-p subset of the columns of $\mathbf{H}_2$ that sums to $\mathbf{s}_2$. Then they make use of advanced algorithms for subset sum, such as the BCJ algorithm, to retrieve the solutions to that instance. It is not hard to see, that instead of working over $\mathbb{Z}_{2^n}$, the generalized BCJ algorithm outlined in Section 4.3 and, hence, also our improved trade-off from Section 4.4, work analogously over $\mathbb{F}_2^n$.

4.5.1 Improved ISD Trade-Offs

The May-Meurer-Thomae (MMT) ISD algorithm [MMT11] originally uses the BCJ construction in depth-2 to retrieve the solutions to the reduced instance. In the following we give an improved version of the MMT algorithm based on our new subset sum trade-off from Section 4.4. Our version improves the overall memory complexity and yields a better trade-off curve, i.e., we achieve runtime improvements for every fixed memory.

To make use of our generalized trade-off description (in depth 2) we need to define appropriate sets D_0, D_1 and D_2. Then to retrieve the running time we calculate the amount of existing representations and optimize the parameter ℓ_1. The pseudocode of our improved MMT algorithm is given in Algorithm 10.

Algorithm 10: New MMT Trade-Off

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, w \in \mathbb{N}$
Output: $\mathbf{e} \in \mathbb{F}_2^n, \mathbf{He} = \mathbf{s}$

Choose optimal ℓ, ℓ_1, p
let $r_1 = \log\binom{p}{p/2}$
$\pi_{\ell_1} : \mathbb{F}_2^{\ell} \to \mathbb{F}_2^{\ell_1}$, $\pi_{\ell_1}(x_1, \ldots, x_\ell) = \{x_1, \ldots, x_{\ell_1}\}$
repeat
  choose random permutation matrix $\mathbf{P}$
  $\bar{\mathbf{H}} = \begin{pmatrix} \mathbf{I}_{n-k-\ell} & \mathbf{H}_1 \\ 0 & \mathbf{H}_2 \end{pmatrix} = \mathbf{QHP}$ $\bar{\mathbf{s}} = (\mathbf{s}_1, \mathbf{s}_2) = \mathbf{Qs}$
  repeat $2^{\ell_1 - r_1}$ **times**
    Choose random $\mathbf{t} \in \mathbb{F}_2^{\ell_1}$
    Compute

$$L_1^{(1)} = \{\mathbf{z}_1 \mid \pi_{\ell_1}(\mathbf{H}_2\mathbf{z}_1) = \mathbf{t}, \mathbf{z}_1 \in D_1\} \quad \text{, via Schroeppel-Shamir}$$
$$L_2^{(1)} = \{\mathbf{z}_2 \mid \pi_{\ell_1}(\mathbf{H}_2\mathbf{z}_2 + \mathbf{s}_2) = \mathbf{t}, \mathbf{z}_2 \in D_1\} \quad \text{, via Schroeppel-Shamir}$$

    Compute $L = \{\mathbf{e}_2 \mid \mathbf{H}_2\mathbf{e}_2 = \mathbf{s}_2, \mathbf{e}_2 = \mathbf{z}_1 + \mathbf{z}_2\}$ from $L_1^{(1)}, L_2^{(2)}$
    for $\mathbf{e}_2 \in L$ **do**
      $\mathbf{e}_1 = \mathbf{H}_1\mathbf{e}_2 + \mathbf{s}_1$
      if $\mathrm{wt}(\mathbf{e}_1) \leq \omega - p$ **then**
        return $P(\mathbf{e}_1, \mathbf{e}_2)$

Complexity

We let D_2 be the set of vectors from $\mathbb{F}_2^{k+\ell}$ with weight p, as it defines our solution set. The MMT algorithm now chooses D_1 as vectors from $\mathbb{F}_2^{k+\ell}$ with weight $p/2$. Finally D_0 is

the set of vectors from $\mathbb{F}_2^{\frac{k+\ell}{2}}$ and weight $p/4$, i.e., a meet-in-the-middle split of D_1, hence $|D_0| = \sqrt{|D_1|}$.The size of D_1 is

$$|D_1| = \binom{k+\ell}{p/2},$$

while the amount of representations of one element from D_2 as sum of two elements from D_1 is

$$2^{r_1} = \binom{p}{p/2} \simeq 2^p.$$

Observe that the binomial coefficient counts the possibilities to distribute half of the ones of the target vector over the first addend, while the other half must then be covered by the second addend. Now, to find one representation of each solution to the reduced instance in the final list we need to ensure

$$\ell_1 \stackrel{!}{=} r_1.$$

Our trade-off from Section 4.4 now allows for $\ell_1 > r_1$ and compensates by repeating the procedure. Note that in depth-2 we have no further saturation constraints, nor can we make use of reconstructing different levels differently many times. The time complexity for finding all solutions to the vectorial subset sum problem then becomes

$$T_{\mathrm{S}} = 2^{\ell_1 - r_1} \cdot \max(\sqrt{|D_1|}, |D_1|/2^{\ell_1}, |D_1|^2/2^{\ell+\ell_1})$$

The memory complexity is equal to the level-0 and level-1 lists, since elements of the final list can be checked on the fly for being a solution. Moreover, by using the Schroeppel-Shamir algorithm for the construction of the level-1 lists we can reduce the memory required for storing the level-0 lists from $|D_0| = \sqrt{|D_1|}$ to $\sqrt{|D_0|} = |D_1|^{1/4}$ (see Section 4.2), which yields

$$M = \max(|D_1|^{1/4}, |D_1|/2^{\ell_1}).$$

4.5.2 Asymptotic Behavior of new Trade-off

For the asymptotic classification of our algorithmic improvement let us first consider the half distance setting, i.e., $\omega := H(1 - \frac{k}{n}) \cdot \frac{n}{2}$. Here our MMT variant improves the memory complexity by almost a square-root down to $2^{0.0135n}$ from $2^{0.0213n}$ of standard MMT, while maintaining the same time complexity of $T = 2^{0.05364n}$. The optimal parameters for our MMT variant in this case are

$$\ell = 0.0278n,\ \ell_1 = 0.0091n \text{ and } p = 0.0064n,$$

where the found worst case rate is $k = 0.45n$ as for standard MMT. We now further optimized the time complexity of our trade-off under a memory limitation of $M \leq 2^{\lambda n}$ for decreasing λ. Figure 4.4 shows the complete trade-off curves for both MMT variants – the original and our improved version. We observe that our trade-off outperforms the original trade-off for all memory parameters.

In the full distance setting we obtain a similar improvement. Here our improved MMT algorithm improves the memory complexity down to $2^{0.0375n}$ from previously $2^{0.053n}$, while achieving the same time complexity of $2^{0.112n}$. Again we obtain runtime improvements over standard MMT for any fixed memory.

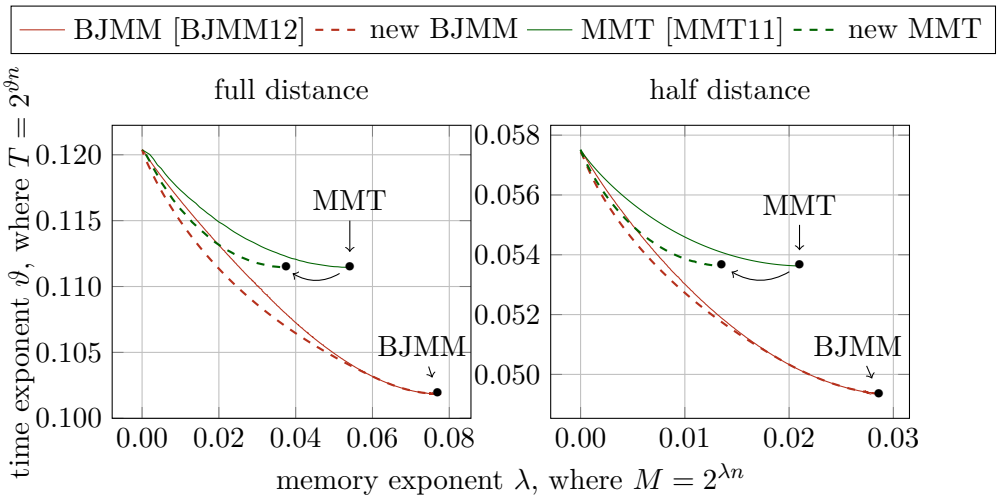


Figure 4.4: Comparison between the implicit (solid) and our new trade-off (dashed) for the MMT and BJMM algorithm. Complexity uses known worst case rates of the algorithm in the full distance (left) and half distance setting (right).

Even though, the MMT algorithm is not the asymptotically fastest ISD algorithm, so far none of its known asymptotic improvements [BJMM12, MO15, BM18] did transfer to the implementation level. This makes the MMT algorithm the preferred choice for record computations [ALL19] as well as security estimates [EMZ22].

BJMM algorithm. However, we also analyzed the algorithm by Becker-Joux-May-Meurer (BJMM) [BJMM12], which in contrast to the MMT algorithm uses slightly different sets D_i. That is, the vectors on each level have a slightly increased weight. Then in the $\mathbb{F}_2$-addition of those vectors some weight is assumed to cancel to still obtain a vector of weight p. The different possibilities, how the weight can cancel, increase the amount of representations and lead to an increased optimal tree-depth of three. This increased tree-depth allows us to make use of our subtree reconstruction technique yielding an improved trade-off curve also shown in Figure 4.4. We observe that the refined choice of the D_i gives the algorithm a possibility to balance the list sizes if memory is not limited. For that reason our strategy yields improvements only for limited memory in the case of the BJMM algorithm.

4.5.3 Practical Results and Security Estimates

We adapted the MMT / BJMM implementation from [EMZ22] to our new trade-off strategy. Interestingly, besides reducing the memory requirements we also obtain practical *running time* improvements, which stem from less, usually costly memory accesses.

We were able to solve several instances provided at `decodingchallenge.org` [ALL19], which were either unsolved or broken using more time and memory. Most notably, we obtained a new record computation in the quasi-cyclic setting, which follows the parameter selection of the BIKE and HQC schemes.

New record computation

Precisely, we solved the QC-3138 instance with code parameters $(n, k, \omega) = (3138, 1569, 56)$ with an estimated bit complexity of 66.7 (respectively 60.7 if counted in 64-bit register

operations) in only 2.23 CPU years. We estimated the expected time to solve this instance on our cluster, based on the processed permutations per second, to about 9.47 CPU years. The previous best implementation from [EMZ22] would need an expected amount of 30.31 CPU years, i.e., our implementation is about 3.2 times faster on this instance.

We also analysed the performance of our implementation on the next instance QC-3366 with parameters $(n, k, \omega) = (3366, 1683, 58)$, which has an estimated bit security of 68.7. We obtain an expected running time of 30.2 CPU years, which corresponds to an improvement by a factor of 5.7.

Furthermore we re-broke the previous QC-2918 record instance with parameters $(n, k, \omega) = (2918, 1459, 54)$ two times in just 224 CPU days, almost precisely hitting its expectation, which is about 6.9 times faster than the previous best implementation.

On McEliece like medium-sized instances we obtain a speedup by a factor of more than 2. We re-broke the current record instance McEliece-1284 using parameters $(n, k, \omega) = (1284, 1028, 24)$ in $X.X$ (expected 11.06) CPU years, where the initial record computation was performed in 22.04 (expected 26.28) CPU years, on similar hardware. Similarly, we estimated the next record instance to consume about 74.68 CPU years, improving from the previous estimate of 156.6 CPU years.

Security Estimates.

Next we investigate the impact of our improvement on the security of cryptographic sized instances. Therefore, we first adapted the estimation scripts from [EB22] to match our implementation, which allows us to precisely estimate the bit-complexity of given instances. Following previous works [EB22, BBC+19, EMZ22] we consider different memory access cost models. A memory access cost tries to model the practically faced memory access timings, by penalizing the algorithm for a high memory usage. Precisely, an algorithm with time complexity T and memory complexity M is assumed to have cost $T \cdot f(M)$, where f determines the penalty. We consider the established models of constant, logarithmic and cube-root access costs, which correspond to $f(M) = 1, f(M) = \log M$ and $f(M) = \sqrt[3]{M}$.

Now we extrapolate the time to solve an instance of proposed parameters from our obtained record computations. Therefore, we scale the time of our largest experiment in the respective setting by the difference in the bit-complexity of our experiment and the suggested parameters.

Methodology Example. Let us give a brief example of that methodology. Take the HQC category 1 parameter set $(n, k, \omega) = (35338, 17669, 132)$, in the constant memory access setting. This instance achieves a bit complexity of 144.7 according to our estimator, while our QC-3138 record has a bit complexity of 66.7 and took us about 2.24 CPU years to compute. We, therefore, extrapolate the time for breaking the HQC 128-bit parameters to $2.24 \cdot 2^{144.7-66.7} \approx 2^{79.16}$ CPU years.

Since the commonly addressed security categories 1, 3 and 5 relate their security to the security of AES-128, -192 and -256, we also estimated the time complexity of breaking AES on our cluster. Therefore, we benchmarked the number of AES encryptions our cluster is able to perform per second from which we obtained the expected time to break AES with respective key size.

BIKE / HQC. Table 4.2 states the security margin (in bits) the corresponding parameter set has over breaking AES. Precisely the table states $\log \frac{T_{\text{Scheme}}}{T_{\text{AES}}}$. Here, T_{Scheme} is the estimated time to break the schemes parameters and T_{AES} the estimated time to break the corresponding

Quasi-Cyclic		Category 1	Category 3	Category 5
constant:				
BIKE	message	−0.65 (3.09)	−0.59 (3.09)	0.26 (3.23)
	key	0.73 (3.15)	−1.07 (3.20)	2.13 (3.74)
HQC		−1.84 (3.08)	1.19 (3.09)	−0.86 (3.09)
logarithmic:				
BIKE	message	−0.36 (3.22)	−0.21 (3.25)	0.84 (3.26)
	key	1.24 (3.18)	−0.10 (3.21)	3.50 (3.24)
HQC		−1.51 (3.23)	1.61 (3.26)	−0.38 (3.28)
cube-root:				
BIKE	message	0.37 (4.10)	0.77 (4.43)	1.99 (4.69)
	key	1.89 (3.88)	0.79 (4.21)	4.60 (4.43)
HQC		−0.67 (4.29)	2.71 (4.63)	0.90 (4.85)

Table 4.2: Bit-difference in security of BIKE/HQC and AES with respective key-length considering different memory access cost.

AES instantiation on our cluster. The number in parentheses states the improvement over a similar analysis performed in [EMZ22], i.e., one obtains their result as a sum of both numbers.

We observe that our time-memory trade-off, despite saving on memory, actually lowers the bit-security estimates of BIKE and HQC by about 3 to 5 bits. Note that, this is more than the improvement of representation-based ISD algorithms like MMT over early algorithms like Stern and Dumer on these instances [EB22].

Further, most instances provided quite exactly this amount as a security margin, i.e., any further improvement most likely leads to the necessity of a parameter adaption. Since our algorithm has a reduced memory complexity we obtain larger gains when considering higher memory access cost models.

In the case of BIKE we distinguish message and key security as both settings allow for slightly different speedups [ABB$^+$17].

McEliece. In Table 4.3 we state the obtained security margins for the round 3 McEliece parameter sets, using the same extrapolation methodology. Since in the McEliece setting ISD algorithms tend to use very high amounts of memory we also consider memory-limited settings. In those we restrict the memory consumption of the algorithm to not exceed 2^{80} or 2^{60} bits respectively. We reduce the security estimates for McEliece by about 1 to 6 bits and obtain the best results in memory-limited settings, where our new time-memory trade-offs can play its strength. Again the number in brackets indicates by how much we reduced the previous estimate from [EMZ22].

Note that under cube-root memory access cost none of the optimal algorithmic configurations exceeds 2^{60} bits of memory.

McEliece	Category 1 $n = 3488$	Category 3 $n = 4608$	Category 5a $n = 6688$	Category 5b $n = 6960$	Category 5c $n = 8192$
constant:					
unlimited	−0.82 (0.91)	−26.10 (1.24)	−24.04 (0.86)	−24.73 (0.93)	5.14 (0.96)
$M \leq 2^{80}$	0.29 (1.25)	−23.95 (2.43)	−13.65 (1.98)	−13.36 (2.49)	21.18 (2.19)
$M \leq 2^{60}$	2.63 (2.17)	−19.85 (0.73)	− 9.07 (5.21)	− 8.58 (4.78)	26.43 (6.27)
logarithmic:					
unlimited	0.84 (0.93)	−24.15 (1.04)	−21.60 (0.90)	−22.26 (0.97)	7.85 (0.99)
$M \leq 2^{80}$	1.54 (1.32)	−22.65 (2.24)	−12.40 (1.94)	−12.11 (2.48)	22.47 (2.17)
$M \leq 2^{60}$	3.46 (2.09)	−19.18 (0.85)	− 8.32 (4.86)	− 7.81 (4.41)	27.21 (5.95)
cube-root:					
	8.94 (1.43)	−13.79 (1.52)	− 1.57 (2.39)	− 0.72 (2.10)	35.43 (2.79)

Table 4.3: Bit-difference in security of Classic McEliece and AES with respective key-length considering different memory access cost.

4.6 Generalization to arbitrary depth d

Note that in general we have

$$\mathcal{L}_{i+1} = \frac{(q_i \cdot \mathcal{L}_i)^2}{2^{\ell_i}},$$

where ℓ_i is the additional bitwise constraint introduced on level i. The time and memory complexity are then given as before. The saturation constraints extend to

$$q_i \cdot \mathcal{L}_i \leq \frac{|D_i|}{2^{\ell_1+\ldots+\ell_i}} \text{ for } i = 2, \ldots, d-1,$$

where d is the depth of the tree. Together with the definition of the filtering probability given in Equation (4.4), we can rewrite the saturation constraints for each level i as

$$\sum_{j=1}^{i}(2^{i-j}-1)\ell_j \geq \sum_{j=1}^{i} 2^{i-j} \cdot r_j \text{ for } i = 1, \ldots d-2,$$

where there exist 2^{r_j} different representations of any element from D_{j+1} as a sum of two elements from D_j. Finally, the requirement of finding one representation of the solution in the final list is expressed via the condition

$$q_d \cdot \mathcal{L}_d = 1,$$

which similar to the saturation constraints rewrites to

$$\sum_{j=1}^{d-1}(2^{d-j}-1)\ell_j = \sum_{j=1}^{d-1} 2^{d-j-1} \cdot r_j. \tag{4.8}$$

5 A Faster Algorithm for Finding Closest Pairs in Hamming Metric

We study the Closest Pair Problem in Hamming metric, which asks to find the pair with the smallest Hamming distance in a collection of binary vectors. We give a new randomized algorithm for the problem on uniformly random input outperforming previous approaches whenever the dimension of input points is small compared to the dataset size. For moderate to large dimensions, our algorithm matches the time complexity of the previously best-known locality sensitive hashing based algorithms. Technically our algorithm follows similar design principles as Dubiner [Dub10] and May-Ozerov [MO15]. Besides improving the time complexity in the aforementioned areas, we significantly simplify the analysis of these previous works. We give a modular analysis, which allows us to investigate the performance of the algorithm also on non-uniform input distributions. Furthermore, we give a proof of concept implementation of our algorithm which performs well in comparison to a quadratic search baseline. This is the first step towards answering an open question raised by May and Ozerov regarding the practicability of algorithms following these design principles.

The content of this chapter is the result of a collaboration with Andre Esser and Robert Kübler. It previously appeared as A Faster Algorithm for Finding Closest Pairs in Hamming Metric in IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021) and is reproduced here with permission.

5.1 Introduction

Finding closest pairs in a given dataset of binary vectors is a fundamental problem in theoretical computer sciences with numerous applications in data science, machine learning, computer vision, cryptography, and many others.

Image data for example is often represented via compact binary codes to allow for efficient closest pair search in applications like similarity search in images or facial recognition systems [CLSF10, LLZZ15, SBBF11]. The usage of binary codes also allows decoding the represented data to common codewords. Here, the most efficient algorithms known for decoding such random binary linear codes also heavily benefit from improved algorithms for the Closest Pair Problem [MO15, BM18]. Another common application lies in the field of bioinformatics, where the analysis of genomes involves closest pair search on large datasets to identify most correlated genetic markers [MDC05, MSL+07].

To be more precise, the Closest Pair Problem asks to find the pair of vectors with the minimal Hamming distance among n given binary vectors. While the general version of this problem does not make any restrictions on the distribution of input points, several settings imply a uniform distribution of dataset elements [BM18, MDC05, MSL+07, MO15]. Usually, in such settings, there is a planted pair, which attains relative distance $\omega \in [0, \frac{1}{2}]$, which has to be found. This uniform version is also known as the *light bulb problem* [Val88]. The

problem can be solved in time linearly in the dataset size[1] as long as the dimension of vectors is constant [Ben80, KM95]. As soon as the dimension is non-constant an effect occurs known as *curse of dimensionality*, which lets the problem become much harder.

The most common framework to assess the problem is based on *locality-sensitive hashing* (LSH), whose research was initiated in the pioneering work of Indyk and Motwani [IM98]. Roughly speaking, a locality-sensitive hash function is more likely to hash points that are close to each other to the same value, rather than points that are far apart. To solve the Closest Pair Problem leveraging an LSH family one chooses a random hash function of the family and computes the hash value of all points in the dataset. In a next step, one computes the pairwise distance only for those pairs hashing to the same value. This process is then repeated for different hash functions until the closest pair is found. The initial algorithm by Indyk-Motwani achieves a time complexity of $n^{\log_2(\frac{2}{1-\omega})}$. In general a time lower bound of $n^{\frac{1}{1-w}}$ is known for LSH based algorithms [Dub10, MNP06]. In [Dub10] Dubiner also gives an abstract idea of an algorithm achieving this lower bound. Later May and Ozerov [MO15] gave the first concrete algorithmic description following similar design principles, also achieving the mentioned lower bound. Additionally, current data-dependent hashing schemes [AR15], where the hash function depends also on the actual points in the dataset, improve on the initial idea by Indyk-Motwani and also match the time lower bound of [Dub10, MNP06].

In the uniform setting Valiant [Val12] was able to circumvent the lower bound by leveraging fast matrix multiplication and hence breaking out of the LSH framework to give an algorithm that runs in time $n^{1.63}\text{poly}(d)$. Remarkably, the complexity exponent of Valiant's algorithm does not depend on the relative distance ω at all. Later this bound was improved to $n^{1.58}\text{poly}(d)$ by Karpa et al. [KKK16] and simplified in an elegant algorithm by Alman [Alm19] achieving the same complexity.

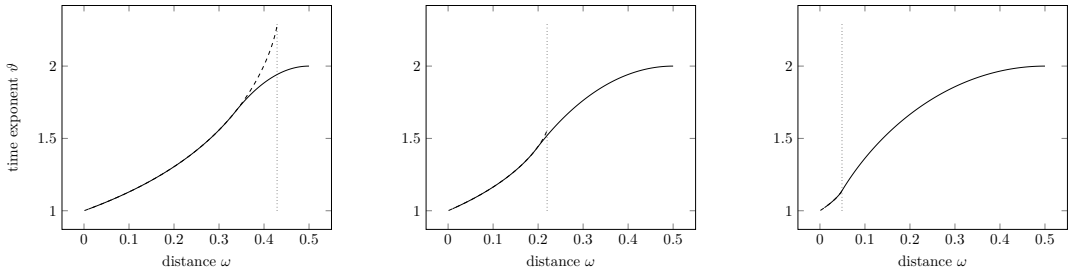
All mentioned algorithms have in common, that they assume a dimension of $d = c(n)\log(n)$, where $c(n)$ is at least a big constant, the results by [AR15, Dub10, Val12] for example take $\frac{1}{c(n)} = o(1)$. Here, the algorithm by May-Ozerov forms an exception by being applicable for any $c(n) \geq \frac{1}{1-H(\frac{\omega}{2})}$, where $H(\cdot)$ denotes the binary entropy function. Nevertheless, the mentioned lower bound is only achieved for $c(n)$ approaching infinity. Recently, Xie, Xu and Xu [XXX19] proposed a new algorithm based on decoding the points of the data set according to some random code, exploiting that close vectors are more likely to be decoded to the same word. Their algorithm is also applicable for any $c(n)$ that allows to bound the number of pairs attaining relative distance ω to a constant number with high probability. The authors are able to derandomize their approach and, thus, obtain the fastest known deterministic algorithm for small constants $c(n)$. However, if one also considers probabilistic procedures, their method is inferior to the one by May-Ozerov.

5.1.1 Our Contribution

We design a randomized algorithm, which achieves the best-known running time for solving the Closest Pair Problem on uniformly random input, when the dimension d is small, which means for small constants $c(n)$. Additionally, our algorithm matches the running time of the best known LSH algorithms for larger values of $c(n)$ and still matches the time lower bound for LSH based schemes if $\frac{1}{c(n)} = o(1)$. To quantify we give in Figure 5.1 the achieved runtime exponent for $c(n) \in \{1.2, 2, 4\}$ of our algorithm in comparison to May-Ozerov. As indicated by the figure, our approach performs also exceptionally well for large closest pair distances,

[1] here we ignore polylogarithmic factors in the dataset size

where common LSH based techniques usually fail [EWF12]. Moreover, we show that for large distances our algorithm is indeed optimal.



(a) Dimension $d = 4\log_2(n)$ (b) Dimension $d = 2\log_2(n)$ (c) Dimension $d = 1.2\log_2(n)$

Figure 5.1: Time complexity exponent ϑ as a function of the relative distance ω of the closest pair for different dimensions. The running time is of the form $n^{\vartheta} \cdot \text{poly}(d)$, where the dashed line represents May-Ozerov's algorithm and the solid line depicts the exponent of our new algorithm. The dotted line gives the maximal ω for which the algorithm by May-Ozerov is still applicable.

Technically our algorithm follows similar design principles as [Dub10, MO15]. At its core, these algorithms group the elements of the given datasets recursively into buckets according to some criterion, which fulfills properties that are similar to those of locality-sensitive hash functions. As the buckets in the recursion are decreasing in size, at the end of the recursion they become small enough to compute the pairwise distance of all contained elements naively.

In contrast to previous works, we exchange the used bucket criteria, which allows us to significantly simplify the algorithms' analysis as well as improve for the mentioned parameter regimes. Also our approach is applicable for any $c(n)$, thus we are able to remove the restriction $c(n) \geq \frac{1}{1-H(\frac{\omega}{2})}$.

Following May-Ozerov and Dubiner, we study the bichromatic version of the Closest Pair Problem, which takes as input two datasets rather than one and the goal is to find the closest pair between those given datasets. Obviously, there exists a randomized reduction between the Closest Pair Problem and its bichromatic version, but our algorithm can also be easily adapted to the single dataset case. However, May and Ozerov require the elements within each dataset to be pairwise independent of each other, as a minor contribution we get rid of this restriction, too.

Also, we investigate the algorithms' performance on different input distributions. Therefore we give a modular analysis, which allows for an easy exchange of dataset distribution as well as the choice of bucketing criterion. We also give numerical upper bounds for the algorithm's complexity exponent on some exemplary input distributions. These examples suggest that the chosen criterion is well suited as long as the distance between input elements concentrates around $\frac{d}{2}$ (as in the case of random input lists), while being non-optimal as soon as the expected distance decreases.

We also address an open research question regarding the practical applicability of algorithms following the design of [Dub10, MO15] raised by May and Ozerov. As their algorithm inherits a huge polynomial overhead in time and space, they left it as an open problem to give a more practical algorithm following a similar design. While our analysis first suggests an equally high overhead, we are able to give an efficient implementation of our algorithm, which requires in addition to the input dataset only constant space. Also, our practical experiments show that most of the overhead of our algorithm is an artifact of the analysis and can be circumvented in practice so that our algorithm performs well compared to a quadratic search baseline.

The rest of the paper is organized as follows: In the subsequent section, we introduce the necessary notation and state the exact definition of the Closest Pair Problem under consideration. In section 3, we then give a detailed description of our new algorithm and establish a proof of its running time as well as its correctness. In the following section 4, we investigate the performance of our algorithm on different input distributions. Finally, in section 5, we give practical improvements of the algorithm and runtime results of our implementation compared to a quadratic search baseline.

5.2 Preliminaries

5.2.1 Notation

For $a, b \in \mathbb{N}$, $a \leq b$ we denote $[a, b] := \{a, a+1, \ldots, b-1, b\}$. In particular, let $[b] := [1, b]$. For a vector $\mathbf{v} \in \mathbb{F}_2^d$ and $I \in [d]$ let $\mathbf{v}_I$ be the projection of $\mathbf{v}$ onto the coordinates indexed by I, i.e. for $\mathbf{v} = (v_1, v_2, \ldots, v_d)$ and $I = \{i_1, i_2, \ldots, i_k\}$ we have $\mathbf{v}_I = (v_{i_1}, \ldots, v_{i_k}) \in \mathbb{F}_2^k$. We denote the uniform distribution on $\mathbb{F}_2^d$ as $\mathcal{U}\left(\mathbb{F}_2^d\right)$. We define $f(n) = \tilde{\mathcal{O}}\left(g(n)\right) :\Leftrightarrow \exists i \in \mathbb{N}\colon f(n) = \mathcal{O}\left(g(n) \cdot \log^i(g(n))\right)$, i.e. the tilde additionally suppresses polylogarithmic factors in comparison to the standard Landau notation $\mathcal{O}$.

Furthermore, we consider all logarithms having base 2. Define the binary entropy function as $\mathrm{H}(x) = -x\log(x) - (1-x)\log(1-x)$ for $x \in (0,1)$, and additionally $\mathrm{H}(0) = \mathrm{H}(1) := 0$. Using this together with Stirling's formula $n! = \Theta\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)$ we obtain $\binom{n}{\omega n} = \tilde{\Theta}\left(2^{\mathrm{H}(\omega)n}\right)$. We additionally define $H^{-1}\colon [0,1] \to [0, \frac{1}{2}]$ to be the inverse of the left branch of H.

Lemma 3. *Let $\mathbf{v}_1, \ldots, \mathbf{v}_n \sim \mathcal{U}\left(\mathbb{F}_2^d\right)$ independent and $M \in \mathbb{F}_2^{n \times n}$ an invertible matrix. Then for*

$$\begin{pmatrix} \mathbf{v}'_1 \\ \vdots \\ \mathbf{v}'_n \end{pmatrix} := M \cdot \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_n \end{pmatrix}$$

it also holds that $\mathbf{v}'_1, \ldots, \mathbf{v}'_n \sim \mathcal{U}\left(\mathbb{F}_2^d\right)$ are independently and uniformly distributed.

Corollary 1. *For $\mathbf{v}, \mathbf{w}, \mathbf{z} \sim \mathcal{U}\left(\mathbb{F}_2^d\right)$ independent, $\mathbf{v}+\mathbf{z}, \mathbf{w}+\mathbf{z} \sim \mathcal{U}\left(\mathbb{F}_2^d\right)$ are also uniform and independent.*

Proof. We have

$$\begin{pmatrix} \mathbf{v}+\mathbf{z} \\ \mathbf{w}+\mathbf{z} \\ \mathbf{z} \end{pmatrix} := \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v} \\ \mathbf{w} \\ \mathbf{z} \end{pmatrix}$$

Since the matrix is invertible we can apply Lemma 3. □

5.2.2 Closest Pair Definition

In this work, we consider the Bichromatic Closest Pair Problem in Hamming metric. Here, the inputs are two lists of equal size containing elements drawn uniformly at random from $\mathbb{F}_2^d$ plus a planted pair, whose Hamming distance is ωd for some known ω. More formally, we state the problem in the following definition. To allow for easy comparison to the result of May-Ozerov, we follow their notation using the dimension as the primary difficulty parameter. Thus we let the list sizes be $n := 2^{\lambda d}$, which means $\lambda = \frac{1}{c(n)}$, where $d = c(n)\log n$.

Definition 11 (Bichromatic Closest Pair Problem)**.** Let $d \in \mathbb{N}$, $\omega \in \left[0, \frac{1}{2}\right]$ and $\lambda \in (0, 1]$. Let $L_1 = (\mathbf{v}_i)_{i \in [2^{\lambda d}]}, L_2 = (\mathbf{w}_i)_{i \in [2^{\lambda d}]} \in \left(\mathbb{F}_2^d\right)^{2^{\lambda d}}$ be two lists containing elements uniformly drawn at random, together with a distinguished pair $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ with $\mathrm{wt}(\mathbf{x} + \mathbf{y}) = \omega d$. We further assume that for each i, j the vectors $\mathbf{v}_i$ and $\mathbf{w}_j$ are pairwise stochastically independent. The *Closest Pair Problem* $\mathcal{CP}_{d,\lambda,\omega}$ asks to find this *closest pair* $(\mathbf{x}, \mathbf{y})$ given L_1, L_2 and the weight parameter ω. We call $(\mathbf{x}, \mathbf{y})$ the *solution* of the $\mathcal{CP}_{d,\lambda,\omega}$ problem.

First, note that $\lambda \leq 1$ is not a real restriction since for $\lambda > 1$ the lists must contain duplicates, which can be safely removed, giving us a problem instance with $\lambda \leq 1$. We also consider the Closest Pair Problem on input lists whose elements are distributed according to some distribution $\mathcal{D}$ different from the uniform one used in Definition 11. To indicate this, we refer to the $\mathcal{CP}_{d,\lambda,\omega}$ *over distribution* $\mathcal{D}$. Note that in this case, the meaningful upper bound for λ is the entropy of $\mathcal{D}$.

Technically speaking, it is also not necessary to know the value of ω, as the time complexity of appropriate algorithms to solve the $\mathcal{CP}_{d,\lambda,\omega}$ problem is solely increasing in ω. Thus if ω is unknown, one would apply the algorithm for each $\omega d = 0, 1, 2, \ldots$ until the solution is found, which results at most in polynomial overhead.

It is well known, that any LSH based algorithm solving the problem of Definition 11 with non-negligible probability needs at least time complexity $|L_1|^{\frac{1}{1-\omega}} = 2^{\frac{\lambda d}{1-\omega}}$ [Dub10, MNP06]. However, this lower bound assumes the promised pair to be uniquely distinguishable from all other pairs in $L_1 \times L_2$. Obviously, if the relation of ω and λ lets us expect more than the promised pair of distance ωd in the input lists, an algorithm solving the Closest Pair Problem needs to find all (or at least a non-negligible fraction) of these closest pairs.[2] Such scenarios for example frequently occur when the solution to the $\mathcal{CP}_{d,\lambda,\omega}$ problem actually is a solution to some different problem [MO15, Hir16, BDGL16, GKH17], which enables a distinction from other closest pairs. Hence, if the input lists contain E closest pairs the optimal time complexity becomes

$$\tilde{\Omega}\left(\max(2^{\frac{\lambda d}{1-\omega}},\ E)\right)$$

Let $(\mathbf{v}, \mathbf{w}) \in L_1 \times L_2 \setminus \{(\mathbf{x}, \mathbf{y})\}$ be arbitrary list elements. If the elements are chosen independently and uniformly at random, as stated in Definition 11 we expect E to be of size

$$\begin{aligned}
\mathbb{E}[|E|] &= (|L_1 \times L_2| - 1) \cdot \Pr\left[\mathrm{wt}(\mathbf{v} + \mathbf{w}) = \omega d\right] + \underbrace{1}_{\text{from } (\mathbf{x},\mathbf{y})} \\
&= \left(2^{2\lambda d} - 1\right) \cdot \frac{\binom{d}{\omega d}}{2^d} + 1 \\
&= \tilde{\Theta}\left(2^{(2\lambda + \mathrm{H}(\omega) - 1)d}\right) \ ,
\end{aligned}$$

and, thus, the optimal time complexity to solve the $\mathcal{CP}_{d,\lambda,\omega}$ problem becomes

$$T_{\mathrm{opt}} = \tilde{\Omega}\left(\max\left(2^{\frac{\lambda d}{1-w}},\ 2^{(2\lambda + \mathrm{H}(\omega) - 1)d}\right)\right) \ . \tag{5.1}$$

5.3 Our new Algorithm

Our algorithm groups the input elements according to some criterion into several buckets, each one representing a new closest pair instance with smaller list size. We then apply this

[2] Note that in such a scenario the searched $(\mathbf{x}, \mathbf{y})$ is probably not the pair with the smallest Hamming distance, however, we still refer to elements attaining Hamming distance ωd as *closest pairs*.

bucketing procedure recursively until the buckets contain few enough elements to eventually solve the Closest Pair Problem represented by them via a naive quadratic search algorithm, the exhaustive search.

As a bucketing criterion we choose the weight of the vectors after adding a randomly drawn vector $\mathbf{z}$ from $\mathbb{F}_2^d$. Thus, each bucket is represented by a vector $\mathbf{z}$ and only those elements $\mathbf{v}$ are added to the bucket, which satisfy $\mathrm{wt}(\mathbf{v}+\mathbf{z}) = \delta d$, where δ is determined later.

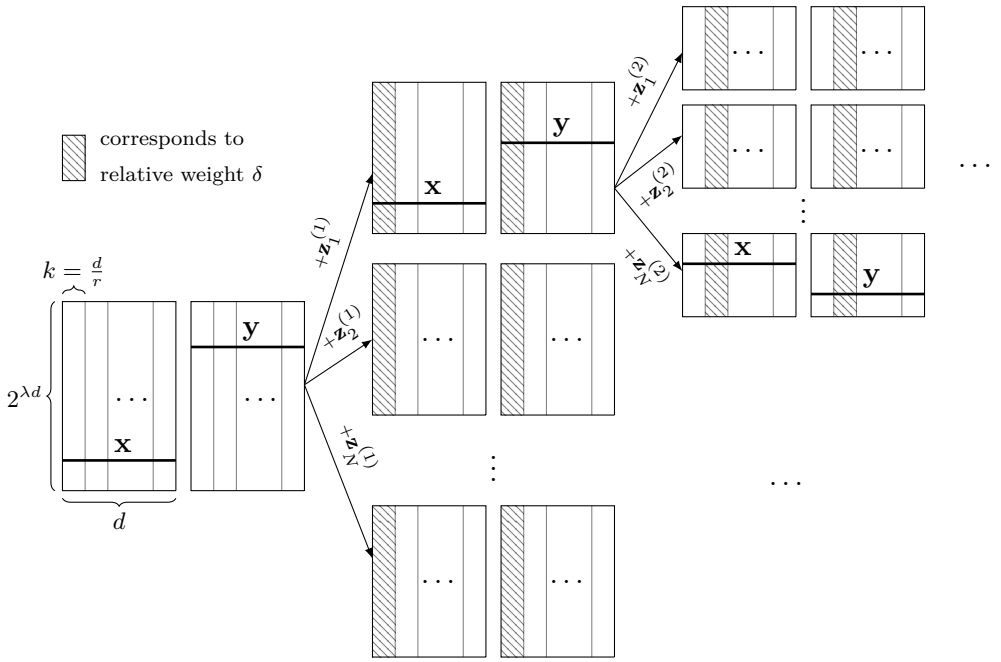


Figure 5.2: We start off on the left side of the illustration with the two input lists L_1, L_2 containing the closest pair $(\mathbf{x}, \mathbf{y})$. Going right, in each iteration of the algorithm, N different $\mathbf{z}_i^{(j)}$ are randomly chosen and all of the list elements are tested if they fulfill the bucketing criterion. The crosshatched pattern indicates the parts where the bucket criterion is fulfilled, i.e. the list vectors differ from $\mathbf{z}_i^{(j)}$ in δk positions.

More precisely in each recursive iteration, our algorithm works only on equally large blocks of the input vectors and not on the full d coordinates, i.e. the weight condition is only checked on the current block. This is a technical necessity to obtain independence of vectors in the same bucket on fresh blocks. Let us formally define the notion of blocks.

Definition 12 (Block)**.** Let $d, r \in \mathbb{N}$ with $r \mid d$ and $i \in [r]$. Then we denote the i-th *block* of $[d]$ as

$$B_{i,r}^d := \left[(i-1)\frac{d}{r} + 1,\ i\frac{d}{r}\right] \ .$$

Note that $[d] = \biguplus_{i\in[r]} B_{i,r}^d$ and $\left|B_{i,r}^d\right| = \frac{d}{r}$ for each $i \in [r]$. For a leaner notation and since the role of d does not change in the course of this paper, we omit the index d in the following, thus we write $B_{i,r} := B_{i,r}^d$.

In each iteration, we choose the number N of buckets in such a way that with overwhelming probability the closest pair lands in at least one of the buckets. Hence, our algorithm creates a tree with branching factor N with the distinguished pair being contained in one of the leaves. The deeper we get into the tree, the smaller and, hence, the easier the closest pair

instances get. An algorithmic description of the whole procedure is given in pseudocode in Algorithm 11.

Algorithm 11: CLOSEST-PAIR(L_1, L_2, ω)

Input : lists $L_1, L_2 \in \left(\mathbb{F}_2^d\right)^{2^{\lambda d}}$, weight parameter $\omega \in \left[0, \frac{1}{2}\right]$
Output: list L containing the solution $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ to the $\mathcal{CP}_{d,\lambda,\omega}$

begin
  Set $r, P, N \in \mathbb{N}$, $\delta \in \left[0, \frac{1}{2}\right]$ properly and define $k := \frac{d}{r}$
  for P *permutations* π **do**
    ▷ permutation on the bit positions Stack $S := [(\pi(L_1), \pi(L_2), 0)]$
    $L \leftarrow \emptyset$
    while S *is not empty* **do**
      $(A, B, i) \leftarrow S.\text{pop}()$
      if $i < r$ **then**
        for N *randomly chosen* $\mathbf{z} \in \mathbb{F}_2^k$ **do**
          $A' \leftarrow (\mathbf{v} \in A \mid \text{wt}((\mathbf{v} + \mathbf{z})_{B_{i+1,r}}) = \delta k)$
          $B' \leftarrow (\mathbf{w} \in B \mid \text{wt}((\mathbf{w} + \mathbf{z})_{B_{i+1,r}}) = \delta k)$
          $S.\text{push}((A', B', i+1))$
      else
        for $\mathbf{v} \in A, \mathbf{w} \in B$ **do**
          ▷ Naive search **if** $\text{wt}(\mathbf{v} + \mathbf{w}) = \omega d$ **then**
            $L \leftarrow L \cup \{(\mathbf{v}, \mathbf{w})\}$

The following theorem gives the time complexity of our algorithm to solve the $\mathcal{CP}_{d,\lambda,\omega}$.

Theorem 2. *theoremmainthm Let $\omega \in \left[0, \frac{1}{2}\right]$ and $\lambda \in [0, 1]$. Then Algorithm 11 solves the $\mathcal{CP}_{d,\lambda,\omega}$ problem with overwhelming success probability in expected time $2^{\vartheta d(1+o(1))}$, where*

$$\vartheta = \begin{cases} (1-\omega)\left(1 - H\left(\frac{\delta^\star - \frac{\omega}{2}}{1-\omega}\right)\right) & \text{for } \omega \leq \omega^\star \\ 2\lambda + H(\omega) - 1 & \text{for } \omega > \omega^\star \end{cases},$$

with $\delta^\star := H^{-1}(1-\lambda)$ and $\omega^\star := 2\delta^\star(1-\delta^\star)$.

The case distinction can intuitively be explained as follows: As long as the number of pairs with distance ωd in the input lists is small enough the algorithm is optimal for a choice of δ such that the lists at the leaves of the tree become polynomial in size. However, if too many closest pairs exist in the input lists, enforcing polynomial size of the leaf nodes lets the probability of the solution being contained in one of them drop immensely. Thus to still ensure the algorithm having success in finding the solution an enormous branching factor would be required. Hence, instead the choice of δ is adapted, which leads to larger leaf nodes and in total to a time complexity that is linear in the number of closest pairs, which matches the lower bound from Equation (5.1).

We establish the proof of Theorem 2 in a series of lemmata and theorems. Note that any bucketing algorithm heavily depends on two probabilities specific to the chosen bucketing

criterion. First, the probability that any element falls into a bucket, which we call p in the remainder of this work. This probability is mainly responsible for the lists' sizes throughout the algorithm. The second relevant probability, which we call q describes the event of both, $\mathbf{x}$ and $\mathbf{y}$, falling into the same bucket, where $(\mathbf{x}, \mathbf{y})$ is the solution to the $\mathcal{CP}_{d,\lambda,\omega}$ problem. This is the probability of $(\mathbf{x}, \mathbf{y})$ *surviving* one iteration meaning that q determines the success probability of the algorithm. In summary, for our choice of bucketing criteria, we get

$$\begin{aligned} p &:= \Pr_{\mathbf{z}}\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k\right] \text{ for any } \mathbf{v} \in \mathbb{F}_2^k \text{ and} \\ q &:= \Pr_{\mathbf{z}}\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}}) = \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}}) = \delta k\right] \ , \end{aligned} \tag{5.2}$$

where $k = \frac{d}{r}$ is the block width. If we assume that the ωd differing coordinates of $\mathbf{x}$ and $\mathbf{y}$ distribute evenly into the r blocks, i.e. $\mathrm{wt}((\mathbf{x}+\mathbf{y})_{B_{i,r}}) = \omega k$ for each i, these probabilities are independent of i for δk fixed. This property is ensured for at least one of the P permutations in Algorithm 11 with overwhelming probability, as we will see in the proof of Theorem 2.

We determine the exact form of q and p later. First, we are going to prove the following statement about the expected running time of Algorithm 11 in dependence on both probabilities.

Theorem 3. *Let q and p be as defined in Equation (5.2), $\omega \in \left[0, \frac{1}{2}\right]$, $\lambda \in [0,1]$ and $r = \frac{\lambda d}{\log^2 d}$. Then Algorithm 11 solves the $\mathcal{CP}_{d,\lambda,\omega}$ problem in expected time*

$$\max\left(q^{-r}, \frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{\left(2^{\lambda d} \cdot p^r\right)^2}{q^r}\right)^{1+o(1)}$$

with a success probability overwhelming in d.

Proof. First, we are going to prove the statement about the time complexity.

The algorithm maintains a stack, containing list pairs together with an associated counter. In every iteration of the loop in line 6, one element is removed from the stack and if the counter i associated with this element is smaller than r, N additional elements $(A', B', i+1)$ are pushed to the stack in line 12. Let us consider the elements on the stack as nodes in a tree of depth r, where all elements with associated counter i are siblings on level i of the tree. Also, depict the elements pushed to the stack in line 12 as child nodes of the currently processed node (A, B, i). Then the total number of elements with associated counter i pushed to the stack is bounded by the number of nodes on level i in a tree with branching factor N, which is N^i.

Next, let us determine the lists' sizes on level i of that tree. Therefore, let the expected size of lists on level i be $\mathcal{L}_i$. As these lists are constructed from the lists of the previous level by testing the weight condition in line 10 and 11, it holds that

$$\mathcal{L}_i = \mathcal{L}_{i-1} \cdot \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})) = \delta k\right] := \mathcal{L}_{i-1} \cdot p \ ,$$

where $i > 0$ and by construction $\mathcal{L}_0 = |L_1|$. By substitution we get

$$\mathcal{L}_i = |L_1| \cdot p^i \ , \text{ for } i = 0, \ldots, r.$$

Now, we are able to compute the time needed to create the nodes on level i of the tree. Observe that for the creation of a level-i node we need to linearly scan through the larger

lists of a node on level $i-1$ to check the weight conditions. Thus, to construct all N^i nodes of level i we need a total time of

$$T_i = \tilde{\mathcal{O}}\left(\mathcal{L}_{i-1} \cdot N^i\right) = \tilde{\mathcal{O}}\left(|L_1| \cdot p^{i-1} \cdot N^i\right) \ ,$$

for each $0 < i \leq r$. Eventually, the list pairs on level r are matched by a naive search with quadratic runtime resulting in

$$T_{r+1} = \tilde{\mathcal{O}}\left(N^r \cdot \mathbb{E}[|A_r| \cdot |B_r|]\right) \ ,$$

where A_r, B_r describe the lists of a level-r node.

The expected value of the product, now, depends on the chosen input distribution. We next argue that for the given input distribution we have

$$\mathbb{E}[|A_r| \cdot |B_r|] = \mathcal{O}(\mathbb{E}[|A_r|] \cdot \mathbb{E}[|B_r|]) = \mathcal{O}(\mathcal{L}_r^2) \ .$$

To see this, first note that for $\mathbf{v}, \mathbf{w}, \mathbf{z}$ independent and uniform, $\mathbf{v}+\mathbf{z}$ and $\mathbf{w}+\mathbf{z}$ are also independent and uniform according to Corollary 1. This in turn implies

$$\begin{aligned}
&\Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \\
=&\Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \cdot \Pr\left[\mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \\
=&p^2
\end{aligned}$$

since deterministic functions of independent random variables are still independent. This also works for either $\mathbf{v} = \mathbf{x}$ or $\mathbf{w} = \mathbf{y}$, but not for $(\mathbf{v}, \mathbf{w}) = (\mathbf{x}, \mathbf{y})$. In this case, however, we have $\Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}})) = \delta k, \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}})) = \delta k\right] = q$ by definition. With this insight, we can express $\mathbb{E}[|A_i| \cdot |B_i|]$ in terms of $\mathbb{E}[|A_{i-1}| \cdot |B_{i-1}|]$ for each i via

$$\begin{aligned}
\mathbb{E}[|A_i| \cdot |B_i| \mid A_{i-1}, B_{i-1}] = &\sum_{\substack{\mathbf{v} \in A_{i-1},\ \mathbf{w} \in B_{i-1} \\ (\mathbf{v},\mathbf{w}) \neq (\mathbf{x},\mathbf{y})}} \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}}) = \delta k\right] \\
&+ \Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}})) = \delta k, \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \\
&= (|A_{i-1}| \cdot |B_{i-1}| - 1)p^2 + q \\
&\leq |A_{i-1}| \cdot |B_{i-1}| \cdot p^2 + 1 \ ,
\end{aligned}$$

Applying the Law of total Expectation we obtain

$$\mathbb{E}[|A_i| \cdot |B_i|] = \mathbb{E}[\mathbb{E}[|A_i| \cdot |B_i| \mid A_{i-1}, B_{i-1}]] \leq \mathbb{E}[|A_{i-1}| \cdot |B_{i-1}|] \cdot p^2 + 1 \tag{5.3}$$

Successive application of Equation (5.3) yields

$$\mathbb{E}[|A_r| \cdot |B_r|] \leq \mathbb{E}[|L_1| \cdot |L_2|] \cdot p^{2r} + r = 2^{2\lambda d} p^{2r} + r = \mathcal{O}(\mathcal{L}_r^2) \tag{5.4}$$

Finally, the algorithm is repeated for P different permutations on the bit positions of elements in L_1, L_2. In summary, the expected time complexity to build all list becomes the sum of the T_i multiplied by P, thus, by choosing $N := \frac{d}{q}$ and $P = (d+1)^{r+1}$ we get

$$
\begin{aligned}
T' = P \cdot \sum_{i=1}^{r+1} T_i &\leq (d+1)^{r+1} \cdot \left(\sum_{i=1}^{r} N^i \cdot |L_1| \cdot p^{i-1} + (|L_1| \cdot p^r)^2 \cdot N^r \right) \\
&= (d+1)^{r+1} \cdot \left(\sum_{i=1}^{r} \frac{|L_1| \cdot d^i}{q} \cdot \left(\frac{p}{q}\right)^{i-1} + \frac{(|L_1| \cdot p^r)^2 \cdot d^r}{q^r} \right) \\
&\leq (d+1)^{2r+1} \cdot \left(\frac{r \cdot |L_1| \cdot p^{r-1}}{q^r} + \frac{(|L_1| \cdot p^r)^2}{q^r} \right) \\
&= \max\left(\frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{\left(2^{\lambda d} \cdot p^r\right)^2}{q^r} \right)^{1+o(1)} ,
\end{aligned}
$$

where the inequality follows from the fact that $\frac{p}{q} \geq 1$ since

$$
\begin{aligned}
q &= \Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}}) = \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}}) = \delta k \right] \\
&\leq \Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}}) = \delta k \right] \\
&= p \ ,
\end{aligned}
$$

and the final equality stems from the fact that $|L_1| = 2^{\lambda d}$ and $r = o(\frac{\lambda d}{\log d})$ as given in the theorem.

Note that T' disregards the fact that no matter how small the lists in the tree become, the algorithm needs to traverse all

$$
T'' = \tilde{\mathcal{O}}\left(N^r\right) = \tilde{\mathcal{O}}\left(\left(\frac{d}{q}\right)^r\right)
$$

nodes of the tree. Hence, the expected time complexity of the whole algorithm is

$$
T = \max(T', T'') \ ,
$$

which proves the claim.

Let us now consider the success probability of the algorithm. Therefore, we assume that the chosen permutation distributes the weight on $\mathbf{x}+\mathbf{y}$ such that in every block of length r the weight is equal to $\frac{\omega d}{r}$, which we describe as a *good* permutation. The probability of a random permutation π distributing the weight in such a way is

$$
\begin{aligned}
\Pr\left[\text{good } \pi\right] &= \Pr\left[\mathrm{wt}\left(\pi(\mathbf{x}+\mathbf{y})_{B_{i,r}}\right) = \frac{\omega d}{r}, \text{ for } i = 1, \ldots, r \right] \\
&= \frac{\binom{\frac{d}{r}}{\frac{\omega d}{r}}^r}{\binom{d}{\omega}} \geq \left(\frac{d}{r}+1\right)^{-r} \ .
\end{aligned}
$$

Thus, the probability of at least one out of $(d+1)^{r+1}$ chosen permutations being good is

$$
\begin{aligned}
p_1 &:= \Pr\left[\text{at least one good } \pi\right] \\
&= 1 - \left(1 - \Pr\left[\text{good } \pi\right]\right)^{(d+1)^{r+1}} \\
&= 1 - \left(1 - \left(\frac{d}{r}+1\right)^{-r}\right)^{(d+1)^{r+1}} \geq 1 - e^{-d} \ .
\end{aligned}
$$

The algorithm succeeds, whenever there exists a leaf node in the tree, containing the distinguished pair $(\mathbf{x}, \mathbf{y})$. As every node in the tree is constructed based on its parent, it follows that all nodes on the path from the root to that leaf need to contain $(\mathbf{x}, \mathbf{y})$. By definition the probability of $\mathbf{x}$ and $\mathbf{y}$ satisfying the bucket criterion at the same time (thus for the same $\mathbf{z}$) is q and since we condition on a good permutation, q is equal for every considered block. Let us define indicator variables X_j for the first level, where $X_j = 1$ iff the j-th node contains $(\mathbf{x}, \mathbf{y})$. Observe that the X_j for independent choices of $\mathbf{z}$ are independent. Thus, clearly the number of trials until $(\mathbf{x}, \mathbf{y})$ is contained in any node on level one is distributed geometrically with parameter q. Hence, the probability of the solution being contained in at least one node on the first level is

$$\begin{aligned} p_2 &:= \Pr\left[\exists (A, B, 1) \in S : (\mathbf{x}, \mathbf{y}) \in A \times B\right] \\ &= 1 - (1-q)^N = 1 - (1-q)^{\frac{d}{q}} \geq 1 - e^{-d} \ . \end{aligned}$$

Now, imagine the pair being contained in some level-i node. Considering that node, we have with the same probability p_2 again that at least one child contains the solution, and the same argument holds until we reach the leaves. Also, by the independent choices of $\mathbf{z}$ the events remain independent which implies that the probability of $(\mathbf{x}, \mathbf{y})$ being contained in a level-r list is p_2^r. In summary, the success probability is

$$\Pr\left[\text{success}\right] = p_1 \cdot p_2^r \geq (1 - e^{-d})^{r+1} \geq 1 - \frac{r+1}{e^d} \geq 1 - \frac{d}{e^d} \ .$$

□

The proof of Theorem 3 already shows, how different distributions may affect the complexity of the algorithm by changing the expected value $\mathbb{E}[|A_r| \cdot |B_r|]$. This influence on the algorithms complexity by different input distributions is further investigated in Section 5.4.

In the next two lemmata, we will proof the exact forms of q and p to conduct the run time analysis.

Lemma 4. *Let $k \in \mathbb{N}$, $\delta \in [0, 1]$. If $\mathbf{x} \in \mathbb{F}_2^k$ and $\mathbf{z} \sim \mathcal{U}(\mathbb{F}_2^k)$ then*

$$\Pr_{\mathbf{z}}\left[\mathrm{wt}(\mathbf{x} + \mathbf{z}) = \delta k\right] = \binom{k}{\delta k} \left(\frac{1}{2}\right)^k .$$

Proof. Since $\mathbf{z} \sim \mathcal{U}(\mathbb{F}_2^k)$, the probability is

$$\frac{\left|\{\mathbf{z} \in \mathbb{F}_2^k \mid \mathrm{wt}(\mathbf{x} + \mathbf{z}) = \delta k\}\right|}{\left|\mathbb{F}_2^k\right|}.$$

To compute the numerator, note that $\mathrm{wt}(\mathbf{x} + \mathbf{z}) = \delta k$ means that $\mathbf{x}$ and $\mathbf{z}$ differ in δk out of k coordinates, for which there are $\binom{k}{\delta k}$ possibilities. Using $\left|\mathbb{F}_2^k\right| = 2^k$, the lemma follows. □

Before we continue, let us make a small definition.

Definition 13. Let $k \in \mathbb{N}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^k$. Then we define $D(\mathbf{x}, \mathbf{y}) \subseteq [k]$ to be the set of coordinates where $\mathbf{x}$ and $\mathbf{y}$ differ, i.e.

$$D(\mathbf{x}, \mathbf{y}) := \{i \in [k] \mid \mathbf{x}_i \neq \mathbf{y}_i\}.$$

Furthermore, let $S(\mathbf{x}, \mathbf{y}) := [k] \setminus D(\mathbf{x}, \mathbf{y})$ be the set of coordinates where they are the same.

Now we derive the exact form of the probability q of a pair with difference ωk falling into the same bucket.

Lemma 5. *Let $k \in \mathbb{N}$, $\delta \in [0,1]$. If $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^k$ with $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \omega k$ and $\mathbf{z} \sim \mathcal{U}(\mathbb{F}_2^k)$. Then*

$$\Pr_{\mathbf{z}}\left[\mathrm{wt}(\mathbf{x}+\mathbf{z}) = \mathrm{wt}(\mathbf{y}+\mathbf{z}) = \delta k\right] = \binom{\omega k}{\frac{1}{2}\omega k}\binom{(1-\omega)k}{(\delta - \frac{\omega}{2})\,k}\left(\frac{1}{2}\right)^k.$$

Proof. Let

$$A := \{\mathbf{z} \in \mathbb{F}_2^k \mid \mathrm{wt}(\mathbf{x}+\mathbf{z}) = \mathrm{wt}(\mathbf{y}+\mathbf{z}) = \delta k\}.$$

In analogy to Lemma 4, the probability we search for is $\frac{|A|}{|\mathbb{F}_2^k|} = |A| \cdot \left(\frac{1}{2}\right)^k$.

In the following, let $\omega_{\mathbf{x}} := \mathrm{wt}(\mathbf{x}+\mathbf{z})$ and analogously $\omega_{\mathbf{y}} := \mathrm{wt}(\mathbf{y}+\mathbf{z})$. Now observe that every coordinate z_i of $\mathbf{z}$ with $i \in S(\mathbf{x},\mathbf{y})$, so belonging to the set of equal coordinates between $\mathbf{x}$ and $\mathbf{y}$, either contributes to both $\omega_{\mathbf{x}}$ *and* $\omega_{\mathbf{y}}$ does not affect either one of them. Let us define the amount of the z_i's with $i \in S(\mathbf{x},\mathbf{y})$ that contribute to the weight as $a := |S(\mathbf{x},\mathbf{y}) \cap D(\mathbf{x},\mathbf{z})|$.

Now consider the z_i's with $i \in D(\mathbf{x},\mathbf{y})$. Clearly, any such z_i contributes *either* to $\omega_{\mathbf{x}}$ *or* to $\omega_{\mathbf{y}}$. Thus, let us define the number of those z_i with $i \in D(\mathbf{x},\mathbf{y})$ that contribute to $\omega_{\mathbf{x}}$ as $b_{\mathbf{x}} := |D(\mathbf{x},\mathbf{y}) \cap D(\mathbf{x},\mathbf{z})|$ and analogously those which contribute to $\omega_{\mathbf{y}}$ as $b_{\mathbf{y}} := |D(\mathbf{x},\mathbf{y}) \cap D(\mathbf{y},\mathbf{z})|$. Obviously we have

$$b_{\mathbf{x}} + b_{\mathbf{y}} = |D(\mathbf{x},\mathbf{y})| = \omega k \tag{5.5}$$

On the other hand we are only interested in those $\mathbf{z}$ for which $\omega_{\mathbf{x}} = \omega_{\mathbf{y}} = \delta k$, which yields the two equations

$$\omega_{\mathbf{x}} = a + b_{\mathbf{x}} = \delta k \tag{5.6}$$
$$\omega_{\mathbf{y}} = a + b_{\mathbf{y}} = \delta k \tag{5.7}$$

All three equations together yield the unique solution

$$b_{\mathbf{x}} = b_{\mathbf{y}} = \frac{\omega k}{2} \text{ and } a = \left(\delta - \frac{\omega}{2}\right) k \ .$$

This shows the following: If $\mathbf{z} \in A$, it is necessary that $\mathbf{z}$ differs from $\mathbf{x}$ (analogously $\mathbf{y}$) in exactly

- $\frac{\omega}{2}k$ out of ωk coordinates of $D(\mathbf{x},\mathbf{y})$ and
- $\left(\delta - \frac{\omega}{2}\right) k$ out of $(1-\omega)k$ coordinates of $S(\mathbf{x},\mathbf{y})$.

Thus, because we can freely combine both conditions, in total there are

$$|A| = \binom{\omega k}{\frac{\omega}{2}k}\binom{(1-\omega)k}{(\delta - \frac{\omega}{2})\,k}$$

different values for $\mathbf{z}$, finishing the proof. □

Now we are ready to prove Theorem 2 about the time complexity of Algorithm 11 for solving the $\mathcal{CP}_{d,\lambda,\omega}$ problem. For convenience we restate the theorem here.

Proof. First let us give the exact form of $\log p$ and $\log q$ using Stirling's formula to approximate the binomial coefficients in Lemma 4 and 5. By setting the block width $k = \frac{d}{r}$ we get

$$\begin{aligned}\log q &= (1-\omega)\Big(H\big(\tfrac{\delta - \frac{\omega}{2}}{1-\omega}\big) - 1\Big)\frac{d}{r}(1+o(1)) \text{ and}\\ \log p &= \big(H(\delta) - 1\big)\frac{d}{r}(1+o(1)) \ .\end{aligned}$$

Now, let us reconsider the running time given in Theorem 3 as

$$T = \max\Big(\underbrace{\frac{1}{q^r}}_{(a)}, \underbrace{\frac{2^{\lambda d}\cdot p^{r-1}}{q^r}}_{(b)}, \underbrace{\frac{\big(2^{\lambda d}\cdot p^r\big)^2}{q^r}}_{(c)}\Big)^{1+o(1)} \ ,$$

where $r = \frac{\lambda d}{\log^2 d}$.

We now show that the running time for all values of $\delta \geq \delta^\star := H^{-1}(1-\lambda)$ is solely dominated by (c). Observe that we have $(c) \geq (b)$, whenever

$$\begin{aligned} & 2^{\lambda d}\cdot p^{2r} \geq p^{r-1}\\ \Leftrightarrow \quad & H(\delta) \geq 1 - \frac{\lambda r}{r+1}\\ \Leftrightarrow \quad & \delta \geq H^{-1}\Big(1 - \frac{\lambda}{1+\frac{1}{r}}\Big) \to H^{-1}(1-\lambda) = \delta^\star \ ,\end{aligned}$$

since $\frac{1}{r} = o(1)$. Also we have $(c) \geq (a)$ for the same choice of delta, as

$$\begin{aligned} & 2^{2\lambda d}\cdot p^{2r} \geq 1\\ \Leftrightarrow \quad & \delta \geq H^{-1}(1-\lambda) = \delta^\star \ .\end{aligned}$$

Thus, for all choices of $\delta \geq \delta^\star$ the running time is $(T_\delta)^{(1+o(1))}$ with

$$\vartheta^\star(\delta) := \frac{\log T_\delta}{d} = 2(\lambda + H(\delta) - 1) + (1-\omega)\Big(1 - \mathrm{H}\big(\tfrac{\delta - \frac{\omega}{2}}{1-\omega}\big)\Big) \ .$$

Now, minimizing $\vartheta^\star$ yields a global minimum at $\delta_{\min} = \frac{1}{2}(1 - \sqrt{1-2\omega})$ attaining a value of

$$\vartheta^\star(\delta_{\min}) = 2\lambda + H(\omega) - 1 \ .$$

As we are restricted to values for δ which are larger than $\delta^\star$ solving $\delta_{\min} \geq \delta^\star$ for ω yields

$$\begin{aligned} & \delta_{\min} \geq \delta^\star\\ \Leftrightarrow \quad & \omega \geq 2\delta^\star(1-\delta^\star) = \omega^\star \ .\end{aligned}$$

This proves the claim of the theorem whenever $\omega > \omega^\star$. For all other values of ω we simply choose $\delta = \delta^\star$, which yields

$$\vartheta = \vartheta^\star(\delta^\star) = (1-\omega)\Big(1 - \mathrm{H}\big(\tfrac{\delta^\star - \frac{\omega}{2}}{1-\omega}\big)\Big) \text{ for } \omega \leq \omega^\star$$

as claimed.

Now to boost the expected running time $2^{\vartheta d(1+o(1))}$ of the algorithm to actually being obtained with overwhelming probability we use a standard Markov argument. Let X denote the random variable describing the running time of the algorithm. Then the probability that the algorithm needs more time than $2^{\sqrt{d}}E[X]$ to finish is

$$\Pr\left[X \geq 2^{\sqrt{d}} \cdot E[X]\right] \leq \frac{E[X]}{2^{\sqrt{d}} \cdot E[X]} = 2^{-\sqrt{d}} \ ,$$

or equivalently the algorithm finishes in less time than $2^{\sqrt{d}}E[X] = 2^{\vartheta d(1+o(1))}$ with overwhelming probability. Also, a standard application of the union bound yields that the intersection of the algorithm finishing within the claimed time and the algorithm having success in finding the solution is still overwhelming.

□

The theorem shows that whenever $\omega > \omega^*$ our algorithm obtains the optimal time complexity for uniformly random lists as given in Equation (5.1). Additionally, our algorithm reaches the time lower bound for locality-sensitive hashing based algorithms for all values of ω, whenever the input list sizes are subexponential in the dimension d, which is shown in the following lemma.

Lemma 6. *Let $\omega \in \left[0, \frac{1}{2}\right]$, and ϑ as defined in Theorem 2. Then we have*

$$\lim_{\lambda \to 0} \frac{\vartheta}{\lambda} = \frac{1}{1-\omega} \ .$$

Proof. Note that for λ converging zero, $\delta^\star = H^{-1}(1-\lambda)$ approaches $\frac{1}{2}$. This implies $\omega^\star := 2\delta^\star(1-\delta^\star) = \frac{1}{2}$ and hence for all choices of ω we have

$$\vartheta = (1-\omega)\left(1 - \mathrm{H}\left(\frac{\delta - \frac{\omega}{2}}{1-\omega}\right)\right) \ .$$

Now, for this choice of ϑ, May and Ozerov [MO15, Corollary 1] already showed the statement of this lemma, by applying L'Hoptial's rule twice.

□

For convenience we restate all parameter choices of Algorithm 11 for solving the $\mathcal{CP}_{d,\lambda,\omega}$ in the following overview:

$$\begin{aligned}
&r = \frac{d}{\log^2 d},\ P = (d+1)^{r+1},\ k = \frac{d}{r} \\
&N = \frac{d}{q},\ \text{where } q = \binom{\omega k}{\frac{1}{2}\omega k}\binom{(1-\omega)k}{(\delta - \frac{\omega}{2})\,k}\left(\frac{1}{2}\right)^k \\
&\delta = \begin{cases} \delta^\star & \text{for } \omega \leq 2\delta^\star(1-\delta^\star) \\ \frac{1}{2}(1-\sqrt{1-2\omega}) & \text{else} \end{cases}\ ,\ \text{with } \delta^\star := H^{-1}(1-\lambda)
\end{aligned} \tag{5.8}$$

5.4 Different Input Distributions

In this section, we show how to adapt the analysis of Algorithm 11 to variable input distributions. Therefore, we first reformulate Theorem 3 in Corollary 2 for the case of considering the $\mathcal{CP}_{d,\lambda,\omega}$ over an arbitrary distribution $\mathcal{D}$. As already indicated in the proof of Theorem 3, this reformulation depends on the expected value $\mathcal{E}$ of the cost of the naive search at the bottom of the computation tree, which is highly influenced by the distribution $\mathcal{D}$. Then, we show how to compute $\mathcal{E}$ and how to upper bound it effectively. Finally, we give upper bounds for the time complexity of the algorithm to solve the $\mathcal{CP}_{d,\lambda,\omega}$ over some generic distributions. These examples suggest that the algorithm is best suited for distributions $\mathcal{D}$, where the weight of the sum $\mathbf{v}+\mathbf{w}$ of elements $\mathbf{v},\mathbf{w}\sim\mathcal{D}$ concentrates at $\frac{d}{2}$.[3]

Let us start with the reformulation of the theorem.

Corollary 2. *Let $\mathcal{D}$ be some distribution over $\mathbb{F}_2^d$, q and p be as defined in Equation (5.2), $\omega\in\left[0,\frac{1}{2}\right]$, $\lambda\in[0,1]$ and $r=\frac{\lambda d}{\log^2 d}$. Also let $\mathcal{E}=\mathbb{E}[|A|\cdot|B|]$ for A and B in line 15 of Algorithm 11 (where the expectation is taken over the distribution of input lists and the random choices of the algorithm). Then Algorithm 11 solves the $\mathcal{CP}_{d,\lambda,\omega}$ problem over $\mathcal{D}$ in time*

$$\max\left(q^{-r},\frac{2^{\lambda d}\cdot p^{r-1}}{q^r},\frac{\mathcal{E}}{q^r}\right)^{1+o(1)}$$

with success probability overwhelming in d.

Proof. The proof follows along the lines of the proof of Theorem 3, by observing that $T_{r+1}=N^r\cdot\mathcal{E}$ and the expected time complexity is again amplified to being obtained with overwhelming probability by using a Markov argument similar to the proof of Theorem 2. □

In the next lemma, we show how to upper bound the value of $\mathcal{E}$.

Lemma 7 (Expectation of Naive Search)**.** *Let $\mathcal{D}$ be some distribution over $\mathbb{F}_2^d$, $\omega\in\left[0,\frac{1}{2}\right]$, $\lambda\in[0,1]$ and $r=\frac{\lambda d}{\log^2 d}$. Also let $\mathcal{E}=\mathbb{E}[|A|\cdot|B|]$ for A and B in line 15 of Algorithm 11 when solving some instance of the $\mathcal{CP}_{d,\lambda,\omega}$ over $\mathcal{D}$ (where the expectation is taken over the distribution of input lists and the random choices of the algorithm). Then we have*

$$\mathcal{E}\le 2^{2\lambda d}\prod_{i=1}^{r}\alpha_i+4r\cdot 2^{\lambda d}\cdot p^r$$

where $\alpha_i:=\Pr_{\mathbf{v},\mathbf{w}\sim\mathcal{D}}\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})=\delta k,\mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}})=\delta k\right]$.

Proof. Given in Section 5.6. □

While Lemma 7 gives an upper bound on the required expectation, it is not very handy. In the next lemma, we show how to further bound this expectation and how it affects the running time of the algorithm.

[3] This behavior seems quite natural as in this case, the solution is most distinguishable from random input pairs.

Lemma 8 (Complexity for Arbitrary Distributions)**.** *Let $\mathcal{D}$ be some distribution over $\mathbb{F}_2^d$, $r := \frac{\lambda d}{\log^2 d}$, $\omega \in \left[0, \frac{1}{2}\right]$ and $\lambda \in [0,1]$. Also let $\mathcal{E} = \mathbb{E}[|A| \cdot |B|]$ for A and B in line 15 of Algorithm 11 when solving some instance of the $\mathcal{CP}_{d,\lambda,\omega}$ over $\mathcal{D}$ (where the expectation is taken over the distribution of input lists and the random choices of the algorithm). Then Algorithm 11 solves the $\mathcal{CP}_{d,\lambda,\omega}$ over $\mathcal{D}$ in time*

$$\max\left(q^{-r}, \frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{2^{\varepsilon d}}{q^r}\right)^{1+o(1)} ,$$

where

$$\varepsilon = 2\lambda - \min_{\substack{i \in [r] \\ \gamma \in [0,1]}} (1-\gamma)\left(1 - H\left(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\right)\right) - \frac{r \cdot \log p_{i,\gamma k}}{d}$$

with $p_{i,\gamma k} := \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = \gamma k\right]$.

Proof. Given in Section 5.6. □

Note that if it further holds that for $\mathbf{v} \sim \mathcal{D}$ each of the r blocks of $\mathbf{v}$ is identically distributed we can further simplify the term of ε from Lemma 8. In this case, we have $p_{i,\gamma k}^r \leq \Pr[\mathrm{wt}(\mathbf{v}+\mathbf{w}) = \gamma d] := p_{\gamma d}$, thus we get

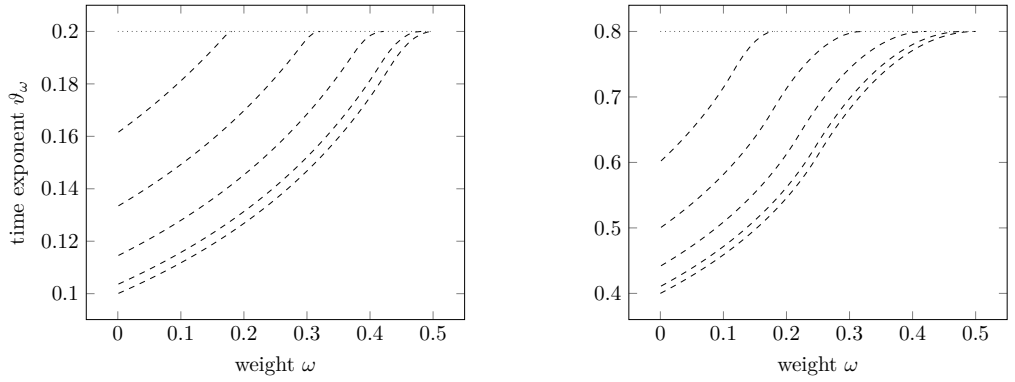
$$\varepsilon = 2\lambda - \min_{\gamma \in [0,1]} (1-\gamma)\left(1 - \mathrm{H}\left(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\right)\right) - \frac{\log p_{\gamma d}}{d} .$$

Now if we are given an arbitrary distribution $\mathcal{D}$ we can maximize ε according to γ. Then we can similar to the proof of Theorem 2 derive a value for δ minimizing the overall time complexity.

We performed this maximization and optimization numerically for some generic input distributions. We considered distributions, where the weight of input vectors is distributed *binomially*, chosen according to a *Poisson* distribution or *fixed* to a specific value. This means, first a weight is sampled according to the chosen distribution and then a vector of that weight is selected uniformly among all vectors of that weight.

The running time of Algorithm 11 for solving the $\mathcal{CP}_{d,\lambda,\omega}$ over the considered distributions seems to be only dependent on the expected weight of vectors contained in the input lists. That means the time complexity for input lists containing random vectors whose weight is either fixed to γd or binomially or Poisson distributed with expectation γd is equal. This can possibly be explained by the low variance of all these distributions, which implies a high concentration around this expected weight.

We see in Figure 5.3, that the value for ω, from where on the complexity becomes quadratic in the lists sizes shifts to the left. This behavior stems from the fact, that the expected weight of a sum of elements is no longer $\frac{d}{2}$, but roughly $2\gamma(1-\gamma)d$. What also stands out is, that the complexity for $\omega = 0$ is no longer linear in the lists sizes. The reason for this is that the probability of random pairs falling into the same bucket and the probability of the closest pair falling into the same bucket converge for decreasing weight of input list elements. This indicates that for input distributions with smaller expected weight a different bucketing criterion might be beneficial. We pose this as an open question for further research.



(a) List sizes $|L_1| = |L_2| = 2^{0.1d}$

(b) List sizes $|L_1| = |L_2| = 2^{0.4d}$

Figure 5.3: Time complexity exponents as a function of the weight of the closest pair for different input list distributions, where the expected weight of input elements is equal to $0.1d$, $0.2d$, $0.3d$, $0.4d$, $0.5d$ from left to right.

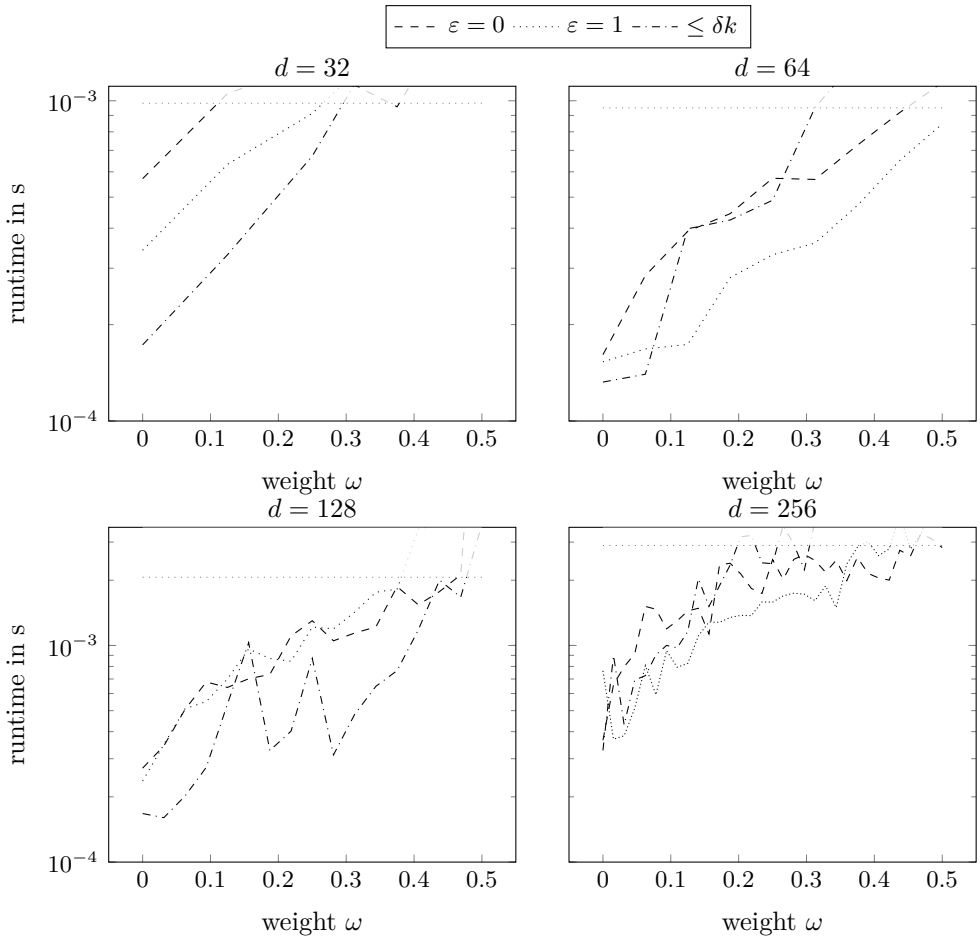


Figure 5.4: Runtime results in seconds in logarithmic scale (y-axis) as the function of the distance ω of the closest pair (x-axis) on random input lists of size 2^{10}. The dotted, dashed and dash-dotted lines indicate the runtime results for the different bucketing strategies used. The straight horizontal line is the time used by a naive quadratic search.

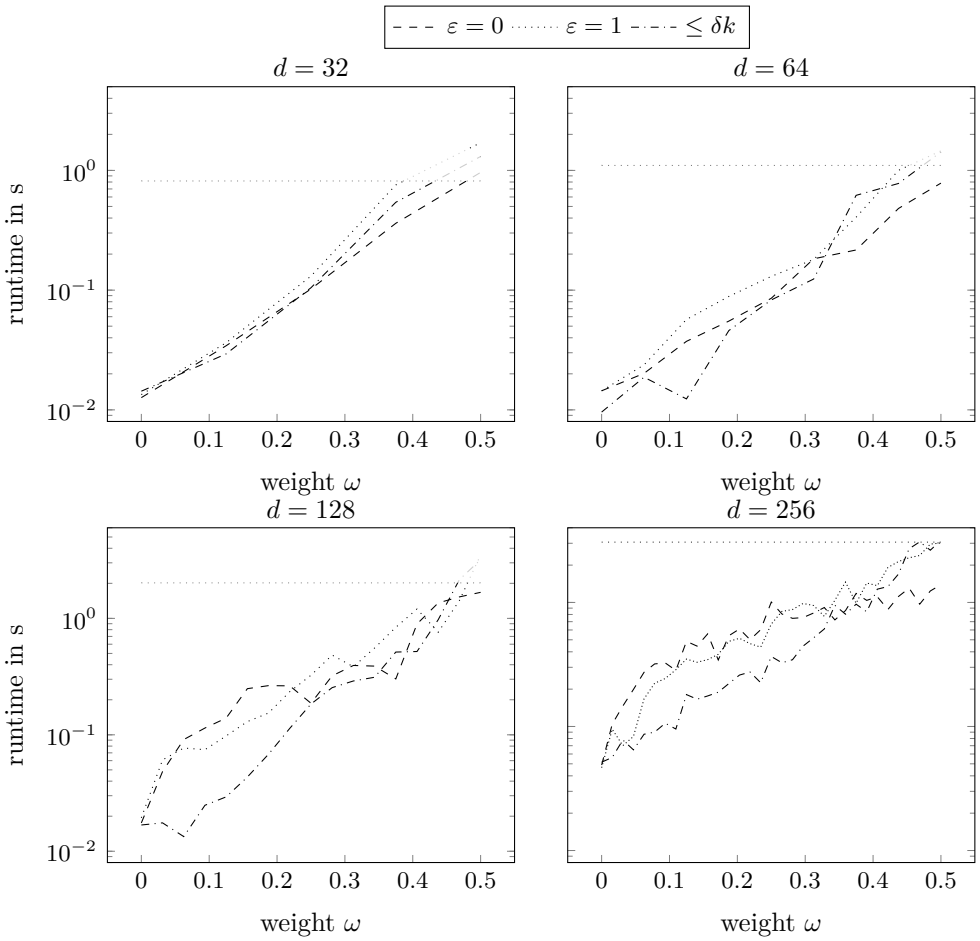


Figure 5.5: Runtime results in seconds in logarithmic scale (y-axis) as the function of the distance ω of the closest pair (x-axis) on random input lists of size 2^{15}. The dotted, dashed and dash-dotted lines indicate the runtime results for the different bucketing strategies used. The straight horizontal line is the time used by a naive quadratic search.

5.5 Practical Experiments

In this section, we give experimental results of the performance of a proof of concept implementation of our new algorithm. These experiments verify the performance gain of our algorithm over a naive quadratic search approach. We also verify the numerical estimates of the algorithm's performance on different input distributions from the previous section and give some practical related improvements to our algorithm. Our implementation is publicly available at `https://github.com/FloydZ/NNAlgorithm`.

Before discussing the benchmark results let us first briefly describe some of the practical improvements we introduced in our implementation, which differ from the description in Section 5.3. We implemented a true depth-first search rather than the iterative description given previously. The iterative description just allowed for a more convenient analysis. Thus, our algorithm needs to store only the lists of a single path from the root to a leaf node at any time. Also, as all lists of subsequent levels are subsets of previous ones, we do not create r different lists. We rather rearrange the elements of the input list such that elements belonging to the list of the subsequent level are consecutive, making it sufficient to just memorize the

range of elements that belong to the next level list. This way, we only need to store the input list plus two integer markers for each level.

Also, it turns out that in practice often a small depth of the tree (not exceeding 8 in our experiments) is already sufficient to achieve good runtime results. Regarding the branching factor N of the tree, we achieve optimal results either for values close to its expectation $\frac{1}{q}$ as given by the analysis or values being significantly smaller. The case of using a very small branching factor can be seen as a pruning strategy, similar to the one used in lattice enumeration algorithms for shortest vector search [ANSS18]. Additionally, we benchmarked three different strategies for the weight criteria:

1. Strictly enforcing a weight of δk in each block, as described in our algorithm.

2. Allowing for a small deviation $\pm\varepsilon$ around δk.

3. Allowing for weights of *at most* δk.

Additionally, we introduced a threshold for the size of the lists in the tree from where the computation of further leaves is aborted and naive search is used instead.

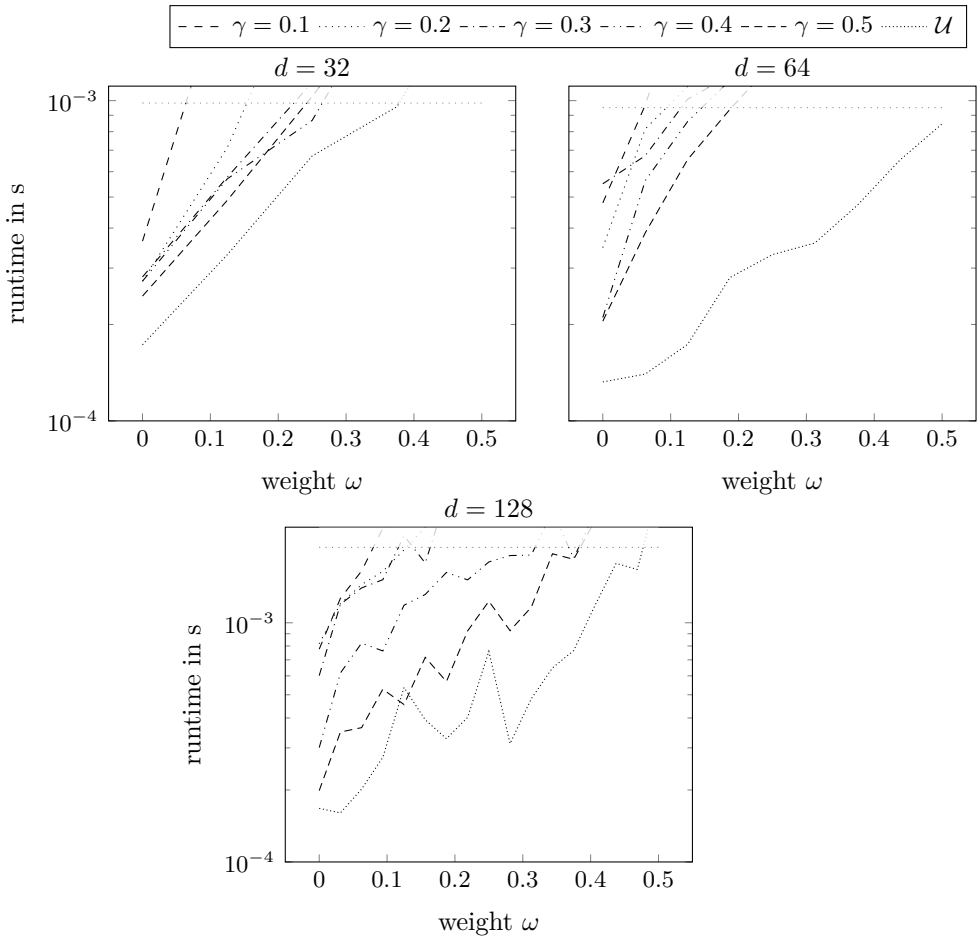


Figure 5.6: Runtime results in seconds in logarithmic scale (y-axis) as the function of the distance ω of the closest pair (x-axis) on input lists of size 2^{10} containing random elements of weight γd. The densely dashed line ($\mathcal{U}$) indicates the runtime on uniformly random lists.

Figure 5.4 shows the runtime results for the different bucket criteria on small input lists of size 2^{10} containing random elements. Here, each data point was averaged over 50 measurements. The experimental results clearly indicate a significant gain over the quadratic search approach. The less significant gain for small dimension d is due to the reduced amount of possible blocks or equivalently the low depth of the computation tree, which lets the algorithm not reach its full potential. In the case of small input lists, we observe that a bucketing strategy that allows a deviation of $\varepsilon = 1$ from δk is beneficial for most values of d.

Figure 5.5 shows the same experiments performed on larger input lists of size 2^{15}. Besides a more significant improvement over the naive search, we can observe that the bucketing criterion that uses δk as an upper bound becomes more beneficial for nearly all values of ω and d.

Eventually, Figure 5.6 shows the experimental runtime results on input lists, whose elements are drawn from a different input distribution, analyzed in Section 5.4. Here the distribution is the uniformly random distribution over vectors of weight γd. One can observe that for growing d the shape of the graph resembles the theoretical results from Figure 5.3.

5.6 Proofs for General Distributions

In this section we give the proofs for the lemmata regarding the performance of our algorithm on different input distributions, which were omitted in the main body of the paper.

Proof of Lemma 7 Similar to the proof of Theorem 3, let us bound $\mathbb{E}[|A_i| \cdot |B_i|]$ in terms of $\mathbb{E}[|A_{i-1}| \cdot |B_{i-1}|]$, $\mathbb{E}[|A_{i-1}|]$ and $\mathbb{E}[|B_{i-1}|]$ for each i.

$$\begin{aligned}
\mathbb{E}[|A_i| \cdot |B_i| \mid A_{i-1}, B_{i-1}] = &\sum_{\substack{\mathbf{v}\in A_{i-1}\setminus\{\mathbf{x}\}\\ \mathbf{w}\in B_{i-1}\setminus\{\mathbf{y}\}}} \underbrace{\Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}}) = \delta k\right]}_{=:\alpha_i} \\
&+ \sum_{\mathbf{v}\in A_{i-1}} \underbrace{\Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}}) = \delta k\right]}_{\leq p} \\
&+ \sum_{\mathbf{w}\in B_{i-1}} \underbrace{\Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}}) = \delta k\right]}_{\leq p} \\
&+ \Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}})) = \delta k, \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \\
&\leq \alpha_i \cdot |A_{i-1}| \cdot |B_{i-1}| + p \cdot (|A_{i-1}| + |B_{i-1}| + 1)
\end{aligned}$$

and hence $\mathbb{E}[|A_i| \cdot |B_i|] \leq \alpha_i \cdot \mathbb{E}[|A_{i-1}| \cdot |B_{i-1}|] + p \cdot (\mathbb{E}[|A_{i-1}|] + \mathbb{E}[|B_{i-1}|] + 1)$. Again, applying this equation successively, we obtain

$$\mathcal{E} = \mathbb{E}[|A_r| \cdot |B_r|] \leq 2^{2\lambda d} \prod_{i=1}^{r} \alpha_i + 4 \cdot 2^{\lambda d} \cdot \sum_{i=1}^{r} \left(\prod_{j=0}^{i-2} \alpha_{r-j} \right) p^{r-i+1} \leq 2^{2\lambda d} \prod_{i=1}^{r} \alpha_i + 4r \cdot 2^{\lambda d} \cdot p^r \quad . \quad \square$$

Proof of Lemma 8 Taking the result for $\mathcal{E}$ from Lemma 7 and plugging into the run time formula from Corollary 2 we get that the $\mathcal{CP}_{d,\lambda,\omega}$ problem over $\mathcal{D}$ can be solved with probability overwhelming in d in time

$$\max\left(q^{-r}, \frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{\mathcal{E}}{q^r}\right)^{1+o(1)} \leq \max\left(q^{-r}, \frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{2^{2\lambda d} \prod_{i=1}^{r} \alpha_i}{q^r}\right)^{1+o(1)}$$

since the right summand $\frac{4r \cdot 2^{\lambda d} \cdot p^r}{q^r}$ of $\frac{\mathcal{E}}{q^r}$ is asymptotically smaller than the second entry in the max, i.e. $\frac{2^{\lambda d} \cdot p^{r-1}}{q^r}$. Thus, is suffices to find an easier upper bound for the first summand $S := 2^{2\lambda d} \prod_{i=1}^{r} \alpha_i$. Remembering $\alpha_i = \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}}) = \delta k\right]$ we receive

$$\begin{aligned}
S &\leq 2^{2\lambda d} \cdot \left(\max_{i \in [r]} \alpha_i\right)^r \\
&= 2^{2\lambda d} \cdot \left(\max_{i \in [r]} \sum_{j=0}^{k} q_{i,j} \cdot \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = j\right]\right)^r \\
&\leq 2^{2\lambda d + o(d)} \cdot \left(\max_{i \in [r],\, j \in [k] \cup \{0\}} q_{i,j} \cdot \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = j\right]\right)^r \\
&= 2^{2\lambda d + o(d)} \cdot \left(\max_{i \in [r],\, \gamma \in [0,1]} q_{i,\gamma k} \cdot \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = \gamma k\right]\right)^r \quad ,
\end{aligned}$$

where $q_{i,\gamma k} = \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}}) = \delta k \mid \mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = \gamma k\right]$. Lemma 5 lets us rewrite this probability as

$$q_{i,\gamma k} = \binom{\gamma k}{\frac{1}{2}\gamma k} \binom{(1-\gamma)k}{(\delta - \frac{\gamma}{2})\, k} \left(\frac{1}{2}\right)^k \leq 2^{-\left(1 - \mathrm{H}\left(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\right)\right)(1-\gamma)k} \quad .$$

We end up with

$$\begin{aligned}
S &\leq 2^{2\lambda d + r \cdot \max\limits_{i \in [r],\, \gamma \in [0,1]} -\left(1 - \mathrm{H}\left(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\right)\right)(1-\gamma)k + \log p_{i,\gamma k} + o(d)} \\
&= 2^{\left(2\lambda + \max\limits_{i \in [r],\, \gamma \in [0,1]} -\left(1 - \mathrm{H}\left(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\right)\right)(1-\gamma) + \frac{r}{d} \cdot \log p_{i,\gamma k}\right) d + o(d)} \\
&= 2^{\left(2\lambda - \min\limits_{i \in [r],\, \gamma \in [0,1]} \left(1 - \mathrm{H}\left(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\right)\right)(1-\gamma) - \frac{r}{d} \cdot \log p_{i,\gamma k}\right) d + o(d)}
\end{aligned}$$

with $p_{i,\gamma k} := \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = \gamma k\right]$, which proves the claim. $\square$

6 Legendre PRF (multiple) key attacks and the power of preprocessing

Due to its amazing speed and multiplicative properties the Legendre PRF recently finds widespread applications e.g. in Ethereum 2.0, multiparty computation and in the quantum-secure signature proposal LegRoast. However, its security is not yet extensively studied.

The Legendre PRF computes for a key k on input x the Legendre symbol $L_k(x) = \left(\frac{x+k}{p}\right)$ in some finite field $\mathbb{F}_p$. As standard notion, PRF security is analysed by giving an attacker oracle access to $L_k(\cdot)$. Khovratovich's collision-based algorithm recovers k using $L_k(\cdot)$ in time $\sqrt{p}$ with constant memory. It is a major open problem whether this birthday-bound complexity can be beaten.

We show a somewhat surprising wide-ranging analogy between the discrete logarithm problem and Legendre symbol computations. This analogy allows us to adapt various algorithmic ideas from the discrete logarithm setting.

More precisely, we present a small memory multiple-key attack on m Legendre keys $k_1, \ldots, k_m$ in time $\sqrt{mp}$, i.e. with amortized cost $\sqrt{p/m}$ per key. This multiple-key attack might be of interest in the Ethereum context, since recovering many keys simultaneously maximizes an attacker's profit.

Moreover, we show that the Legendre PRF admits precomputation attacks, where the precomputation depends on the public p only – and not on a key k. Namely, an attacker may compute e.g. in precomputation time $p^{\frac{2}{3}}$ a hint of size $p^{\frac{1}{3}}$. On receiving access to $L_k(\cdot)$ in an online phase, the attacker then uses the hint to recover the desired key k in time only $p^{\frac{1}{3}}$. Thus, the attacker's online complexity again beats the birthday-bound.

In addition, our precomputation attack can also be combined with our multiple-key attack. We explicitly give various trade-offs between precomputation and online phase. E.g. for attacking m keys one may spend time $mp^{\frac{2}{3}}$ in the precomputation phase for constructing a hint of size $m^2p^{\frac{1}{3}}$. In an online phase, one then finds *all m keys in total time* only $p^{\frac{1}{3}}$.

Precomputation attacks might again be interesting in the Ethereum 2.0 context, where keys are frequently changed such that a heavy key-independent precomputation pays off.

The content of this chapter is the result of a collaboration with Alexander May. It previously appeared as Legendre PRF (multiple) key attacks and the power of preprocessing in Computer Security Foundations Symposium (CSF) and is reproduced here with permission.

6.1 Introduction

6.1.1 Motivation

Blockchain technology received enormous attention over the past years by enabling secure, decentralized payments and multi party computations. One of the most famous and powerful implementation of this technology is the Ethereum Blockchain. The newly proposed Ethereum

2.0 protocol [Fou20a, Fou20b] tries to increase throughput for validating transactions in order to become competitive with modern credit card transaction systems.

Throughput is increased by moving away from the energy-consuming Proof-of-Work approach to a Proof-of-Stake. As opposed to Proof-of-Work, in Proof-of-Stake a user's voting power is not tied to its computing power, but to the stake he owns. If the Legendre PRF does not provide sufficient security, a malicious user u may let another user $\bar{u}$ with Legendre key $\bar{k}$ download and validate transactions. User u then recovers $\bar{u}$'s secret key $\bar{k}$ in order to maliciously claim $\bar{u}$'s reward.

More recent Legendre PRF applications are in multi-party computation [GRR+16], and in designing quantum secure signatures [BD20].

6.1.2 Related work on Legendre PRF security

Let p be a prime and $x \in \mathbb{F}_p$. We call x a *quadratic residue* in the finite field's multiplicative group $\mathbb{F}_p^*$, if there exists an $y \in \mathbb{F}_p^*$ with $y^2 = x$. We define the Legendre symbol

$$\left(\frac{x}{p}\right) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x \text{ is a quadratic residue} \\ -1 & \text{else.} \end{cases}$$

It is well-known that $\left(\frac{x}{p}\right) = x^{\frac{p-1}{2}} \bmod p$. The multiplicativity of the Legendre symbol follows directly.

Choose a key $k \in \mathbb{F}_p$. Then the *Legendre PRF*, as proposed by Damgård [Dam88], is the function $L_k : \mathbb{F}_p \to \{-1, 0, 1\}$ with

$$L_k(x) = \left(\frac{x+k}{p}\right).$$

Therefore, the Legendre PRF satisfies for all $i \in \mathbb{F}_p$

$$L_k(x+i) = \left(\frac{x+k+i}{p}\right) = L_0\left(x+k+i\right).$$

Conversely, if $L_k(x+i) = L_0(y+i)$ for sufficiently many i, then we can conclude that $y = x + k \bmod p$. Thus, finding x, y satisfying the identity $L_k(x+i) = L_0(y+i)$ for sufficiently many i gives us a way to compute $k = y - x \bmod p$. We call $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ a *collision* between the two functions $L_k(\cdot)$ and $L_0(\cdot)$ if $L_k(x+i) = L_0(y+i)$ for all $0 \leq i < \lceil 3 \log p \rceil$. We show in our work that, assuming Legendre PRF security, enforcing the identity at $\lceil 3 \log p \rceil$ points is sufficient to guarantee that a collision (x, y) yields the secret key $k = y - x$.

Notice that evaluation of $L_0(\cdot)$ is possible using the public p only, whereas evaluation of $L_k(\cdot)$ requires oracle access. Oracle access realization is the usual cryptographic attack model for PRFs – which is for conservative reasons quite strong, and not always satisfied in practical applications.

Khovratovich [Kho19] defined a memoryless algorithm with $L_k(\cdot)$ oracle access for recovering a Legendre key k within the typical birthday-type time bound $\tilde{\mathcal{O}}(\sqrt{p})$, where the $\tilde{\mathcal{O}}$-notation suppresses factors polynomial in $\log p$. In the less strong attack model without oracle access, but only $M \leq p^{\frac{1}{4}}$ evaluations of $L_k(\cdot)$ on known points, Beullens, Beyne, Udovenko, Vitto [BBUV20] and Kaluderovic, Kleinjung, Kostic [KKK20] proposed an algorithm with inferior time complexity $\mathcal{O}(p \log^2 p / M^2)$.

It was left as an open problem, whether the $\sqrt{p}$ bound can be beaten with classical algorithms. We answer this question in the affirmative, when we either allow for (more expensive) precomputations that *do not require* $L_k(\cdot)$ oracle access, and/or allow for amortized cost per key in multiple-key attacks.

On quantum computers, Russell and Shparlinski [RS04] showed that k can be recovered in polynomial time given oracle access to a quantum embedding of $L_k(\cdot)$ that can be asked in superposition – a *very strong* and in practical settings sometimes questionable attack model.

6.1.3 Oracle-access based Attack Model

Our results can be seen as a generalization of Khovratovich's memory-less algorithm [Kho19] that also uses $L_k(\cdot)$ oracle access. Most practical scenarios that we are aware of however do not provide such a strong attack model.

E.g. in Ethereum 2.0 the so-called *Proof-of-custody* for user u with secret key k works as follows. User u downloads periodically public data m_i, hashes to $h(m_i) \in \mathbb{F}_p$, and publishes the bit $L_k(h(m_i))$. After a certain time period, all users reveal their secret key k. User u can claim a reward on data m_j only if all bits $L_k(h(m_i))$ verify correctly for all published m_i within this time period. Hence, an attacker obtains evaluations of $L_k(\cdot)$ only on random known points $h(m_i)$, rather than points of his choice.

A similar attack scenario applies for the LegRoast signature scheme [BD20] that is based on the MPC-in-the-head paradigm [IKOS09]. Here, a user u's public key is an n-bit string $(L_k(x_1), \ldots, L_k(x_n))$, where k is u's secret key, and the x_i are public and randomly chosen in $\mathbb{F}_p$. Again, an attacker obtains evaluations of $L_k(\cdot)$ on random known points x_i.

The setting, where an attacker obtains PRF evaluations on known (random) points is called *known plaintext attack* in the literature. Many practical PRF applications, e.g. also for AES, only allow for known plaintext attacks. Nevertheless, for PRFs the well-established standard security notion is a *chosen plaintext attack* (CPA) that allows an attacker to query $L_k(\cdot)$ on points adaptively chosen by himself, i.e., an attacker receives $L_k(\cdot)$ oracle access.

Since PRFs are widely applied in practice in various scenarios, it is crucial to establish security even against the stronger CPA type. In fact, our algorithms directly use adaptive CPA queries, e.g. for achieving small memory consumption. Thus, our cryptanalytic results are of interest to study the security of Ethereum 2.0 and LegRoast, but do not directly lead to an attack on these.

6.1.4 Our contributions

Legendre PRF vs dlog

Let us first discuss the analogy between attacking the Legendre PRF via collisions and collision-based discrete logarithm algorithms. Let G be a discrete logarithm group of order q with generator g, and let $h = g^{k'}$ be a discrete logarithm instance. By finding (x, y) such that $hg^x = g^y$, we compute the discrete logarithm $k' = y - x \bmod q$, analogous to the Legendre setting.

Just as $L_k(\frac{x+i}{p}) = L_0(\frac{y+i}{p})$, for sufficient many i, the identity $hg^x = g^y$ is asymmetric in the sense that only the left-hand size depends on the secret discrete logarithm k', whereas the right-hand side can be computed solely based on the group specification. This asymmetry is used in *precomputation attacks* on the discrete logarithm as introduced in Mihalcik [Mih10] Lee, Cheon, Hong [LCH11] and Bernstein, Lange [BL12], where one performs a (rather large) precomputation that depends on the group only, and outputs a (rather small) hint. Upon

receiving a discrete logarithm instance, one then determines the unknown k' more quickly using the hint. Various tradeoffs are possible, e.g. within precomputation time $\tilde{\mathcal{O}}(q^{\frac{2}{3}})$ one can compute a hint of size $\tilde{\mathcal{O}}(q^{\frac{1}{3}})$. Upon receiving $h = g^{k'}$, the hint then allows to determine k' in time only $\tilde{\mathcal{O}}(q^{\frac{1}{3}})$.

Legendre Precomputation Attack

As already pointed out, in the Legendre setting the identity $L_k(\frac{x+i}{p}) = L_0(\frac{y+i}{p})$ offers a similar asymmetry. The identity's right-hand side depends on p only and thus allows for precomputation, whereas computation of the left-hand side requires $L_k(\cdot)$ oracle access. Hence, it might not come as a surprise that we obtain a similar Legendre key precomputation attack. Analogous, we may spend $\tilde{\mathcal{O}}(p^{\frac{2}{3}})$ time to compute a hint of size $\tilde{\mathcal{O}}(p^{\frac{1}{3}})$. Upon receiving access to $L_k(\cdot)$, we then compute the secret k using only $\tilde{\mathcal{O}}(p^{\frac{1}{3}})$ queries to $L_k(\cdot)$.

Not only does our precomputation attack break the $\sqrt{p}$-bound for recovering Legendre keys – in the online phase, once we have precomputed our hint. Our attack also accounts for scenarios that only offer limited number of $L_k(\cdot)$-queries. Similar to the discrete logarithm setting, we get for Legendre keys various tradeoffs between precomputation, key recovery phase and success probability.

Legendre Multiple-Key Attack

In the discrete logarithm setting, it was first noticed by Kuhn and Struick [KS01] using ideas from Escott, Sager, Selkirk, Tsapakidis [ESST99] that m discrete logarithm instances $h_1 = g^{k'_1}, \ldots, h_m = g^{k'_m}$ can be solved memory-less more efficiently than naively applying Pollard's $\tilde{\mathcal{O}}(\sqrt{p})$-algorithm m times. Namely, Kuhn and Struick showed that reusing the data structure for h_1, the discrete logarithm of h_2 can be found slightly more efficient, and so on. In total, *all m discrete logarithm instances* can be computed in time $\mathcal{O}(\sqrt{mp})$.

Again, the multiple-key discrete logarithm setting transfers to the Legendre PRF world. Namely, we are able to extend Khovratovich's $\tilde{\mathcal{O}}(\sqrt{p})$-algorithm — the Legendre variant of Pollard — to a multiple-key attack on m key simultaneously. To this end, we use some graph-based techniques that were introduced by Fouque, Joux, Mavromati [FJM14]. As result, we obtain an attack on m Legendre keys $k_1, \ldots, k_m$ using oracle access to $L_{k_1}(\cdot), \ldots, L_{k_m}(\cdot)$ that recovers *all m keys in total time* $\tilde{\mathcal{O}}(\sqrt{mp})$.

Our total time in turn implies that the amortized cost per Legendre key is only $\tilde{\mathcal{O}}(\sqrt{p/m})$, again beating the $\sqrt{p}$-bound.

Legendre Multiple-Key Attack with Precomputation

In the discrete logarithm setting, Corrigan-Gibbs and Kogan [CK18] showed that multiple-key attacks can be combined with precomputation, again allowing for various tradeoffs. We also transfer this combination to the Legendre key setting. This implies e.g. an attack that uses precomputation time $\tilde{\mathcal{O}}(mp^{\frac{2}{3}})$ to build a hint of size $\tilde{\mathcal{O}}(m^2p^{\frac{1}{3}})$. Upon access to $L_{k_1}(\cdot), \ldots, L_{k_m}(\cdot)$, one then computes *all m keys in total time* only $\tilde{\mathcal{O}}(p^{\frac{1}{3}})$.

Notice that in the multiple-key setting a large precomputation pays off in the sense that its cost amortizes over all keys. This explains why the multi-key precomputation setting is especially attractive for recovering Legendre keys.

Conclusion: Our attacks do not directly apply to Ethereum 2.0, since they require Legendre PRF oracle access $L_k(\cdot)$, that is typically not provided in a blockchain scenario. Nevertheless, *precomputation as well as multiple-key attacks* seem to be highly relevant in the Ethereum 2.0 context, where Legendre keys are frequently changed such that heavy key-independent precomputations pay off to optimize success probability in a (short) online key-dependent attack phase. Moreover, the more keys an attacker recovers in an online phase, the larger is his reward. Therefore, amortization of attack costs over many keys also pays off.

In the light of our novel Legendre PRF precomputation attacks one might consider — rather than the previous $p^{\frac{1}{2}}$-security level provided by Khovratovich's algorithm — a more conservative lower $p^{\frac{1}{3}}$-security level. Using this third-root bound, the 256-bit prime p used by Ethereums 2.0 still provides a high security level (of at least 85-bit), even against attacks with $L_k(\cdot)$ oracle access.

More Dlog-like Attacks and Limitations

We would like to notice that other collision-based discrete logarithm algorithms also transfer to the Legendre setting. This includes Pollard's Lambda method [Pol78] for secrets within a certain range, as well as the Esser-May method [EM20] for secrets with low Hamming weight. However, we felt that these attacks are less relevant in the Legendre key setting, in which we are not aware of any application with Legendre keys in a certain range, or with small Hamming weight.

Moreover, we would like to point out that despite the similarities between the discrete logarithm problem and the Legendre PRF, there also exists exist crucial differences that introduce technical difficulties for directly transferring discrete logarithm algorithms. Namely, the discrete logarithm setting provides us with a group structure that is heavily used in many algorithms. As an example, Corrigan-Gibbs and Kogan [CK18] compute in their multiple-key attack for some random $r_1, \ldots, r_m$ the value $h = h^{r_1} \cdot \ldots \cdot h^{r_m}$ that has discrete logarithm $r_1 k'_1 + \ldots + r_m k'_m$. Thus, in the discrete logarithm setting we easily obtain random linear combinations of the k'_i. This property greatly simplifies the analysis of Corrigan-Gibbs and Kogan's algorithm.

As opposed to the discrete logarithm, for the Legendre PRF we do not have a group structure. This implies that neither can we compute a multiple $r_1 k_1, r_1 \in \mathbb{F}_p$ using the oracle $L_{k_1}(\cdot)$, nor are we able to compute $k_1 + k_2$ using two oracles $L_{k_1}(\cdot)$ and $L_{k_2}(\cdot)$.

The missing group structure poses some additional technical problems, when we transfer in the subsequent chapters the above-mentioned discrete logarithm algorithms to the Legendre PRF setting. Nevertheless, we always succeed to design alternative algorithms that provide analogous complexity results.

Related Work and Open Problems

Our work is not the first that uses collision-finding techniques in the context of precomputation without having a group structure. E.g. Coretti, Dodis, Guo and Steinberger [CDGS18] and later Akshima, Cash, Drucker and Wee [ACDW20] designed precomputation attacks and lower bounds for salted hash functions. They showed that the salting technique is a good defense against the efficacy of precomputations for hash function collisions.

We are quite confident that the lower bounds of Corrigan-Gibbs and Kogan [CK18] from the discrete logarithm setting in generic groups also transfer to our Legendre PRF setting, when using only our limited set of allowed operations. However, we feel that such an artificially

limited *generic Legendre PRF model* would only provide misleading security guarantees. As opposed to many discrete logarithm groups, where we only have generic attacks, the Legendre PRF setting seems to offer a richer mathematical structure. E.g. the attacks of Beullens, Beyne, Udovenko, Vitto [BBUV20] and Kaluderovic, Kleinjung, Kostic [KKK20] exploit the Legendre symbol's multiplicativity to reduce the number of $L_k(\cdot)$ oracle calls. Unfortunately, we have to leave it open whether similar techniques can be applied in our setting.

Since we are purely focusing on key-recovering attacks, one might also wonder whether there exist more efficient Legendre PRF distinguishers. Given the wide analogy between discrete logarithm and Legendre PRF attacks, it is tempting to adapt e.g. the more efficient DDH-like distinguisher of Corrigan-Gibbs and Kogan [CK18] to the Legendre PRF setting. We failed to construct distinguishers with better efficiency than our key-recovery attacks, and we leave their existence as an open problem.

Our paper is structured as follows. In Section 6.2 we provide some basic definitions for properly defining collision-based random walk algorithms in the Legendre PRF setting. Our Legendre precomputation attack is given in Section 6.3. For didactic reasons, we then first generalize in Section 6.4 our precomputation attack to the multiple-key setting, since both algorithms share a similar analysis. Eventually, in Section 6.5 we provide our multiple-key attack *without* precomputation.

6.2 Legendre PRF Basics

All logarithms in this paper are base 2. Let p be prime, and let $(\frac{x}{p})$ be the Legendre symbol of x in $\mathbb{F}_p$. Since $(\frac{x}{p}) = x^{\frac{p-1}{2}} \bmod p$, the Legendre symbol can be computed in time $\mathcal{O}(\log^3 p)$, polynomial in the bit-size of p.

For ease of notation, throughout the paper we suppress all run time factors that are polynomial in $\log p$, by hiding them in soft-Oh notation, e.g. $3p\log^2 p = \tilde{\mathcal{O}}(p)$. We call any function inverse that grows faster than a polynomial in $\log p$ negligible, denoted $\mathrm{negl}(p)$. We call success probability $1 - \mathrm{negl}(p)$ overwhelming.

For a key $k \in \mathbb{F}_p$ the *original Legendre PRF* [Dam88] is defined as the function

$$\bar{L}_k : \mathbb{F}_p \to \{-1, 0, 1\},\ x \mapsto \left(\frac{x+k}{p}\right).$$

Obviously, if $y = x + k$ then $\bar{L}_0(y) = \bar{L}_k(x)$. In order to use collision-based algorithms, we would like to conclude that conversely $\bar{L}_0(y) = \bar{L}_k(x)$ implies $y = x + k$. To this end, we define a function L_k with sufficiently large range R.

Definition 14 (Legendre point)**.** Define $r = \lceil 3\log p \rceil$ and $R = \{-1, 0, 1\}^r$. We set

$$L_k : \mathbb{F}_p \to R,\ x \mapsto \left(\left(\frac{x+k}{p}\right), \ldots, \left(\frac{x+k+r-1}{p}\right)\right).$$

We denote by $L := L_0$ the *key-independent* function, and we define the set of all *Legendre points* as $P = \{L(y) \mid y \in \mathbb{F}_p\} \subset R$.

Notice that $L(x) \in \{-1, 1\}^r$ unless $x = 0$ or $x > p - r$. For simplicity, let us for a moment exclude these border cases. Under the assumption that $\bar{L}_k$ is a PRF, it is not hard to see that the r-bit range $L : \mathbb{F}_p \to \{-1, 1\}^r$ is a secure PRG (pseudorandom number generator).

In fact, Damgård suggested such a Legendre pseudorandom generator in [Dam88]. Therefore, for a random seed x the output $L(x)$ is supposed to be pseudorandom. There is strong theoretical and practical evidence [Per92, Dav33, Bac91, RS04, BBUV20, KKK20] that the distribution of $L(x)$ is even statistically close to uniform in $\{-1,1\}^r$. For simplicity of exposition, we heuristically assume such a uniform distribution. A failure of our heuristic would open the door for Legendre symbol distinguishing attacks.

Heuristic 1 (Uniformity)**.** *Let $x \in \{1,2,\ldots,p-r\}$ be chosen uniformly at random. Then $L(x)$ is uniformly distributed in $\{-1,1\}^r$. That is for all fixed $c \in \{-1,1\}^r$ we have* $\Pr[L(x) = c] = \frac{1}{2^r}$.

In the subsequent sections, we define random walks over the set $P = \{L(x) \mid x \in \mathbb{F}_p\}$ of Legendre points. Notice that P is *not* equipped with a group structure, as opposed to the discrete logarithm setting.

The following Lemma 9 implies that $|P| = p$ with overwhelming probability. This in turn implies that collisions of our random walks result in recovery of the secret Legendre PRF key k.

Lemma 9. *Let $L_k : \mathbb{F}_p \to \{-1,0,1\}^{\lceil 3\log p\rceil}$. Under Heuristic 1, with overwhelming probability all argument pairs x, y with $y \neq x + k$ satisfy $L(y) \neq L_k(x)$. Hence, with overwhelming probability*

$$L(y) = L_k(x) \;\Rightarrow\; y = x + k.$$

Proof. Let $r = \lceil 3\log p\rceil$, and let $k \in \mathbb{F}_p$ be chosen uniformly at randomly. First, consider the case of argument pairs x, y such that at exactly one of $L(y), L_k(x)$ is in $\{-1,0,1\}^r \setminus \{-1,1\}^r$. That is, either $L(y)$ or $L_k(x)$ contains a zero entry. Then obviously $L(y) \neq L_k(x)$. Second, consider the case that $L(y), L_k(x)$ both contain zeros. By Definition 14, every Legendre point can have at most one zero. Since $y \neq x+k$ the zero entries of $L(y)$, $L_k(x)$ must be in different positions, again implying $L(y) \neq L_k(x)$.

Thus, we may w.l.o.g. assume argument pairs x, y with $L(y), L_k(x) \in \{-1,1\}^r$. Since k is uniformly at random, by Heuristic 1 the Legendre point $L_k(x) = L(k + x)$ is also uniformly at random. Therefore,

$$\Pr\left[L(y) = L_k(x) \mid y \neq x + k\right] = \frac{1}{2^r} = \frac{1}{2^{\lceil 3\log p\rceil}} \leq \frac{1}{p^3}.$$

The number of pairs x, y with $y \neq x+k$ is upper-bounded by $p(p-1)$, since we exclude Legendre points with zero entries. Using Bernoulli's inequality, all these x, y satisfy $L(y) \neq L_k(x)$ with probability at least

$$\left(1 - \frac{1}{p^3}\right)^{p(p-1)} \geq 1 - \frac{p(p-1)}{p^3} \geq 1 - \frac{1}{p}.$$

□

6.3 Precomputation Attack

Let us first give a high-level description of our Legendre PRF precomputation attack, see also Figure 6.1. In a nutshell, in the precomputation phase we perform sufficiently many

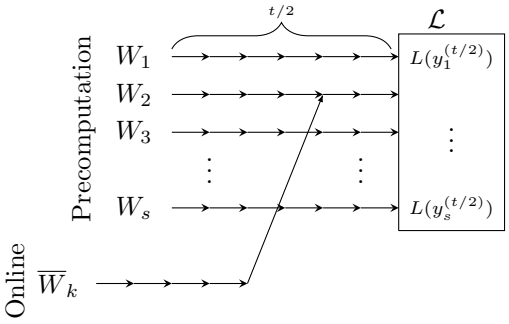


Figure 6.1: Precomputation attack

key-independent random walks $W_1, \ldots, W_s$ on the set of Legendre points P (Definition 14), where we only store the walks' endpoints. The endpoints serve as a hint for the online phase.

Upon receiving $L_k(\cdot)$ oracle access, we then compute in the online phase the Legendre key k by letting a *key-dependent* random walk $\overline{W}_k$ – defined via $L_k(\cdot)$ – collide with one of the precomputed walks. We detect the collision using our stored endpoints.

6.3.1 Random Walks – Precomputation and Online

Let $R = \{-1, 0, 1\}^{\lceil 3 \log p \rceil}$ and $P = \{L(y) \mid y \in \mathbb{F}_p\} \subset R$ (Definition 14). We define a random function $f : P \to \mathbb{F}_p$. Notice that f is compressing. In practice, f may be instantiated via some appropriate hash function. The function f helps us in a random walk W to map Legendre points $L(y)$ back to arguments y' for the Legendre PRF. We define W on P as follows.

Precomputation phase. Let $y^{(1)} \in \mathbb{F}_p$. Then W's starting point is $L(y^{(1)}) \in P$. Next, W computes $y^{(2)} = y^{(1)} + f(L(y^{(1)})) \bmod p$ and steps to its second point $L(y^{(2)}) \in P$. In general, W computes an arbitrary number of steps, where

$$y^{(i+1)} = y^{(i)} + f\left(L\left(y^{(i)}\right)\right) \bmod p \text{ for } i \geq 1 \tag{6.1}$$

with random walk points $L\left(y^{(i)}\right) \in P$. Notice that W is *key-independent*, since it does not involve oracle queries $L_k(\cdot)$. Thus, we can compute W in a *precomputation* phase solely based on the public information p. Assume that we walk W for $t/2$ steps. Then we only store the endpoint $L(y^{(t/2)})$ and its argument $y^{(t/2)}$. The endpoint $L(y^{(t/2)})$ allows us to detect collisions between walks, whereas $y^{(t/2)}$ allows us to find the Legendre key. This procedure is repeated with s different starting points $y_1^{(1)}, \ldots, y_s^{(1)}$.

Online phase. Now assume that we obtain $L_k(\cdot)$ oracle access. We want to compute in an *online* phase the secret Legendre key k. To this end we perform a *key dependent walk* $\overline{W}_k$ as follows. Choose $x^{(1)} \in_R \mathbb{F}_p$ and compute starting point $L_k(x^{(1)})$. In general, for a key dependent walk $\overline{W}_k$ we calculate the next point as

$$x^{(i+1)} = x^{(i)} + f\left(L_k\left(x^{(i)}\right)\right) \bmod p \text{ for } i \geq 1 \tag{6.2}$$

with random walk points $L_k\left(x^{(i)}\right) \in P$.

6.3.2 Colliding Walks solve Legendre.

Assume that a key-independent walk W collides with a key-dependent walk $\overline{W}_k$. I.e., there exist arguments $y^{(i)}, x^{(j)}$ with colliding points

$$L(y^{(i)}) = L_k(x^{(j)}).$$

Using Lemma 9, we immediately conclude from such a collision that

$$k = y^{(i)} - x^{(j)} \bmod p. \tag{6.3}$$

Moreover, we want to show that once two chains of points from P computed in walks $W, \overline{W}_k$ collide, they stay in the same points, i.e.

$$L(y^{(i)}) = L_k(x^{(j)}) \Rightarrow L(y^{(i+1)}) = L_k(x^{(j+1)}).$$

To this end let us assume $L(y^{(i)}) = L_k(x^{(j)})$. We already know that this implies $y^{(i)} = x^{(j)} + k \bmod p$. Using Equation (6.1), we obtain

$$\begin{aligned} y^{(i+1)} &= y^{(i)} + f\left(L\left(y^{(i)}\right)\right) \\ &= x^{(j)} + k + f\left(L\left(x^{(j)} + k\right)\right) \bmod p. \end{aligned}$$

This in turn implies

$$\begin{aligned} L\left(y^{(i+1)}\right) &= L\left(x^{(j)} + k + f\left(L\left(x^{(j)} + k\right)\right)\right) \\ &= L_k\left(x^{(j)} + f\left(L_k\left(x^{(j)}\right)\right)\right) \\ &= L_k\left(x^{(j+1)}\right). \end{aligned}$$

It remains to show that we can efficiently find arguments $y^{(i)}, x^{(j)}$ with colliding points $L(y^{(i)}), L_k\left(x^{(j)}\right)$. Since from the first colliding point on both walks stay in the same points, walk $\overline{W}_k$ eventually reaches W's endpoint. Let $L(y^{(t/2)}) = L_k(x^{(j)})$ denote this endpoint. The corresponding arguments $y^{(t/2)}, x^{(j)}$ reveal the Legendre secret key k via Equation (6.3).

The resulting precomputation attack Pre-Legendre is described in Algorithm 12. Using the parameter choice $s = t = p^{\frac{1}{3}}$ in the following Theorem 4, we achieve precomputation in time $\tilde{\mathcal{O}}(p^{\frac{2}{3}})$ using a hint of size $\tilde{\mathcal{O}}(p^{\frac{1}{3}})$, whereas the online phase runs in time $\tilde{\mathcal{O}}(p^{\frac{1}{3}})$ with constant success probability $\epsilon = \Omega(st^2/p) = \Omega(1)$.

Theorem 4. *Assume that we are given oracle access to a Legendre PRF $L_k(\cdot) : \mathbb{F}_p \to P$. Under Heuristic 1, for any $s, t \in \mathbb{N}$ with $s^2 t \leq p$ algorithm Prep-Legendre precomputes in time $\tilde{\mathcal{O}}(st)$ a hint of size $\tilde{\mathcal{O}}(s)$, which allows to find k in online time $\tilde{\mathcal{O}}(t)$ with success probability $\Omega\left(\frac{st^2}{p}\right)$.*

Proof. Let us first consider correctness and success probability. If Pre-Legendre finds a collision in line 12, then by Lemma 9 with overwhelming probability k is the correct Legendre key. It remains to show that Pre-Legendre does not output FAIL too often.

We show that the success probability $\epsilon = \Pr(\overline{\text{FAIL}})$ for finding a collision in line 12 of $\overline{W}_k$ with some precomputed walk W_ℓ's endpoint is $\Omega(st^2/p)$. Hence, we obtain constant success

Algorithm 12: Pre-Legendre

Input : $p, L_k(\cdot) : \mathbb{F}_p \to P$ with $P = \{L(y) \mid y \in \mathbb{F}_p\} \subset \{0, \pm 1\}^{\lceil 3 \log p \rceil}$
Output : $k \in \mathbb{F}_p$

begin
  Choose $s, t \in \mathbb{N}$ s.t. $st^2 \leq p$. ▷ `E.g.` $s, t = \lceil p^{\frac{1}{3}} \rceil$
  Define random $f : P \to \mathbb{F}_p$.
  for $\ell = 1, \ldots, s$ ▷ `Precomputation phase`
  do
    Choose a random $y_\ell^{(1)} \in \mathbb{F}_p$.
    Start in $L(y_\ell^{(1)})$ a $t/2$-step walk W_ℓ (Eq. 6.1)

$$y_\ell^{(i+1)} = y_\ell^{(i)} + f\left(L\left(y_\ell^{(i)}\right)\right) \bmod p$$

    with points $L\left(y_\ell^{(i)}\right) \in P$.
    Store $(L(y_\ell^{(t/2)}), y_\ell^{(t/2)})$ in a list $\mathcal{L}$ sorted by first argument.
  end
  Choose a random $x^{(1)} \in \mathbb{F}_p$. ▷ `Online phase`
  Start in $L_k(x^{(j)})$ a t-step walk $\overline{W}_k$ (Eq. 6.2)

$$x^{(j+1)} = x^{(j)} + f\left(L_k\left(x^{(j)}\right)\right) \bmod p$$

  with points $L_k\left(x^{(j)}\right) \in P$.
  if $L_k\left(x^{(j)}\right) = L(y_\ell^{(t/2)})$ *with* $\left(L(y_\ell^{(t/2)}), y_\ell^{(t/2)}\right) \in \mathcal{L}$ **then**
    return $k = y_\ell^{(t/2)} - x^{(j)} \bmod p$.
  else
    return *FAIL*.
  end
end

probability for $st^2 = \Omega(p)$, e.g. for the choice $s = t = \lceil p^{\frac{1}{3}} \rceil$. Our analysis closely follows the analysis from Corrigan-Gibbs and Kogan [CK18] for the discrete logarithm setting.

We first observe that the preprocessing walks $W_1, \ldots, W_s$ with $t/2$-steps touch at most $st/2$ Legendre points. Moreover, we show that on expectation these s walks touch at least $st/4$ distinct points.

To prove this, let X_ℓ be a random variable for the number of points touched by precomputation walk W_ℓ, $\ell = 1, \ldots, s$. Further, let $X = X_1 + \ldots + X_s \leq st/2$. We show in the following that $\Pr[X \geq st/4] \geq \frac{1}{2}$.

Using Bernoulli's inequality and $st^2 \leq p$, every $t/2$-step walk touches the maximum number $t/2$ of new point with probability at least

$$\Pr\left[X_\ell = \frac{t}{2}\right] \geq \left(\frac{p - st/2}{p}\right)^{\frac{t}{2}} = \left(1 - \frac{st}{2p}\right)^{\frac{t}{2}} \geq 1 - \frac{st^2}{4p} \geq \frac{3}{4}.$$

Therefore, every walk in the precomputation phase covers on expectation at least $\mathbb{E}[X_\ell] \geq$

$\frac{3}{4} \cdot \frac{t}{2} = \frac{3}{8}t$ new points. By linearity of expectation we have

$$\mathbb{E}[X] = \sum_{\ell=1}^{s} \mathbb{E}[X_\ell] \geq \frac{3}{8}st.$$

Using Markov's inequality and $X \leq st/2$, we obtain

$$\Pr\left[X < \frac{st}{4}\right] \leq \Pr\left[\frac{st}{2} - X \leq \frac{st}{4}\right] \leq \frac{\frac{st}{2} - \mathbb{E}[X]}{\frac{st}{4}} \leq \frac{1}{2}.$$

Therefore, $\Pr[X \geq st/4] \geq \frac{1}{2}$ as desired.

Let us assume in the following that $X \geq st/4$ Legendre points are covered during precomputation. Let E be the event that within the first $t/2$ steps of the t-step online walk $\overline{W}_k$ we hit one of the X covered points. Using $1 - x \leq e^{-x}$ and $1 - e^{-x} \geq x/2$ for $x \leq 1$, we obtain

$$\Pr[E] \geq 1 - \left(1 - \frac{st}{4p}\right)^{t/2} \geq 1 - e^{-\frac{st^2}{8p}} \geq \frac{st^2}{16p}.$$

Notice that the event E implies that in the remaining $t/2$ steps of the online phase we must hit some precomputed endpoint $L(y_\ell^{(t/2)})$ in $\mathcal{L}$. This implies success probability at least

$$\epsilon = \Pr[X \geq st/4] \cdot \Pr[E] \geq \frac{st^2}{32p}.$$

Thus, with probability ϵ we output the secret Legendre key k.

It remains to show the complexity statements. Precomputation takes time $\tilde{\mathcal{O}}(st)$ using memory $\tilde{\mathcal{O}}(s)$. The online phase runs in time $\tilde{\mathcal{O}}(t)$. □

Remark 1. *We may amplify the success probability of* Pre-Legendre *arbitrary close to* 1 *by running more key-dependent walks with different starting points, while reusing the precomputation structure. This is our strategy in the experimental Section 6.6.*

6.4 Multiple-Key Precomputation Attack

The high-level idea of our precomputation attack on multiple keys is similar to our precomputation attack on a single key from the previous Section 6.3, see also Figure 6.2.

Again, in a precomputation phase we run only key-independent walks $W_1, \ldots, W_s$, and store their endpoints in a list $\mathcal{L}$. The endpoints serve as a hint for the online phase.

Let us in the online phase attack Legendre keys $k_1, \ldots, k_m$, for which we obtain oracle access to $L_{k_\ell}(\cdot)$, $\ell = 1, \ldots, m$. Using these oracles we define key-dependent walks $\overline{W}_{k_\ell}$ that with high probability collide into some precomputed walk W_i. As in Section 6.3, collisions from $\overline{W}_{k_\ell}$ are detected via hitting some precomputed endpoint in $\mathcal{L}$. A collision of $\overline{W}_{k_\ell}$ enables us to recover k_ℓ using Equation (6.3).

The resulting procedure is given in Algorithm 13. For the choice $s = m^2 p^{\frac{1}{3}}$ and $t = p^{\frac{1}{3}}$ in Theorem 5 we obtain precomputation time $\tilde{\mathcal{O}}(mp^{\frac{2}{3}})$ and a hint of size $\tilde{\mathcal{O}}(m^2 p^{\frac{1}{3}})$, whereas the online phase finishes in time only $\tilde{\mathcal{O}}(p^{\frac{1}{3}})$ for computing all m Legendre keys with constant success probability.

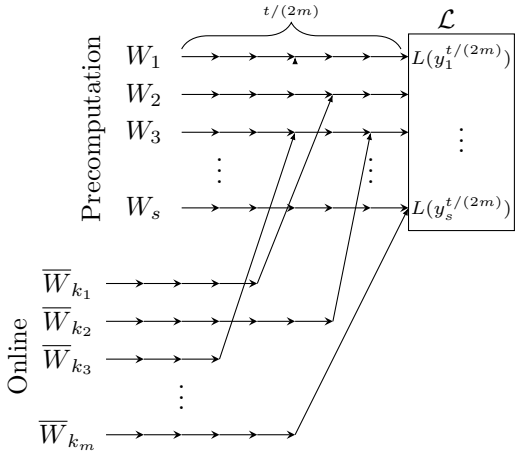


Figure 6.2: Multiple-key precomputation attack. Here $W_1, \ldots, W_s$ denote key-independent and $\overline{W}_{k_1}, \ldots, \overline{W}_{k_m}$ key-dependent random walks.

Theorem 5. *Assume that we are given oracle access to m Legendre PRFs $L_{k_1}(\cdot), \ldots, L_{k_m}(\cdot) : \mathbb{F}_p \to P$. Under Heuristic 1, for any $s, t \in \mathbb{N}$ with $st^2 \leq m^2 p$ algorithm Pre-Mult-Legendre precomputes in time $\tilde{\mathcal{O}}\left(\frac{st}{m}\right)$ a hint of size $\tilde{\mathcal{O}}(s)$, which allows to find each k_ℓ, $1 \leq \ell \leq m$ with success probability $\epsilon_\ell = \Omega\left(\frac{st^2}{m^2 p}\right)$ in total online time $\tilde{\mathcal{O}}(t)$.*

Proof. Let us first consider correctness and success probability. Here, we closely follow the analysis from the proof of Theorem 4. If we output in line 15 of Pre-Mult-Legendre a key k_ℓ then this key is correct by the discussion from Section 6.3.

It remains to show that the success probability ϵ_ℓ for recovering key k_ℓ is sufficiently large. We show that $\epsilon_\ell = \Omega(\frac{st^2}{m^2 p})$.

Let X_ℓ be a random variable for the number of new points touched by the $\frac{t}{2m}$-step precomputation walk W_ℓ, $\ell = 1, \ldots, s$. Let $X = X_1 + \ldots + X_s$. Since $X_\ell \leq \frac{t}{2m}$, we have $X \leq \frac{st}{2m}$. Using $st^2 \leq m^2 p$ and Bernoulli's inequality, every walk touches the maximal number $\frac{t}{2m}$ of new points with probability

$$\begin{aligned}\Pr\left[X_\ell = \frac{t}{2m}\right] &\geq \left(\frac{p - \frac{st}{2m}}{p}\right)^{\frac{t}{2m}} = \left(1 - \frac{st}{2pm}\right)^{\frac{t}{2m}} \\ &\geq 1 - \frac{st^2}{4m^2 p} \geq \frac{3}{4}.\end{aligned}$$

Thus, $\mathbb{E}[X_i] \geq \frac{3}{4} \cdot \frac{t}{2m} = \frac{3t}{8m}$ and $\mathbb{E}[X] \geq \frac{3st}{8m}$. Using Markov's inequality we get

$$\Pr\left[X < \frac{st}{4m}\right] \leq \Pr\left[\frac{st}{2m} - X \leq \frac{st}{4m}\right] \leq \frac{\frac{st}{2m} - \mathbb{E}[X]}{\frac{st}{4m}} \leq \frac{1}{2}.$$

This implies that with probability at least $\frac{1}{2}$ our precomputation structure covers $X \geq \frac{st}{4m}$ points. Assume in the following that $X \geq \frac{st}{4m}$. Let E_ℓ be the event that the key-dependent walk $\overline{W}_{k_\ell}$ hits within its first $\frac{t}{2m}$ steps one of the X covered point. Then

$$\Pr[E_\ell] \geq 1 - \left(1 - \frac{st}{4mp}\right)^{\frac{t}{2m}} \geq 1 - e^{-\frac{st^2}{8m^2 p}} \geq \frac{st^2}{16m^2 p}.$$

Algorithm 13: PRE-MULT-LEGENDRE

Input : $p, L_{k_1}(\cdot), \ldots, L_{k_m}(\cdot) : \mathbb{F}_p \to P$ with $P = \{L(y) \mid y \in \mathbb{F}_p\} \subset \{0, \pm 1\}^{\lceil 3 \log p \rceil}$
Output : $\{k_1, \ldots, k_m\} \in \mathbb{Z}_p^m$

begin
  Choose $s, t \in \mathbb{N}$ with $st^2 \leq m^2 p$.
  Define random $f : P \to \mathbb{F}_p$.
  for $\ell = 1, \ldots, s$ ▷ `PRECOMPUTATION phase`
  do
    Choose a random $y_\ell^{(1)} \in \mathbb{F}_p$
    Start in $L(y_\ell^{(1)})$ a $\frac{t}{2m}$-step walk W_ℓ

$$y_\ell^{(i+1)} = y_\ell^{(i)} + f\left(L\left(y_\ell^{(i)}\right)\right) \bmod p$$

    with points $L\left(y_\ell^{(i)}\right) \in P$.
    Store $(L(y_\ell^{(\frac{t}{2m})}), y_\ell^{(\frac{t}{2m})})$ in a list $\mathcal{L}$ sorted by first argument.
  end
  for $\ell = 1, \ldots, m$ ▷ `ONLINE phase`
  do
    Choose random $x_\ell^{(1)} \in \mathbb{F}_p$.
    Start in $L_{k_\ell}(x_\ell^{(1)})$ a $\frac{t}{m}$-step walk $\overline{W}_{k_\ell}$

$$x_i^{(j+1)} = x^{(j)} + f\left(L_{k_i}\left(x_i^{(j)}\right)\right) \bmod p.$$

    with points $L_{k_\ell}\left(x_\ell^{(j)}\right) \in P$.
    if $L_{k_\ell}\left(x_\ell^{(j)}\right) = L(y_{\ell'}^{(t/2)})$ *with* $\left(L(y_{\ell'}^{(\frac{t}{2m})}), y_{\ell'}^{(\frac{t}{2m})}\right) \in \mathcal{L}$ **then**
      return $k_\ell = y_{\ell'}^{(t/2)} - x_\ell^{(j)}$.
    else
      return *FAIL "*k_ℓ*"*.
    end
  end
end

In the event E_ℓ, we must hit by the discussion in section 6.3.2 in the remaining $\frac{t}{2m}$ steps of walk $\overline{W}_{k_\ell}$ a precomputed endpoint in $\mathcal{L}$. This in turn allows us to compute k_ℓ. Thus, we succeed to compute k_ℓ with probability at least

$$\epsilon_\ell = \Pr\left[X_i \geq \frac{st}{4m}\right] \cdot \Pr[E_\ell] \geq \frac{st^2}{32m^2p} = \Omega\left(\frac{st^2}{m^2p}\right).$$

It remains to show the complexity statements. The precomputation phase runs in time $\tilde{\mathcal{O}}(\frac{st}{m})$ using memory $\tilde{\mathcal{O}}(s)$. The online phase runs in time $\tilde{\mathcal{O}}(t)$. □

Remark 2. *Theorem 5 guarantees constant success probability for each key k_ℓ if $st^2 = \Omega(m^2 p)$. If Pre-Mult-Legendre fails to find some k_ℓ, we may simply rerun the key-dependent walk W_{k_ℓ} with a fresh starting point. This is our strategy in the experimental Section 6.6.*

6.5 Multiple-Key Attack (without Precomputation)

The strategy for our multiple-key attack substantially deviates from the algorithms in the previous sections. Recall that in Sections 6.3 and 6.4 we computed in the precomputation phase fixed length *key-independent* walks together with their endpoints as hint. In the online phase we then let *key-dependent* walks collide into the precomputation walks, thereby detecting collisions via endpoints.

In contrast, for our multiple-key attack *without precomputation* we solely compute *key-dependent* walks, and therefore only consider collisions between key-dependent walks, see also Figure 6.3. Our key-dependent walks are of variable length, and we stop them only if we hit some set $D \subset P$ of *distinguished* Legendre points. These distinguished points allow us to detect collisions between two walks $\overline{W}_{k_i}, \overline{W}_{k_j}$ that use Legendre keys k_i, k_j. Such a collision in turn gives us a Legendre key relation for $k_i - k_j$. Upon having collected sufficiently many of such relations, we eventually compute the Legendre keys.

The resulting algorithm recovers m Legendre keys $k_1, \ldots, k_m$ in time $\tilde{\mathcal{O}}(\sqrt{mp})$ using optimal memory $\tilde{\mathcal{O}}(m)$. Notice that we already need memory $\Omega(m)$ to store all keys. Our algorithm's complexity $\tilde{\mathcal{O}}(\sqrt{mp})$ should be compared with the naive approach that takes time $\tilde{\mathcal{O}}(m\sqrt{p})$ by running m-times Khovratovich's $\tilde{\mathcal{O}}(\sqrt{p})$ attack [Kho19].

6.5.1 High-Level idea

We run $\Theta(m)$ key-dependent walks $\overline{W}_{k_\ell}$ using m oracles $L_{k_\ell}(\cdot)$, $1 \le \ell \le m$, and as opposed to Sections 6.3 and 6.4 *no* key-independent walk. These walks give us $\Omega(m)$ mutual collisions. Let us assume that two walks $\overline{W}_{k_\ell}, \overline{W}_{k_{\ell'}}$ with different keys $k_i \neq k_j$ collide. Then there exist $x_\ell^{(u)}, x_{\ell'}^{(v)}$ such that

$$L_{k_i}\left(x_\ell^{(u)}\right) = L_{k_j}\left(x_{\ell'}^{(v)}\right) = L\left(x_{\ell'}^{(v)} + k_j\right).$$

Using Lemma 9, we conclude that

$$k_i - k_j = x_{\ell'}^{(v)} - x_\ell^{(u)} \bmod p. \tag{6.4}$$

Thus, every collision among two walks defines a *relation* as in Equation (6.4) between two keys k_i, k_j.

Let us define an undirected graph $G = (V, E)$ with $|V| = m$ and an initially empty edge set E. Every relation as in Equation (6.4) adds an edge $\{i, j\}$ to E with label $\ell_{ij} = x_{\ell'}^{(v)} - x_\ell^{(u)} \in \mathbb{F}_p$. Assume that we collected sufficiently many edges such that G gets connected, i.e. G contains a spanning tree T. Take an arbitrary vertex $i \in T$ corresponding to key k_i. Compute k_i in time $\tilde{\mathcal{O}}(\sqrt{p})$ using Khovratovich's algorithm. Now, traverse T starting in vertex i. Let $\{i, j\}$ be the first traversal edge with label ℓ_{ij}. Using the relation from Equation (6.4), we conclude that $k_j = k_i - \ell_{ij}$. Thus, by traversing T we recover all m keys.

In our algorithm, we will not wait until G gets fully connected, since by the results of Erdös, Renyi [ER60] this requires $\Omega(m \log m)$ relations. Instead, we use from Erdös, Renyi [ER60] that after $\Omega(m)$ relations, G has a so-called *giant component* $V_G \subseteq V$, a connected set of vertices that contains all but a (small) constant fraction of V. We compute a spanning tree T

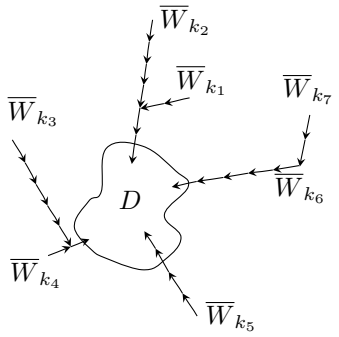


Figure 6.3: Multiple-key attack. D denotes a distinguished point set and $\overline{W}_{k_1}, \ldots, \overline{W}_{k_7}$ are key-dependent random walks.

in this giant component V_G, and recover by the above algorithm all keys within V_G. We then remove the known keys, and recursively run our algorithm on the remaining keys.

6.5.2 How to detect collisions

Recall that we have to address a technical collision-detection issue, since as opposed to our algorithms from Sections 6.3 and 6.4 we do no longer collide into some precomputed structure. Instead, our online walks mutually collide. In order to detect these collisions, we use the van Oorschot-Wiener *distinguished point* strategy [vW99].

Let $g = (g_1, \ldots, g_r) \in P$ be a Legendre point. Fix a random $d \in \{0,1\}^k$ with $k \leq r$. Then we call g *distinguished* if

$$(g_1, \ldots, g_k) = d,$$

i.e., the Legendre point g starts on its first k coordinates with d. Under Heuristic 1, any random Legendre point is distinguished with probability $q = 2^{-k}$.

Now, we run every walk $\overline{W}_{k_\ell}$ until it hits a distinguished point g, see Figure 6.3. We then store this distinguished points g in a sorted list $\mathcal{L}$. In our algorithm we choose q (and therefore $k \approx \log(1/q)$) such that with good probability our walks have the desired lengths. Assume now that two walks $\overline{W}_{k_i}, \overline{W}_{k_j}$ with different keys $k_i \neq k_j$ collide. Then there exist $x_\ell^{(u)}, x_{\ell'}^{(v)}$ such that

$$L_{k_i}\left(x_\ell^{(u)}\right) = L_{k_j}\left(x_{\ell'}^{(v)}\right),$$

from which we conclude via Equation (6.4) that

$$x_\ell^{(u)} = x_{\ell'}^{(v)} + k_j - k_i.$$

Analogous to Section 6.3 we show that once $\overline{W}_{k_i}, \overline{W}_{k_j}$ collide, they stay in the same Legendre points, i.e.

$$L_{k_i}\left(x_\ell^{(u+1)}\right) = L_{k_j}\left(x_{\ell'}^{(v+1)}\right).$$

This follows from

$$\begin{aligned}
&L_{k_i}\left(x_\ell^{(u+1)}\right) = L_{k_i}\left(x_\ell^{(u)} + f\left(L_{k_i}\left(x_\ell^{(u)}\right)\right)\right)\\
&= L_{k_i}\left(x_{\ell'}^{(v)} + k_j - k_i + f\left(L_{k_i}\left(x_{\ell'}^{(v)} + k_j - k_i\right)\right)\right)\\
&= L\left(x_{\ell'}^{(v)} + k_j + f\left(L\left(x_{\ell'}^{(v)} + k_j\right)\right)\right)\\
&= L_{k_j}\left(x_{\ell'}^{(v)} + f\left(L_{k_j}\left(x_{\ell'}^{(v)}\right)\right)\right)\\
&= L_{k_j}\left(x_{\ell'}^{(v+1)}\right).
\end{aligned}$$

Hence, $\overline{W}_{k_i}, \overline{W}_{k_j}$ must eventually hit the same distinguished point $g = L_{k_i}(x_\ell^{(u+c)}) = L_{k_j}(x_{\ell'}^{(v+c)}) \in \mathcal{L}$ for some $c \geq 0$. This allows us to derive a relation as in Equation (6.4).

This ends the high-level description of our algorithm. The resulting procedure Mult-Legendre is described in Algorithm 14.

Theorem 6. *Assume that we are given oracle access to m Legendre PRFs $L_{k_1}(\cdot), \ldots, L_{k_m}(\cdot) : \mathbb{F}_p \to P$. Under Heuristic 1, algorithm Mult-Legendre finds all $k_1, \ldots, k_m$ in total time $\tilde{\mathcal{O}}(\sqrt{mp})$ with overwhelming success probability.*

Proof. The correctness of Mult-Legendre follows from the discussion above. In the following we show that in a single run of Mult-Legendre we obtain with overwhelming probability $1 - \mathrm{negl}(m)$ at least 98% of all keys.

In a nutshell, we first prove that with overwhelming probability $3m$ out of our $4m$ walks $\overline{W}_{k_\ell,i}$ perform at least $\frac{t}{m}$ steps. From this we conclude that we obtain at least $2m$ key relations as in Equation (6.4), which in turn gives as a giant component in G that allows us to recover at least 98% of our Legendre keys.

Let us first prove that at least a $\frac{3}{4}$-fraction of our random walks have length at least $\frac{t}{m}$. Notice that all walks either hit a distinguished point, or run into a self loop. We detect potential self loop in line 11, hence we may assume w.l.o.g. that all walks end in a distinguished point.

Let $X_{\ell,i}$ be an indicator variable that takes value 1 iff walk $\overline{W}_{k_\ell,i}$ has length at least $\frac{t}{m}$. We hit a distinguished point with probability $2^{-k} \leq q$. This implies

$$\Pr[X_{\ell,i} = 1] \geq (1-q)^{\frac{t}{m}} = \left(1 - \frac{m}{5t}\right)^{\frac{t}{m}} \geq 1 - \frac{1}{5} = \frac{4}{5}.$$

Let $X = \sum_{\ell=1}^{m} \sum_{i=1}^{4} X_{\ell,i}$. Then the expected number of walks with length at least $\frac{t}{m}$ is at least $\mathbb{E}[X] \geq \frac{16}{5}m$. Let $\mu = \frac{16}{5}m$. Using the Chernoff bound $\Pr[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$, we obtain

$$\Pr[X \leq 3m] \leq e^{-\frac{m}{160}}.$$

Hence, with overwhelming probability $1 - \mathrm{negl}(m)$ we obtain at least $3m + 1$ walks of length at least $\frac{t}{m}$. In the following analysis, we consider those online walks with minimum length $\frac{t}{m}$. We define an indicator variable $Y_{\ell,\ell'}$ that takes value 1 iff walks $\overline{W}_{k_\ell,i}$ and $\overline{W}_{k_{\ell'},i'}$ for any $i, i' \in \{1,2,3,4\}$ collide. Using $t^2 \geq mp$, we obtain

$$\begin{aligned}
\Pr[Y_{\ell,\ell'} = 1] &\geq 1 - \left(\frac{p - \frac{t}{m}}{p}\right)^{\frac{t}{m}} = 1 - \left(1 - \frac{t}{mp}\right)^{\frac{t}{m}}\\
&\geq 1 - e^{-\frac{t^2}{m^2 p}} \geq \frac{t^2}{2m^2 p} \geq \frac{1}{2m}.
\end{aligned}$$

Algorithm 14: MULT-LEGENDRE

Input : $p, L_{k_1}(\cdot), \ldots, L_{k_m}(\cdot) : \mathbb{F}_p \to P$ with $P = \{L(y) \mid y \in \mathbb{F}_p\} \subset \{0, \pm 1\}^{\lceil 3 \log p \rceil}$
Output : $\{k_1, \ldots, k_m\} \in \mathbb{Z}_p^m$

begin
  Choose $t = \lceil \sqrt{mp} \rceil$.
  Define random $f : P \to \mathbb{F}_p$.
  Set $q = \frac{m}{5t}$ and $k = \lceil \log(1/q) \rceil$.
  Choose a random $d \in \{0,1\}^k$.
  Let $D = d \times \{0,1\}^{r-k} \subset P$.
  for $\ell = 1, \ldots, m$ **do**
    for $i = 1, \ldots, 4$ **do**
      Choose a random $x_\ell^{(1)} \in \mathbb{F}_p$.
      Run from $L_{k_\ell}(x_\ell^{(1)})$ a random walk $\overline{W}_{k_\ell, i}$

$$x_\ell^{(j+1)} = x_\ell^{(j)} + f\left(L_{k_\ell}\left(x_\ell^{(j)}\right)\right) \bmod p,$$

      until $\overline{W}_{k_\ell,i}$ hits a distinguished point

$$g_\ell = L_{k_\ell}\left(x_\ell^{(j_\ell)}\right) \in D.$$

      if $\overline{W}_{k_\ell,i}$ *takes more than* $8\frac{t}{m}$ *steps* **then**
        go back to step 9. ▷ `detect loop`
      Store $\left(g_\ell, \ell, x_\ell^{(j_\ell)}\right)$ in list $\mathcal{L}$.
    end
  end
  Sort $\mathcal{L}$ by its 1^{st} entry.
  Define an undirected graph $G = (\{1, \ldots m\}, \emptyset)$.
  for *every* $\left(g_\ell, \ell, x_\ell^{(j_\ell)}\right) \neq \left(g_\ell, \ell', x_{\ell'}^{(j_{\ell'})}\right) \in \mathcal{L}$ **do**
    if $\ell \neq \ell'$ **then**
      Include in E edge $\{\ell, \ell'\}$ with label $x_{\ell'}^{(j_{\ell'})} - x_\ell^{(j_\ell)}$.
  end
  Compute G's largest connected component $V_G = \{v_1, \ldots v_b\}$ and a spanning tree T of V_G.
  Compute k_{v_1} with Khovratovich's algorithm.
  return $k_{v_1}, \ldots, k_{v_b}$ by traversing T.
  Recursively call MULT-LEGENDRE with the remaining keys $\{k_1, \ldots, k_m\} \setminus \{k_{v_1}, \ldots, k_{v_b}\}$.
end

Note that $Y_{\ell,\ell'} = 1$ iff walks $\overline{W}_{k_\ell,i}, \overline{W}_{k_{\ell'},i'}$ give us a relation on two Legendre keys. Thus, in total we obtain at least $Y := \sum_{1 \le \ell < \ell' \le 3m+1} Y_{\ell,\ell'}$ key relations. Therefore, the expected number of relations is at least

$$\mathbb{E}[Y] \ge \binom{3m+1}{2} \cdot \frac{1}{2m} \ge \frac{9}{4}m.$$

Notice that a $\frac{1}{m}$-fraction of the relations are useless, since they do not satisfy condition $\ell \neq \ell'$ in line 18, i.e. they involve the same key. Moreover, Mult-Legendre may produce the same edge $\{\ell, \ell'\}$ several times. However, it is easy to see that the expected number of these duplicates is constant. After subtracting these (in total constant many) useless relations, we conclude via another Chernoff bound argument that $Y \geq 2m$ with overwhelming probability.

From the results of Erdös and Renyi [ER60] we know that graphs with m vertices and $cm := 2m > \frac{m}{2}$ randomly chosen edges contain with overwhelming probability a connected component of size at least

$$\left(1 - \frac{1}{2c} \sum_{k=1}^{\infty} \frac{k^{k-1}}{k!} (2ce^{-2c})^k\right) m > 0.98m.$$

This eventually enables us to recover at least 98% of all Legendre keys in a single run of Mult-Legendre.

It remains to show the running time for recovering all keys $k_1, \ldots, k_m$. Let us consider the two **for**-loops that construct $4m$ walks. We repeat the inner loop, whenever we encounter a walk that takes more than $8\frac{t}{m}$ steps. This happens with probability at most

$$(1-q)^{\frac{8t}{m}} \leq e^{-\frac{8}{5}} \approx 0.2.$$

Thus, in each iteration of our **for**-loops with probability at least $\frac{4}{5}$ a walk hits a distinguished point within $8\frac{t}{m}$ steps and stops. Let Y be a random variable for the number of iterations in both **for**-loops. Then $\mathbb{E}[Y] \leq 4m \cdot \frac{5}{4} = 5m$. By Markov's inequality

$$\Pr[Y \geq 10m] \leq \frac{5m}{10m} = \frac{1}{2}.$$

Hence, with probability at least $\frac{1}{2}$ our walk construction is completed by running at most $10m$ iterations of length at most $8\frac{t}{m}$. This takes time at most $\tilde{\mathcal{O}}(t)$.

On walk completion, our list $\mathcal{L}$ contains $4m$ entries. Thus, $G = (V, E)$ can be constructed in time $\tilde{\mathcal{O}}(m)$. We run Depth First Search (DFS) on G to compute a DFS tree T of its giant component in time $\mathcal{O}(|V| + |E|) = \mathcal{O}(m)$.

Khovratovich's algorithm in line 22 runs in time $\tilde{\mathcal{O}}(\sqrt{p})$, and a traversal of T can be done in time $\mathcal{O}(m)$. Since $m \leq p$, we have

$$m = \sqrt{m} \cdot \sqrt{m} \leq \sqrt{mp} < t.$$

Thus, we obtain total run time $\tilde{\mathcal{O}}(t + m) = \tilde{\mathcal{O}}(t) = \tilde{\mathcal{O}}(\sqrt{mp})$ of one iteration of Mult-Legendre. As shown before, every iteration of Mult-Legendre recovers at least a 98%-fraction of Legendre keys. Hence, we recover all Legendre keys in time

□

6.6 Experiments

Since our algorithms for computing the Legendre symbol involve the (natural) Heuristic 1 concerning uniformity of Legendre points, we check the validity of Heuristic 1 and therefore the statements of Theorems 4 to 6 experimentally. To this end, we implemented all three Legendre key algorithms Pre-Legendre, Pre-Mult-Legendre and Mult-Legendre and analyzed the number of random walk steps as a function of the field size p.

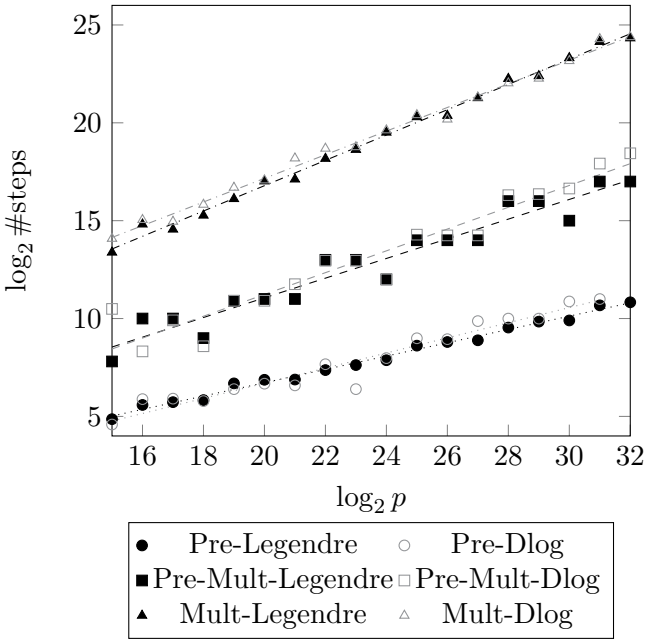


Figure 6.4: Number of random walk steps as a function of p on a double logarithmic scale. Comparison of our algorithm Pre-Legendre, Pre-Mult-Legendre ($m = p^{1/6}$) and Mult-Legendre ($m = p^{1/3}$) to analogous discrete logarithm algorithms. Each data point is averaged over 10 samples, corresponding data points are interpolated by lines.

Let g^{k_ℓ} be a discrete logarithm problem. For better comparison with our Legendre PRF algorithms, we also implemented discrete logarithm algorithms in the same field $\mathbb{F}_p$ using analogous key-independent and key-dependent walks

$$x^{(i+1)} = g^{x^{(i)}} \bmod p, \text{ respectively } x^{(i+1)} = g^{k_\ell} \cdot g^{x^{(i)}} \bmod p.$$

All benchmarks were performed on an Intel i7-8550U CPU @ 1.80GHz. Each data point represents the average over 10 samples. Our code is publically available at `https://github.com/FloydZ/prep-legendre`.

The results are depicted in Figure 6.4. We see that our Legendre algorithms require as many random walk steps as the corresponding discrete logarithm implementations. This is what we expect under Heuristic 1. We therefore believe that precise estimations for the number of random walk steps – including the constant hidden in the $\mathcal{O}$-notation, as extensively analyzed in the discrete logarithm literature (see Table 3 in [GWZ17] for an overview) – directly translate to the Legendre PRF setting.

We experimentally validated the run time statements of our Theorems 4 to 6 as follows.

Pre-Legendre Using the parameter choice $s = t = p^{\frac{1}{3}}$, we precomputed a hint of size $s = p^{\frac{1}{3}}$. In Figure 6.4, we depict the corresponding number of random walk steps in the online phase on a logarithmic scale. By Theorem 4, we require online at most $t = p^{1/3}$ random walk steps to recover the Legendre key with constant success probability. In our experiments, we repeated an unsuccessful t-step random online walk with a different starting point, until we eventually found the Legendre key, see also Remark 1. Thus, we always succeeded at the cost of a slightly increased running time. The interpolation line through our data points has a slope of 0.34, showing that thereby we do not significantly sacrifice run time.

Pre-Mult-Legendre We chose to attack $m = p^{\frac{1}{6}}$ with the parameter choice $s = p^{\frac{1}{3}}$ and $t = p^{\frac{1}{2}}$. That is, we again precomputed a hint of size $s = p^{\frac{1}{3}}$. By Theorem 5, we require in total only $t = p^{\frac{1}{2}}$ random walk steps to recover each key with constant success probability. As in the experiments for Pre-Legendre, we also repeated in the multiple-key setting unsuccessful walks with different starting points, until we eventually recovered the desired key, see Remark 2. Thus, our data points in Figure 6.4 reflect the total online time to recover all $m = p^{\frac{1}{6}}$ keys. The slope of the interpolation line is 0.52, again showing that we only marginally sacrifice run time to recover all keys.

Mult-Legendre In the multiple-key setting without precomputation, we chose to attack $m = p^{\frac{1}{3}}$ keys. By the proof Theorem 6, a single run of Mult-Legendre gives us at least $2m$ Legendre key relations, from which we can recover those keys that lie in the giant component of the graph G. Experimentally, we recover on average $3.46m$ relations. The collection step of our key relations is supposed to finish in time $\tilde{\mathcal{O}}(\sqrt{mp}) = \tilde{\mathcal{O}}(p^{\frac{2}{3}})$, which is validated by the slope 0.64 of our interpolation line.

Bibliography

[AB21] Martin Albrecht and Gregory Bard. *The M4RI Library*. The M4RI Team, 2021.

[ABB+17] Nicolas Aragon, Paulo SLM Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. Bike: bit flipping key encapsulation. 2017.

[ACDW20] Akshima, David Cash, Andrew Drucker, and Hoeteck Wee. Time-space tradeoffs and short collisions in merkle-damgård hash functions. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 157–186. Springer, Cham, August 2020.

[ADH+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746. Springer, Cham, May 2019.

[AFG+23] Carlos Aguilar-Melchor, Thibauld Feneuil, Nicolas Gama, Shay Gueron, James Howe, David Joseph, Antoine Joux, Edoardo Persichetti, Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue. SDitH — Syndrome Decoding in the Head. Technical report, National Institute of Standards and Technology, 2023. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`.

[AGLL94] Derek Atkins, Michael Graff, Arjen Lenstra, and Paul Leyland. Rsa-129 challenge, 1994. `https://www.ontko.com/pub/rayo/primes/hr_rsa.txt`.

[AKKM13] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määttä. Space-time tradeoffs for subset sum: An improved worst case algorithm. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP 2013, Part I*, volume 7965 of *LNCS*, pages 45–56. Springer, Berlin, Heidelberg, July 2013.

[ALL19] Nicolas Aragon, Julien Lavauzelle, and Matthieu Lequesne. decodingchallenge.org, 2019.

[Alm19] Josh Alman. An illuminating algorithm for the light bulb problem. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, volume 69 of *OASICS*, pages 2:1–2:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[ANSS18] Yoshinori Aono, Phong Q Nguyen, Takenobu Seito, and Junji Shikata. Lower bounds on lattice enumeration with extreme pruning. In *Annual International Cryptology Conference*, pages 608–637. Springer, 2018.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801, 2015.

[Bac91] Eric Bach. Realistic analysis of some randomized algorithms. *Journal of Computer and System Sciences*, 42(1):30–53, 1991.

[Bar07] Gregory V Bard. *Algorithms for solving linear and polynomial systems of equations over finite fields, with applications to cryptanalysis*. University of Maryland, College Park, 2007.

[BBC+19] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. A finite regime analysis of information set decoding algorithms. *Algorithms*, 12(10):209, 2019.

[BBC+20] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray A. Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier A. Verbel. Improvements of algebraic attacks for solving the rank decoding and MinRank problems. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 507–536. Springer, Cham, December 2020.

[BBFK05] F. Bahr, M. Boehm, Jens Franke, and Thorsten Kleinjung. Rsa-193 challenge, 2005. `https://mathworld.wolfram.com/news/2005-11-08/rsa-640/`.

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 633–666. Springer, Cham, December 2020.

[BBUV20] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. Cryptanalysis of the Legendre PRF and generalizations. *IACR Trans. Symm. Cryptol.*, 2020(1):313–330, 2020.

[BCDL19] Rémi Bricout, André Chailloux, Thomas Debris-Alazard, and Matthieu Lequesne. Ternary syndrome decoding with large weight. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 437–466. Springer, Cham, August 2019.

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 364–385. Springer, Berlin, Heidelberg, May 2011.

[BCL+17] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic mceliece. 2017.

[BD20] Ward Beullens and Cyprien Delpech de Saint Guilhem. LegRoast: Efficient post-quantum signatures from the Legendre PRF. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 130–150. Springer, Cham, 2020.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016.

[BDK+18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: a cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[Ben80] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[Ber10] Daniel J Bernstein. Grover vs. mceliece. In *Post-Quantum Cryptography: Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings 3*, pages 73–80. Springer, 2010.

[BJMM12] 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 David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, Berlin, Heidelberg, April 2012.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, pages 435–440. ACM Press, May 2000.

[BL06] Daniel J Bernstein and Arjen K Lenstra. A general number field sieve implementation. In *The development of the number field sieve*, pages 103–126. Springer, 2006.

[BL12] Daniel J. Bernstein and Tanja Lange. Computing small discrete logarithms faster. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 317–338. Springer, Berlin, Heidelberg, December 2012.

[BLP08] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In Johannes Buchmann and Jintai Ding, editors, *Post-quantum cryptography, second international workshop, PQCRYPTO 2008*, pages 31–46. Springer, Berlin, Heidelberg, October 2008.

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 743–760. Springer, Berlin, Heidelberg, August 2011.

[BM17] Leif Both and Alexander May. Optimizing bjmm with nearest neighbors: full decoding in 22/21n and mceliece security. In *WCC workshop on coding and cryptography*, volume 214, 2017.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 25–46. Springer, Cham, 2018.

[BMPS20] Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. LESS is more: Code-based signatures without syndromes. In Abderrahmane Nitaj and Amr M. Youssef, editors, *AFRICACRYPT 20*, volume 12174 of *LNCS*, pages 45–65. Springer, Cham, July 2020.

[BMVT78] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[BTZ12] Shi Bai, Emmanuel Thomé, and Paul Zimmermann. Rsa-212 challenge, 2012.

[CC94] Anne Canteaut and Hervé Chabanne. *A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem.* PhD thesis, INRIA, 1994.

[CC98] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to mceliece's cryptosystem and to narrow-sense bch codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.

[CCU+20] Tung Chou, Carlos Cid, Simula UiB, Jan Gilcher, Tanja Lange, Varun Maram, Rafael Misoczki, Ruben Niederhagen, Kenneth G Paterson, Edoardo Persichetti, et al. Classic McEliece: conservative code-based cryptography 10 october 2020. 2020.

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John Steinberger. Random oracles and non-uniformity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 227–258. Springer, 2018.

[CDMT22] Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. Statistical decoding 2.0: Reducing decoding to LPN. In Shweta Agrawal and Dongdai Lin, editors, *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*, volume 13794 of *Lecture Notes in Computer Science*, pages 477–507. Springer, 2022.

[CG90] John T Coffey and Rodney M Goodman. The complexity of information set decoding. *IEEE Transactions on Information Theory*, 36(5):1031–1037, 1990.

[CGF91] John T Coffey, Rodney M Goodman, and Patrick G Farrell. New approaches to reduced-complexity decoding. *Discrete Applied Mathematics*, 33(1-3):43–60, 1991.

[Cha95] Florent Chabaud. On the security of some cryptosystems based on error-correcting codes. In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 131–139. Springer, Berlin, Heidelberg, May 1995.

[CK18] Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 415–447. Springer, Cham, April / May 2018.

[CLOS91] Matthijs J. Coster, Brian A. LaMacchia, Andrew M. Odlyzko, and Claus P. Schnorr. An improved low-density subset sum algorithm. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 54–67. Springer, Berlin, Heidelberg, April 1991.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[CS98] Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 187–199. Springer, Berlin, Heidelberg, October 1998.

[CTS16] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, pages 144–161. Springer, 2016.

[Dam88] Ivan Damgård. On the randomness of legendre and jacobi sequences. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 1988.

[Dav33] H. Davenport. On the distribution of quadratic residues (mod p). *Journal of the London Mathematical Society*, s1-8(1):46–52, 1933.

[DDKS12] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 719–740. Springer, Berlin, Heidelberg, August 2012.

[DEM19] Claire Delaplace, Andre Esser, and Alexander May. Improved low-memory subset sum and LPN algorithms via multiple collisions. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *LNCS*, pages 178–199. Springer, Cham, December 2019.

[Din18] Itai Dinur. An algorithmic framework for the generalized birthday problem. *Designs, Codes and Cryptography*, pages 1–30, 2018.

[Din21] Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 374–403. Springer, Cham, October 2021.

[DMEH+96] Bruce Dodson, Peter Montgomery, Marije Elkenbracht-Huizing, Arjen K. Lenstra, Matt Fante, Paul Leyland, Damian Weber, and Joerg Zayer. Rsa-130 challenge, 1996. `https://www.ontko.com/pub/rayo/primes/hr_rsa.txt`.

[DST19] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave: A new family of trapdoor one-way preimage sampleable functions based on codes.

In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 21–51. Springer, Cham, December 2019.

[DSv21] Léo Ducas, Marc Stevens, and Wessel P. J. van Woerden. Advanced lattice sieving on GPUs, with tensor cores. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 249–279. Springer, Cham, October 2021.

[Dub10] Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, 2010.

[Dum89] Il'ya Isaakovich Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25(1):24–32, 1989.

[Dum91] II Dumer. On constructing optimal tests and defect-correcting codes. In *Proceedings. 1991 IEEE International Symposium on Information Theory*, pages 313–313. IEEE, 1991.

[EB22] Andre Esser and Emanuele Bellini. Syndrome decoding estimator. In *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography*, volume 13177 of *Lecture Notes in Computer Science*, pages 112–141. Springer, 2022.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 486–514. Springer, Cham, August 2017.

[EM19] Andre Esser and Alexander May. Better sample–random subset sum in $2^{0.255n}$ and its impact on decoding random linear codes. *arXiv preprint arXiv:1907.04295, withdrawn*, 2019.

[EM20] Andre Esser and Alexander May. Low weight discrete logarithm and subset sum in $2^{0.65n}$ with polynomial memory. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 94–122. Springer, Cham, May 2020.

[EMZ22] Andre Esser, Alexander May, and Floyd Zweydinger. McEliece needs a break - solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 433–457. Springer, Cham, May / June 2022.

[ER60] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[ERCB+21] Andre Esser, Sergi Ramos-Calderer, Emanuele Bellini, José I Latorre, and Marc Manzano. An optimized quantum implementation of isd on scalable quantum resources. *arXiv preprint arXiv:2112.06157*, 2021.

[ERCB+22] Andre Esser, Sergi Ramos-Calderer, Emanuele Bellini, José I Latorre, and Marc Manzano. Hybrid decoding–classical-quantum trade-offs for information set decoding. In *Post-Quantum Cryptography: 13th International Workshop, PQCrypto 2022, Virtual Event, September 28–30, 2022, Proceedings*, pages 3–23. Springer, 2022.

[Ess20] Andre Esser. *Memory-efficient algorithms for solving subset sum and related problems with cryptanalytic applications.* PhD thesis, Ruhr University Bochum, Germany, 2020.

[Ess22] Andre Esser. Revisiting nearest-neighbor-based information set decoding. Cryptology ePrint Archive, Paper 2022/1328, 2022. `https://eprint.iacr.org/2022/1328`.

[ESST99] A Escott, J Sager, A Selkirk, and Dimitrios Tsapakidis. Attacking elliptic curve cryptosystems using the parallel pollard rho method. *CryptoBytes–The Technical Newsletter of RSA Laboratories*, 4(2):15–19, 1999.

[EWF12] Mani Malek Esmaeili, Rabab Kreidieh Ward, and Mehrdad Fatourechi. A fast approximate nearest neighbor search algorithm in the hamming space. *IEEE transactions on pattern analysis and machine intelligence*, 34(12):2481–2488, 2012.

[EZ23] Andre Esser and Floyd Zweydinger. New time-memory trade-offs for subset sum: Improving ISD in theory and practice. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 360–390. Springer, Cham, April 2023.

[FJM14] Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to discrete logarithm, Even-Mansour and PRINCE. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 420–438. Springer, Berlin, Heidelberg, December 2014.

[Fou20a] Ethereum Foundation. Ethereum 2.0, 2020. `https://ethereum.org/en/eth2/`.

[Fou20b] Ethereum Foundation. Ethereum 2.0, 2020. `https://github.com/ethereum/eth2.0-specs`.

[FS09] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 88–105. Springer, Berlin, Heidelberg, December 2009.

[GKH17] Cheikh Thiécoumba Gueye, Jean Belo Klamti, and Shoichi Hirose. Generalization of bjmm-isd using may-ozerov nearest neighbor algorithm over an arbitrary finite field $\mathbb{F}_q$. In *International Conference on Codes, Cryptology, and Information Security*, pages 96–109. Springer, 2017.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 430–443. ACM Press, October 2016.

[GWZ17] Steven D Galbraith, Ping Wang, and Fangguo Zhang. Computing elliptic curve discrete logarithms with improved baby-step giant-step algorithm. *Advances in Mathematics of Communications*, 11(3):453, 2017.

[Hir16] Shoichi Hirose. May-ozerov algorithm for nearest-neighbor problem over $\mathbb{F}_q$ and its application to information set decoding. In *International Conference for Information Technology and Communications*, pages 115–126. Springer, 2016.

[HJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 235–256. Springer, Berlin, Heidelberg, May / June 2010.

[Hob12] David Hobach. Practical analysis of information set decoding algorithms. *Masterarbeit*, 2012.

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.

[HS13] Yann Hamdaoui and Nicolas Sendrier. A non asymptotic analysis of information set decoding. *Cryptology ePrint Archive*, 2013.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM Journal on Computing*, 39(3):1121–1152, 2009.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th ACM STOC*, pages 604–613. ACM Press, May 1998.

[JS91] Antoine Joux and Jacques Stern. Improving the critical density of the lagarias-odlyzko attack against subset sum problems. In *Fundamentals of Computation Theory: 8th International Conference, FCT'91 Gosen, Germany, September 9–13, 1991 Proceedings 8*, pages 258–264. Springer, 1991.

[KAF+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter L Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit rsa modulus. In *Annual Cryptology Conference*, pages 333–350. Springer, 2010.

[KAF+20] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Rsa-250 challenge, 2020.

[Kho19] Dmitry Khovratovich. Key recovery attacks on the Legendre PRFs within the birthday bound. Cryptology ePrint Archive, Report 2019/862, 2019. `https://eprint.iacr.org/2019/862`.

[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*, pages 507–527. Springer, 2018.

[KKK16] Matti Karppa, Petteri Kaski, and Jukka Kohonen. A faster subquadratic algorithm for finding outlier correlations. In Robert Krauthgamer, editor, *27th SODA*, pages 1288–1305. ACM-SIAM, January 2016.

[KKK20] Novak Kaluderovic, Thorsten Kleinjung, and Dusan Kostic. Improved key recovery on the legendre PRF, 2020.

[KL22] Pierre Karpman and Charlotte Lefevre. Time-memory tradeoffs for large-weight syndrome decoding in ternary codes. In *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography*, volume 13177 of *Lecture Notes in Computer Science*, pages 82–111. Springer, 2022.

[KM95] Samir Khuller and Yossi Matias. A simple randomized sieve algorithm for the closest-pair problem. *Information and Computation*, 118(1):34–37, 1995.

[Knu98] Donald E Knuth. *The art of computer programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998.

[Knu11] Donald E Knuth. *The Art of Computer Programming: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 2011.

[Kru89] Evgenii Avramovich Kruk. Decoding complexity bound for linear block codes. *Problemy Peredachi Informatsii*, 25(3):103–107, 1989.

[KS01] Fabian Kuhn and René Struik. Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms. In Serge Vaudenay and Amr M. Youssef, editors, *SAC 2001*, volume 2259 of *LNCS*, pages 212–229. Springer, Berlin, Heidelberg, August 2001.

[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*, pages 69–89. Springer, 2017.

[Lab91] RSA Laboratories. Rsa factoring challenge, 1991. `https://groups.google.com/g/sci.crypt/c/AA7M9qWWx3w/m/EkrsR69CDqIJ`.

[Lan] Gregory Landais. Dumer implementation. `https://github.com/cr-marcstevens/inria-collision-dec`. Accessed: 2023-01-15.

[Lan12] Grégory Landais. *Code of Grégory Landais*, 2012.

[LB88] Pil Joong Lee and Ernest F Brickell. An observation on the security of mceliece's public-key cryptosystem. In *Advances in Cryptology—EUROCRYPT'88: Workshop on the Theory and Application of Cryptographic Techniques Davos, Switzerland, May 25–27, 1988 Proceedings 7*, pages 275–280. Springer, 1988.

[LCH11] Hyung Tae Lee, Jung Hee Cheon, and Jin Hong. Accelerating ID-based encryption based on trapdoor DL using pre-computation. Cryptology ePrint Archive, Report 2011/187, 2011. `https://eprint.iacr.org/2011/187`.

[Leo88] Jeffrey S Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.

[LL93] A.K. Lenstra and H.W.J. Lenstra. *The Development of the Number Field Sieve*. Lecture Notes in Mathematics. Springer, 1993.

[LLMP93] Arjen K Lenstra, Hendrik W Lenstra, Mark S Manasse, and John M Pollard. The factorization of the ninth fermat number. *Mathematics of Computation*, 61(203):319–349, 1993.

[LLZZ15] Jiwen Lu, Venice Erin Liong, Xiuzhuang Zhou, and Jie Zhou. Learning compact binary face descriptor for face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10):2041–2056, 2015.

[MAB+18] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. Hamming quasi-cyclic (hqc). *NIST PQC Round*, 2(4):13, 2018.

[May21] Alexander May. How to meet ternary LWE keys. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 701–731, Virtual Event, August 2021. Springer, Cham.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic codes. *Coding Theory*, 4244:114–116, 1978.

[MDC05] Jonathan Marchini, Peter Donnelly, and Lon R Cardon. Genome-wide strategies for detecting multiple loci that influence complex diseases. *Nature genetics*, 37(4):413–417, 2005.

[Meu12] Alexander Meurer. *A Coding-Theoretic Approach to Cryptanalys*. PhD thesis, PhD thesis, Ruhr Universität Bochum, 2012.

[Mih10] Joseph P. Mihalcik. An analysis of algorithms for solving discrete logarithms in fixed groups. master's thesis. Naval Postgraduate School, 2010. `https://calhoun.nps.edu/bitstream/handle/10945/5395/10Mar_Mihalcik.pdf`.

[Mis13] Hisanori Mishima. Wifc (world integer factorization center), 2013. `https://www.asahi-net.or.jp/~KC2H-MSM/mathland/matha1/`.

[ML91] Mark Manasse and Arjen Lenstra. Rsa-100 challenge, 1991. `https://www.ontko.com/pub/rayo/primes/hr_rsa.txt`.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, Berlin, Heidelberg, December 2011.

[MNP06] Rajeev Motwani, Assaf Naor, and Rina Panigrahi. Lower bounds on locality sensitive hashing. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 154–157, 2006.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 203–228. Springer, Berlin, Heidelberg, April 2015.

[Mon95] Peter L. Montgomery. A block Lanczos algorithm for finding dependencies over gf(2). In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT'95*, volume 921 of *LNCS*, pages 106–120. Springer, Berlin, Heidelberg, May 1995.

[MP69] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 479:480, 1969.

[MS07] Lorenz Minder and Amin Shokrollahi. Cryptanalysis of the sidelnikov cryptosystem. In *Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26*, pages 347–360. Springer, 2007.

[MSL+07] Solomon K Musani, Daniel Shriner, Nianjun Liu, Rui Feng, Christopher S Coffey, Nengjun Yi, Hemant K Tiwari, and David B Allison. Detection of gene× gene interactions in genome-wide association studies of human population data. *Human heredity*, 63(2):67–84, 2007.

[Mur99] Brian Antony Murph. Polynomial selection for the number field sieve integer factorisation algorithm, 1999. `https://maths-people.anu.edu.au/~brent/pd/Murphy-thesis.pdf`.

[NFK] Shintaro Narisada, Kazuhide Fukushima, and Shinsaku Kiyomoto. Record 21, syndrome decoding in the goppa-mceliece setting. `https://decodingchallenge.org/goppa/record/21`. Accessed: 2023-01-14.

[NFK21] Shintaro Narisada, Kazuhide Fukushima, and Shinsaku Kiyomoto. Fast gpu implementation of dumer's algorithm solving the syndrome decoding problem. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 971–977. IEEE, 2021.

[NFK22] Shintaro NARISADA, Kazuhide FUKUSHIMA, and Shinsaku KIYOMOTO. Multiparallel mmt: Faster isd algorithm solving high-dimensional syndrome decoding problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2022.

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.

[PBH98] Vera Pless, Richard A Brualdi, and William Cary Huffman. *Handbook of coding theory*. Elsevier Science Inc., 1998.

[PBLvT09] Christiane Peters, DJ Bernstein, T Lange, and H van Tilborg. Explicit bounds for generic decoding algorithms for code-based cryptography. In *Proc. of the International Workshop on Coding and Cryptography, WCC*, pages 68–180, 2009.

[Per92] Rene Peralta. On the distribution of quadratic residues and nonresidues modulo a prime number. *Mathematics of Computation*, 58(197):433–440, 1992.

[Pet10] Christiane Peters. Information-set decoding for linear codes over f q. In *International Workshop on Post-Quantum Cryptography*, pages 81–94. Springer, 2010.

[Pol78] John M Pollard. Monte carlo methods for index computation. *Mathematics of computation*, 32(143):918–924, 1978.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[RS04] Alexander Russell and Igor E Shparlinski. Classical and quantum function reconstruction via character evaluation. *Journal of Complexity*, 20(2-3):404–422, 2004.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[SBBF11] Christoph Strecha, Alex Bronstein, Michael Bronstein, and Pascal Fua. Ldahash: Improved matching with smaller descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 34(1):66–78, 2011.

[Sen11] Nicolas Sendrier. Decoding one out of many. In *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4*, pages 51–67. Springer, 2011.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(4):623–656, 1948.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, oct 1997.

[SS81] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.

[SS92] Vladimir Michilovich Sidelnikov and Sergey O Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes. 1992.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In *International Colloquium on Coding Theory and Applications*, pages 106–113. Springer, 1988.

[Ste89] Jacques Stern. A method for finding codewords of small weight. *Coding theory and applications*, 388:106–113, 1989.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

[Tea] The CADO-NFS Development Team. CADO-NFS, an implementation of the number field sieve algorithm. development version: `http://cado-nfs.gforge.inria.fr/`.

[tRAC+99] Herman J.J. te Riele, Karen Aardal, Stefania Cavallar, Walter M. Lioen, Bruce Dodson, Jeff Gilchrist, Arjen K. Lenstra, Paul Leyland, Joel Marchand, Peter L. Montgomery, Francois Morain, Gerard Guillerm, Alec Muffett, Brian Murphy, Chris Putnam, Craig Putnam, and Paul Zimmermann. Rsa-150 challenge, 1999. `https://www.ontko.com/pub/rayo/primes/hr_rsa.txt`.

[tRCL+99] Herman J.J. te Riele, Stefania Cavallar, Walter M. Lioen, Peter L. Montgomery, Bruce Dodson, Arjen K. Lenstra, Paul Leyland, Brian Murphy, and Paul Zimmermann. Rsa-140 challenge, 1999. `https://www.ontko.com/pub/rayo/primes/hr_rsa.txt`.

[TS16a] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *Post-Quantum Cryptography*, pages 144–161. Springer, 2016.

[TS16b] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 144–161. Springer, Cham, 2016.

[UV21] Aleksei Udovenko and Giuseppe Vitto. Breaking the $ikep182 challenge. Cryptology ePrint Archive, Report 2021/1421, 2021. `https://eprint.iacr.org/2021/1421`.

[Val88] Leslie G Valiant. Functionality in neural nets. In *COLT*, volume 88, pages 28–39, 1988.

[Val12] Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and juntas. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 11–20. IEEE, 2012.

[van90] Johan van Tilburg. On the McEliece public-key cryptosystem. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 119–131. Springer, New York, August 1990.

[Var21a] Various. pqc-forum: Round 3 official comment: Classic mceliece, 2021. Available at: `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/EiwxGnfQgec`.

[Var21b] Various. pqc-forum: Security strength categories for code based crypto (and trying out crypto stack exchange), 2021. Available at: `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/6XbG66gI7v0`.

[Vas] Valentin Vasseur. Dumer implementation with doom. `https://github.com/vvasseur/isd`. Accessed: 2023-01-15.

[VL98] Jacobus Hendricus Van Lint. *Introduction to coding theory*, volume 86. Springer Science & Business Media, 1998.

[vT94] Johan van Tilburg. Security-analysis of a class of cryptosystems based on linear error-correcting codes. 1994.

[vW94] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu, editors, *ACM CCS 94*, pages 210–218. ACM Press, November 1994.

[vW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, January 1999.

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Berlin, Heidelberg, August 2002.

[WL15] Maoning Wang and Mingjie Liu. Improved information set decoding for code-based cryptosystems with constrained memory. In *International Workshop on Frontiers in Algorithmics*, pages 241–258. Springer, 2015.

[XXX19] Ning Xie, Shuai Xu, and Yekun Xu. A new coding-based algorithm for finding closest pair of vectors. *Theoretical Computer Science*, 782:129–144, 2019.