`CryptographicEstimators`: a Software Library for Cryptographic Hardness Estimation

Andre Esser, Javier Verbel, Floyd Zweydinger, and Emanuele Bellini

Technology Innovation Institute, UAE
{andre.esser, javier.verbel, floyd.zweydinger, emanuele.bellini}@tii.ae

Abstract. The estimation of the computational complexity of hard problems is essential for determining secure parameters for cryptographic systems. To date, those estimations are often performed in an ad-hoc manner. This led to a scattered landscape of available estimation scripts, with multiple scripts for the same problem with varying outputs. Overall, this complicates the task of reaching consensus on the hardness of cryptographic problems. Furthermore, for designers it makes it difficult to gather precise information on the concrete difficulty of the underlying problems. Especially in the light of the still ongoing NIST PQC standardization effort and the upcoming call for post-quantum secure digital signature schemes there is a pressing need for a reliable point of access for concrete security estimates.

In this work we present the first open-source software library entirely dedicated to cryptographic hardness estimation, the `CryptographicEstimators` library. In contrast to most previous estimators, this library follows a modern object-oriented software architecture, which provides a wide variety of features. Overall the design is optimized to ease extending existing estimators by new algorithms and makes it simple to integrate completely new estimators.

In this work we further specify the algorithmic cost model underlying the estimators. In order to provide a starting point for the project, we gathered and integrated estimators for six different hardness assumptions, including the syndrome decoding problem, the multivariate quadratic problem, the code equivalence problem, the permuted kernel problem and different flavors thereof. In our effort of gathering those estimation scripts, we also normalized those estimates to fit into the cost model and to measure the same unit operations.

Keywords: computational hardness · parameter selection · hardness assumptions · open source software · estimators

1 Introduction

Due to the shift from classical security towards post-quantum secure systems in recent years, there is a wide diversity among used hardness assumptions for cryptographic primitives. Especially, the NIST PQC effort for standardization of

post-quantum secure cryptographic schemes has led to an unprecedented range of security foundations being taken into consideration. Further, the variety of this list is expected to grow with the upcoming NIST PQC call for digital signatures.

In order to select suitable parameters that guarantee certain levels of security, all these schemes require an estimation of the computational complexity to solve the underlying hard problem. Therefore, cryptographic hardness estimators are essential for designing secure cryptographic systems by providing this concrete measure of computational complexity. Over the last years, many successful estimator projects have been established for different problems, such as the learning with errors (LWE) estimator [1] (now evolved into the lattice estimator[1]) or the syndrome decoding (SD) estimator [34]. Those are examples of two of the most sophisticated and developed estimator projects, and yet they are mostly collections of routines implementing algorithm's complexity formulas. A fact that adds a burden to maintaining and collaboratively advancing these estimators. The recently designed multivariate quadratic (MQ) estimator [11] does a first step towards following modern software-design principles by being based on an object-oriented design.

The reason for not following a well-designed software architecture is that most of these estimators are ad-hoc projects. They serve as a proof of concept for a corresponding publication, rather than being designed as a collaboratively maintained project. There exist many more estimators of smaller scale and for many problems there exist multiple different estimation scripts, producing more or less varying hardness estimates. This scattered project landscape and dependencies between estimators make it challenging for designers to gather information on the true complexity and to reach a consensus on the computational hardness of a problem. An issue that can lead to incorrect or incomplete hardness estimates and in turn insecure systems.

The only estimator, which has found adaptation as a collaborative community project so far is the LWE estimator. An estimator project that has shown which impact such an adaptation can have on the understanding and the consensus-building of the computational hardness of a problem.

In this work, we introduce the first open-source library dedicated entirely to cryptographic hardness estimation, the `CryptographicEstimators` library. Inspired by the recent MQ estimator, the `CryptographicEstimators` library follows a modern object-oriented class design that makes it easy to extend existing estimators or to integrate new ones.

The main goal of this library is to consolidate existing estimators, become established as a collaborative community project and, hence, be the basis for newly designed estimators. We provide a platform for professional exchange and the technical tools for collaboratively advancing the estimators. This ultimately contributes to the overall goal of reaching a consensus as community on the individual hardness of each problem. Additionally, to foster its widespread adoption

[1] Available at https://github.com/malb/lattice-estimator

by designers, we provide a full graphical user interface that allows access to the estimators via a *web application*.[2]

In order to provide a starting point and an initial codebase for the project, we gathered and integrated a total of six existing, previously mostly scattered, estimators into the `CryptographicEstimators` library.

Our contribution The main technical contribution of this work is the accompanying open-source `CryptographicEstimators` library, available on GitHub[3]. The main objective of the manuscript is to set the necessary basis to establish the `CryptographicEstimators` library as collaborative project. Therefore, we give a definition of the algorithmic cost model and the estimation methodology that the included estimators follow. Further, we describe the general software architecture of the library and its functionalities on a high level. Eventually, we give an overview of the included estimators, such as a definition of the corresponding hardness assumptions and the algorithms covered. Note that this work is not intended as a *guide for users*[4] or a *technical whitepaper*, which will be released separately in short time.

The main contributions of the `CryptographicEstimators` library can be summarized as follows.

Accessibility. The `CryptographicEstimators` library provides an essential tool for cryptographic system design. They serve as a single point of access for designers to obtain reliable estimates on the hardness of chosen parameters. For an even wider adaptation, a fully automated graphical user interface, accessible via a web application, is provided.

Dependencies. Dependencies between the hardness of problems can be modelled ideally in the `CryptographicEstimators` library. Already in the initial state, some included estimators use other estimators to obtain computational estimates for certain subroutines.

Consensus. Even if not all estimators' initial states should reflect the consensus of the whole community, the collaborative nature of the project provides an ideal platform for consensus-building. Furthermore, the `CryptographicEstimators` library encourages open discussions and welcomes contributions from the wider community, which leads to further refined estimates and improved reliability.

Collaboration. The `CryptographicEstimators` library is the first project launched intentionally as a collaborative estimator project. The impact that such a community driven project can have on the understanding of the hardness of a computational problem is witnessed by the LWE (now lattice) estimator.

[2] Web application accessible at https://estimators.crypto.tii.ae

[3] Library source code accessible at https://github.com/Crypto-TII/CryptographicEstimators

[4] User guide accessible at https://github.com/Crypto-TII/CryptographicEstimators/blob/develop/User_Guide.ipynb

Software Design. The library is build according to modern software-design principles. It follows a sophisticated object-oriented architecture and is fully modular. All basic functionalities are provided by the base (or parent) classes making the integration of new estimators and the extension of existing ones especially simple.

As a second main contribution we consolidate and integrate estimators for six different cryptographic hardness assumptions into the `CryptographicEstimators` library. This includes estimators for the multivariate quadratic (MQ) problem [11], the linear equivalence (LE) problem [7,15], the permutation equivalence (PE) problem [15], the permuted kernel (PK) problem [56], the binary syndrome decoding (SD) problem [34,36] and the syndrome decoding problem over larger fields [15,52]. We selected these problems, as many recent constructions base their security on exactly those problems [6,16–18,20,40]. In particular, those are all signature constructions of which some are likely to be submitted to the upcoming NIST call for post-quantum secure digital signature schemes. Therefore, the `CryptographicEstimators` library comes timely in supporting the secure parameter selection process of this standardisation effort. In case of multiple available estimation scripts for the same algorithm, we integrated the one that found the larger adoption throughout the community.

In this context we normalize all complexity estimations to the same unit operations to make the estimations of different algorithms comparable. This, together with slight corrections and performance improvements we integrate, requires a few adaptations to the existing estimation scripts. We document all these changes and additionally provide test scripts in the `CryptographicEstimators` library that show how to perform the conversion to obtain the exact numbers from the online available scripts, up to a negligible tolerance.

In total, the initial library contains six different estimators which together provide computational estimates for the time and memory complexity of thirty-two different algorithms.

Outline In Section 2 we describe the algorithmic cost model and the methodology for measuring time and memory complexity used by all integrated estimators, including a discussion about memory access costs. In Section 3 we outline the technical design architecture of the `CryptographicEstimators` library, which includes an overview of the class design as well as the main features provided by the library. Finally, in Section 4 we discuss the different hardness assumptions currently covered by the `CryptographicEstimators` library, including a short overview of included algorithms and necessary problem specific characteristics.

2 Cryptographic Hardness Estimation

The estimation of the hardness of a given cryptographic problem requires to estimate the time and memory complexity of different algorithms. In the context of the `CryptographicEstimators` library, all estimations are performed in the

Random Access Machine (RAM) model of computation, which is the established model for cryptographic hardness estimation. The reasons for this widespread adaptation are diverse, but main factors making the model attractive are its *simplicity* and its *flexibility*. It allows to directly map human readable descriptions of algorithmic procedures into a cost model and is independent of precise hardware implementations.

Further, the model omits the cost of memory access. In particular, the RAM model assigns only a single unit of time complexity to any memory access. While this makes the model even simpler to use, it is not accurately modeling real-world behavior. In practice, memory access, especially if memory usage is high, can be quite costly. We therefore discuss in Section 2.3 how to account for memory access costs in the RAM model. But first we specify more precisely how the time and memory complexity for each algorithm are computed.

2.1 Measuring Time and Memory Complexities

For every specific problem $\mathcal{P}$, we define an elementary operation op and a basic element el, as a common unit to measure the time complexity and memory complexity among all algorithms solving $\mathcal{P}$. For example, in the particular case of the MQ problem over the finite field $\mathbb{F}_q$ (see Definition 4.2), the operation op is defined as the multiplication of two elements in $\mathbb{F}_q$, and the object el is one element in $\mathbb{F}_q$.

Given an instance of $\mathcal{P}$ and an algorithm $\mathcal{X}$ solving it, we define the time complexity of $\mathcal{X}$ as the number of times the elementary operation op (or an equivalent one[5]) has to be executed by $\mathcal{X}$ to solve the input instance. The memory complexity is given by the maximum number of elements el that has to be stored at the same time during the execution of $\mathcal{X}$.

Adhering to this standard methodology ensures comparability of all algorithms solving $\mathcal{P}$. In order to allow for comparison of the hardness among different problems, each problem $\mathcal{P}$ defines how to translate a basic operation op into bit operations, i.e., how many bit operations correspond to one op operation, and how many bits are needed to store one basic element el.

2.2 Accuracy of the Complexity Estimate

In order to make the complexity estimations of different algorithms solving a specific problem or more generally the hardness estimations of different problems comparable, we specify to which degree of accuracy executed operations are counted towards the estimate.

Generally, we differentiate two modes of estimation accuracy, which we label $\mathsf{Estimate}$ and TildeO . Estimations obtained in the TildeO mode are based on an asymptotic approximation of the running time that might disregard polylogarithmic factors in the main term. Precisely, if the running time can be expressed asymptotically as $\tilde{\mathcal{O}}(T(n))$, n being the problem dimension, then the output

[5] For example, in RAM model a memory access forms an equivalent operation.

value is simply obtained by plugging in the concrete dimension n into $T(n)$. We integrate this mode for further investigation and comparison purposes, but the standard mode of operation is the Estimate mode.

The Estimate mode is defined to provide the most precise cost estimate, according to the current state-of the-art. Therefore all constant and polynomial factors necessary to solve the problem have to be taken into account. If not specified otherwise, we refer with estimates always to the Estimate mode.

2.3 Modelling Memory Access Costs

The RAM model can be easily adapted to account for memory access costs. Instead of counting only one unit of time complexity per memory access we define a memory access cost c. Therefore one memory access is equivalent to c executions of op. Practice has shown, that the value of $c = f(M)$ is usually a function in the total memory consumption M that grows in M. This means if only a small amount of memory is consumed by the algorithm, individual accesses are cheap, otherwise they are more costly. A behavior that is related to physical distances traveled depending on the memory size.

This introduces additional dependencies between the time complexity T and the memory complexity M of an algorithm that complicates the optimization of its running time. Therefore, usually, an upper bound on the time complexity including the memory access cost is obtained by assuming the worst case of T memory accesses all performed on a memory of size M, leading to a total algorithmic cost of $T_M = T \cdot f(M)$. However, usually the memory accesses are only a fraction of the whole time complexity and not all of them are performed on a memory of size M. In order to compensate for this overestimation, the function f is sometimes chosen to underestimate the real-world behavior of memory access costs.

Generally, using this methodology for including memory access costs in the algorithmic cost model should be customized for each problem or more precisely each algorithm. It depends heavily on the ratio between the time complexity and the performed memory accesses, as well as on the memory usage over time, which function f leads to the most accurate estimates.

3 Technical Design and Functionalities of the Library

In contrast to previous ad-hoc estimator projects, the `CryptographicEstimators` library follows a pre-designed software architecture, which is designed to provide a base infrastructure for the hardness estimation of computational problems. In this framework, main functionalities and common attributes are provided by default to newly added estimators and algorithms. This design makes it especially easy to maintain and extend existing estimators as well as integrate new ones at minimal overhead. In this section we describe the main technical design, i.e., the class architecture, as well as the main features the library offers to users as well as to contributors on a high level. This section does not aim at a full technical

description of how features are realized, but outlines the general concept of the estimation framework.

3.1 Base Classes

The main design of the `CryptographicEstimators` library splits into three general base classes: The *problem*, the *algorithm* and the *estimator* classes. An overview of this structure is given in Figure 1. The base classes are abstract, which means they just provide a blueprint in form of basic attributes and functionalities. Each specific problem, estimator or algorithm is then implemented inheriting from the corresponding base class. Meaning it automatically follows the given blueprints by including all its functionalities and attributes by default.

The main principal of the design is that each algorithm as well as each estimator holds an object of the problem they are estimating. In addition the estimator holds an object of each algorithm that is applicable to the corresponding problem instance. Once called, the estimator can then gather the information on the complexity estimation based on the problem and algorithm objects it contains.

Base problem class. The base problem class contains the common attributes and methods amongst all cryptographic problems, such as a list of problem parameters and an amount of existing solutions. Classes for specific problems like the SD problem (see Definition 4.1) or the MQ problem (see Definition 4.2) are inherited from the base problem class. Those classes then implement all problem specific attributes and functionalities. Problem objects are used to describe concrete problem instances. Additionally the base problem class defines abstract functions to convert elementary operations as well as the memory unit to bit complexities, which then have to be implemented by every specific problem class. Further, the base class defines a memory bound parameter, which defines the maximum amount of memory, in the unit measured or in bits, available when solving the problem.

Base algorithm class. Each algorithm solving a specific problem is inheriting from the base algorithm class. The base algorithm class implements abstract methods to compute the time and memory complexities of an algorithm. Again, the concrete instantiations of those functions must be provided by each concrete algorithm. Further, the base class implements the functionality of automatically searching for an optimal parameter set of the algorithm that minimizes the time complexity, while ensuring all given constraints. We refer to this parameter set as the *optimal parameters*. If the time or memory complexity is requested it triggers the search for these optimal parameters and then returns the time and memory complexities for those optimal parameters. There are various additional attributes and functionalities provided by the algorithm base class. This includes a way to specify optimization parameters considered by the automatic optimization routine, accessing the optimal parameters, triggering their search, resetting the configuration and specifying optimization ranges.

Base estimator class. An estimator of a problem $\mathcal{P}$ is mainly (1) A set of algorithm objects for solving $\mathcal{P}$, and (2) a set of functionalities to facilitate the manipulation, configuration and visualization of all included algorithms and their estimations. Again, for each problem, an estimator is created using the same blueprint as the base estimator, i.e. by inheriting from the same base class. However, in contrast to the specific problem or algorithm classes, no further routine needs to be implemented. The existence of this problem specific estimator class is more a technical necessity.

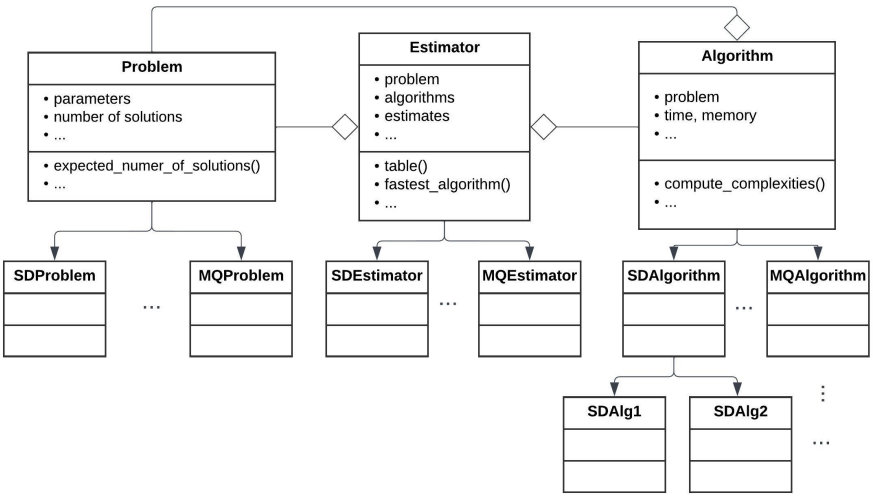

Fig. 1: UML diagram of `CryptographicEstimators` class design

3.2 Features of the Library

In the following, we highlight some of the main features the `CryptographicEstimators` library provides to users and to contributors.

Usage Features of the Estimators Let us start with features concerning the usage of the library.

General functionalities of the estimator. When an estimator object is created over an instance of a problem $\mathcal{P}$, it automatically builds the algorithm objects of all algorithms applicable for the concrete problem instance $\mathcal{P}$. The estimator provides individual access to these algorithms and the possibility to configure them. In addition, the estimator allows to set a configuration to all algorithms

simultaneously. Once estimates are computed, i.e., the optimization of the running time of applicable algorithms have been performed, they are stored within the objects for further investigation. Therefore, if requested again, the estimates do not have to be recomputed. However, on demand, a recomputation can be triggered. The estimator provides functionalities to visualize and compare the computed estimates. It implements features for selective access, as retrieving the fastest algorithm for solving $\mathcal{P}$ or its time complexity. Additionally, it allows access to all internal states of the algorithms after their optimization for further investigation.

Customization of the Estimation. The estimator class provides several customization features that affect how estimates are calculated.

- *Restricting algorithm parameters*: There are several means to customize the parameter optimization of an algorithm. It is possible to set custom ranges for all or certain parameters considered in the running time minimization. Parameters can also be fixed to certain values. Besides allowing to explore the algorithm's complexity under certain parameter constraints, this feature can also accelerate the optimization process.
- *Excluding algorithms*: Algorithms can also be completely excluded from the estimation, which can further accelerate computations.
- *Specifying memory access cost:* The estimation can be performed under the consideration of memory access cost. Therefore, an individual memory access cost function (see Section 2.3) can be provided or one of the preset models can be chosen.[6] Finally, the memory access cost is not only added to the final result, but taken into account while optimizing the parameters for time-complexity minimization.
- *Choosing bit complexities:* Following the description from Section 2.1, the estimates can be provided in terms of the elementary operations specified for the problem or their translation to the corresponding amount of bit operations. In case bit complexities are selected, memory is analogously provided in bits of memory rather than unit elements. The standard output of the estimator is always using *bit complexities*.
- *Limiting the memory complexity:* The estimator allows to provide a memory upper bound in bits or unit elements. This memory upper bound is then strictly enforced in the parameter optimization.
- *Setting the complexity type*: The estimator can be configured to either use the Estimate or the TildeO type for estimation accuracy (see Section 2.2). If not specified, the default mode of operation is Estimate .
- *Including quantum complexity*: The estimator allows to include quantum estimations in the output. So far, none of the integrated algorithms provides a function to estimate its quantum complexity. However, the feature is available for easy extension.

[6] The preset models include square-root, cube-root or logarithmic memory access cost functions.

Graphical user interface The `CryptographicEstimators` library provides a fully graphical user interface accessible via a web application. In this application, a user can estimate the hardness of any cryptographic problem available in the `CryptographicEstimators` library and download the results as a LaTeX table. In addition, to ease reproducibility of estimates it allows to export and import the configuration used to obtain a specific estimation.

Development Features of the Estimators. Besides following a modular software architecture, the `CryptographicEstimators` library provides various additional features that ease extension and maintenance of included estimators as well as the integration of new ones.

Adding new algorithms. The base classes of the library provide the functionality of automatic parameter management and optimization. Therefore a new algorithm can be included by only specifying the names and ranges of its optimization parameters and a corresponding time and memory complexity function depending on those parameters. The minimization of the running time is then automatically performed in the given ranges considering all given constraints, such as a memory limit. The base classes keep track of found parameters by default. After providing those basic functions the algorithm is automatically included in future runs of the estimator.

Integrating new estimators. The library facilitates the integration of new estimators. It provides a script to generate the minimal code necessary to include a new estimator. This generated code includes for example a concrete problem and algorithm class with the necessary inheritance structure. After executing the script, only the problem specific parameters need to be specified and then algorithms can be added to this new estimator.

Extending the GUI. The GUI generation is fully automatized. Therefore for each estimator there exist an entry in a GUI configuration file specifying which input fields should be displayed, what kind of GUI element to assign, e.g. a text field, and to which internal configuration parameter those inputs are mapped. From this configuration file the GUI is then generated. Again, the library provides a script to generate a basic entry for an new estimator in this configuration file containing the common input fields. This entry needs then only to be extended by the problem specific parameters.

Customizing the Optimization. In case the automatic optimization over the given parameter ranges should not yield the desired performance, we implement various measures to allow developers to easily speed up the optimization. This includes specifying parameter dependencies, defining early abort criteria for the time complexity computation or providing completely custom optimization procedures.

4 Covered Estimators

In this section we outline the six different estimators which together form the initial state of the library. For each estimator we define the corresponding hardness assumption and give a brief overview or classification of the included algorithms. For the full details on the respective algorithms we refer the reader to the relevant literature. Further, we detail the elementary operation and memory unit used in the estimation and provide a profile summarizing the current state of the estimator. Additionally, for each estimator we outline the cause of deviations from the original scripts, if any, such as slight corrections and normalization to the elementary operations. Note that for all estimators we include test scripts that verify them against the original scripts, precisely detailing how to perform the conversion.

Note that for every problem we assume the existence of at least one solution, which is typical for the cryptographic setting.

4.1 Syndrome Decoding Problem

Most code-based schemes, so as all schemes currently in the 4th round of the NIST standardisation process, build their computational hardness upon the decoding of random linear codes. A linear code $\mathcal{C}$ is a k-dimensional sub-space of $\mathbb{F}_q^n$. One usually describes $\mathcal{C}$ via a parity check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, i.e., $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n \mid \mathbf{Hc} = \mathbf{0}\}$. Further, let $\mathbf{x} = \mathbf{c} + \mathbf{e}$ be an erroneous code word with $\mathbf{e}$ of small known hamming weight ω. Then $\mathbf{s} = \mathbf{Hx} = \mathbf{He}$ is defined as the *syndrome*. Thus, decoding $\mathbf{x}$ to $\mathbf{c}$ is equivalent to recovering the error vector $\mathbf{e}$ with weight ω from given syndrome $\mathbf{s}$. More formally, we give the following definition.

Definition 4.1 (Syndrome Decoding Problem (SDP)). *Given a parity check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ of a linear code $\mathcal{C}$ of length n and dimension k, a syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$ and a target weight $\omega \in \mathbb{N}$. The Syndrome Decoding Problem asks to find an error vector $\mathbf{e} \in \mathbb{F}_q^n$ with $\mathbf{He} = \mathbf{s}$ and $\mathrm{wt}(\mathbf{e}) = \omega$.*

Our library provides two estimators for the SDP, one for the specific case of the binary field, i.e., the case of $q = 2$ and one for general q. We make this distinction as there are several algorithms specifically tailored to the binary case, not directly applicable for $q > 2$.

Independent of the choice of q, the currently best known strategy to tackle general instances of the SDP is Information Set Decoding (ISD) introduced by Prange [54] and its subsequent improvements. Let us briefly sketch this general technique, first for the the binary case after which we outline differences to the case of arbitrary q.

ISD via Prange. Let $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ be a permutation Matrix. Then $\mathbf{HP}$ is still a valid problem instance with solution $\mathbf{P}^{-1}\mathbf{e}$. Now assume that $\mathbf{P}^{-1}\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) =$

$(\mathbf{e}_1, \mathbf{0}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^k$, i.e., the whole weight of $\mathbf{e}$ distributes onto $\mathbf{e}_1$ via the permutation. Then by applying Gaussian Elimination to $\mathbf{H}$ modelled as multiplication with an invertible matrix $\mathbf{Q} \in \mathbb{F}_2^{(n-k)\times(n-k)}$, the following equation holds

$$\mathbf{QHe} = (\mathbf{I}_{n-k}\mathbf{H}_1)(\mathbf{e}_1, \mathbf{e}_2) = \mathbf{e}_1 = \mathbf{Qs} \tag{1}$$

In other words, if the error positions are located only in the first $n-k$ coordinates of the error vector $\mathbf{P}^{-1}\mathbf{e}$, we are able to identify this case, by testing if $\mathrm{wt}(\mathbf{Qs}) = \mathrm{wt}(\mathbf{e}_1) = \mathrm{wt}(\mathbf{e}) = \omega$. From here, one recovers $\mathbf{e}$ efficiently as $\mathbf{P}(\mathbf{e}_1, \mathbf{0})$. In order to find a permutation distributing the weight in such a way, random permutations are tested, where the expected amount of permutations until success is

$$\frac{\binom{n}{\omega}}{\binom{n-k}{\omega}}.$$

For each such permutation the algorithm computes the weight of $\mathbf{Qs}$ until it is equal to ω.

Advanced ISD procedures. The dominating factor of the running time of Prange's algorithm is the amount of permutations needed until success. In order to decrease this amount of permutations all modern ISD algorithms allow for some weight p being located in the last k coordinates of $\mathbf{P}^{-1}\mathbf{e}$, i.e., $\mathbf{e}_2 \neq \mathbf{0}$. Therefore Equation (1) changes to

$$\mathbf{QHe} = (\mathbf{I}_{n-k}\mathbf{H}_1)(\mathbf{e}_1, \mathbf{e}_2) = \mathbf{e}_1 + \mathbf{H}_1\mathbf{e}_2 = \mathbf{Qs}.$$

The algorithms then use different strategies to enumerate $\mathbf{e}_2$ and report success when they find an $\mathbf{e}_2$ which satisfies $\mathrm{wt}(\mathbf{Qs} + \mathbf{H}_1\mathbf{e}_2) = \mathrm{wt}(\mathbf{e}_1) = \omega - p$, returning $\mathbf{e} = \mathbf{P}(\mathbf{e}_1, \mathbf{e}_2)$. The amount of permutations needed until success now improves to

$$\frac{\binom{n}{\omega}}{\binom{n-k}{\omega-p}\binom{k}{p}},$$

while each iteration comes at a higher cost, now also involving the enumeration procedure. The early variant by Lee-Brickell [47] simply iterates over all choices for $\mathbf{e}_2$, while later improvements like Stern or Dumer use meet-in-the-middle strategies [32, 60].

Binary Syndrome Decoding For the binary case, most recent improvements use for enumeration of the weight-p vector $\mathbf{e}_2$ a binary search-tree based approach, i.e., a multi-level meet-in-the-middle. In order to further improve the enumeration they combine the search-tree with the *representation technique* [9, 49] and *nearest-neighbor search* [23, 24, 33, 50]. The practical complexity of those algorithms has been studied multiple times [2, 13, 34–36, 42].

Esser and Bellini give a comprehensive study of all major ISD improvements, with several practical tweaks [34] and also provide an open-source estimator tool [10]. We integrated all algorithms from their estimator into the

`CryptographicEstimators` library. Following the work of Esser-Bellini, we count as elementary operations $\mathbb{F}_2^n$ vector additions. Therefore, a conversion to bit complexities (in terms of time and memory) is obtained by multiplication with n. Further we integrated a recent memory improvement by Esser-Zweydinger [36] as a separated algorithm, yielding a total of ten different algorithms used for estimating the hardness of the binary syndrome decoding problem. In the following we summarize the current estimator profile:

Binary Syndrome Decoding Estimator	
Name	`SDEstimator`
Parameters	(n, k, w): length, dimension, weight
Elementary operation (op)	$\mathbb{F}_2^n$ vector addition
Memory unit (el)	vector in $\mathbb{F}_2^n$
Bit complexity factor time	n
Bit complexity factor memory	n
No. of algorithms	10
Available modes	Estimate and TildeO

Table 1: Profile of the binary SDP estimator

In addition to the always available Estimate mode, the binary syndrome decoding estimator offers the possibility of computing estimates in the TildeO mode. Essentially, this means it can be used to compute the asymptotic workfactors[7] of most of the included algorithms. In [34] this functionality was provided via an accompanying C-library which complicates the library set-up. Therefore, we got rid of this dependency by providing the functionality of this C-library via pure python. We perform the necessary numerical optimization via the *scipy* python library, following the example of [22].

Deviations from online available scripts. The estimates for the binary SDP in `CryptographicEstimators` match the online available scripts from [34,36]. Only the case of memory access costs in combination with bit complexities has slight deviations. This is because [34] computed the memory access costs always based on the number of unit elements el that have to be stored, while we use the number of bits in that case.

Decoding over $\mathbb{F}_q$ Many recent works base their security directly or indirectly on the hardness of the non-binary syndrome decoding problem [5, 6, 19, 40]. Therefore, the `CryptographicEstimators` library also includes an estimator for the hardness of the SDP over fields $\mathbb{F}_q$, where $q > 2$. The algorithms in those cases are essentially adaptations of rather simple ISD algorithms from the binary

[7] The running time of ISD algorithms for $\omega = c_\omega n$ can be written as $T = 2^{c_T \cdot n}$, where c_ω and c_T are constants. The constant c_T is usually referred to as *workfactor*.

setting. Namely the most basic algorithm by Prange and the improvements made by Lee-Brickell and Stern.

Since the change of the base field does not change the expected number of permutations, Prange's algorithm applies without changes. Contrary, the complexity of enumeration based improvements like Lee-Brickell and Stern depend on the field size and therefore change. In the case of Stern, we integrated an existing estimator by Peters, who established the complexity of the procedure for non-binary fields in [52]. For Lee-Brickell we integrate an estimation routine from Beullens [15] with a small correction.

In contrast to the Syndrome Decoding Estimator over $\mathbb{F}_2$ we count as elementary operations additions over $\mathbb{F}_q$, which follows the estimator by Peters. The reasoning for not considering multiplications is that the field size is usually quite small and multiplications can be implemented via lookup-tables. The estimator of the Syndrome Decoding Problem of $\mathbb{F}_q$ is summarized with the following profile:

Syndrome Decoding Estimator over $\mathbb{F}_q$	
Name	`SDFqEstimator`
Parameters	(n, k, w, q): length, dimension, weight, field size
Elementary operation (op)	$\mathbb{F}_q$ addition
Memory unit (el)	$\mathbb{F}_q$ element
Bit complexity factor time	$\log_2 q$
Bit complexity factor memory	$\log_2 q$
No. of algorithms	3
Available modes	Estimate

Table 2: Profile of the SDP estimator for non-binary fields

Deviations from online available scripts. We implement a slight correction of the LeeBrickell script using a factor of $\binom{k}{p}$ rather than k^p in its running time. As the optimal p is usually found as 2, this results in a one bit improved complexity estimate. Also the normalization to $\mathbb{F}_q$ additions as elementary operation causes a necessary scaling of the estimates from [15] by a factor of n.

4.2 Multivariate Quadratic Problem

Let $\mathbb{F}_q$ denote a finite field with q elements, and let $\mathbb{F}_q[x_1, x_2, \ldots, x_n]$ denote the set of all polynomials with variables $x_1, x_2, \ldots, x_n$ and coefficients in $\mathbb{F}_q$. A quadratic polynomial in $\mathbb{F}_q[x_1, x_2, \ldots, x_n]$ is a polynomial of the form

$$\sum_{1 \leq i < j \leq n} a_{i,j} x_i x_j + \sum_{i=1}^{n} b_i x_i + c,$$

where $a_{i,j}, b_i, c \in \mathbb{F}_q$. The MQ problem is defined as the problem of finding a common preimage to a set of *multivariate quadratic equations* over a finite field. Formally,

Definition 4.2 (Multivariate Quadratic Problem (MQP)). *Given a set of m quadratic polynomials $f_1, f_2, \ldots, f_m \in \mathbb{F}_q[x_1, x_2, \ldots, x_n]$ and elements $b_1, b_2, \ldots, b_m \in \mathbb{F}_q$. The MQ problem asks to find $(a_1, a_2, \ldots, a_n) \in \mathbb{F}_q^n$ such that*

$$f_1(a_1, a_2, \ldots, a_n) - b_1 = \cdots = f_n(a_1, a_2, \ldots, a_n) - b_n = 0.$$

An instance of the MQ problem is said to be underdefined when $m < n$, square when $m = n$, and overdefined when $n < m$.

We are interested in estimating the complexity of random instances of the MQ problem, i.e., those where the f_i are randomly chosen from all possible quadratic polynomials. For random instances, the most efficient MQ algorithms can be roughly classified as pure enumeration algorithms, e.g., fast exhaustive search [25] and algorithms that primarily consist of algebraic operations on the input set of polynomials, as for example the XL algorithm [28]. In the following we outline some general strategies and classifications of those algorithms. For details on the respective algorithms we refer the reader to the corresponding publications or [11] for an overview.

Underdefined instances. There are algorithms that are only applicable to underdefined instances. For instance the algorithm by Kipnis, Patarin and Goubin [45] is effective if $m << n$. In addition, these instances can be efficiently transformed into square ones by either fixing $n - m$ variables or by applying the Thomae-Wolf strategy [61].

Algorithms specialized for $\mathbb{F}_2$. Some algorithms solve the MQ problem in the specific case that the underlying field is binary. For instance, Dinur's algorithms [30,31], Björklund et al.'s algorithm [21], and the Booleansolve algorithm [3].

Gröbner basis algorithm. Gröbner basis algorithms work by finding a Gröbner basis, in the *lexicographical monomial order*, of the ideal generated by the polynomials $f_i - b_i$. For square and overdefined problems, a solution can be efficiently derived from such a basis. Some of these algorithms are Buchberger's algorithm [29], F4 [37], and F5 [38].

Hybrid algorithms. The hybrid algorithms involve an exhaustive search of some of the coordinates of the solution vector followed by one algebraic approach such as XL, F5, etc. Some of these algorithms are Crossbred [44], hybrid-F5 [14], FXL [27] and BooleanSolve [3].

We integrate the estimations of a recently proposed estimator for the MQ problem[8] by Bellini, Makarim, Sanna, and Verbel [11].

Following that work, we define as elementary operation the multiplication over $\mathbb{F}_q$, while the memory complexity is measured in the number of stored elements

[8] The code is accessible at https://github.com/Crypto-TII/multivariate_quadratic_estimator

Multivariate Quadratic Estimator

Name	`MQEstimator`
Parameters	(n, m, q): No. of variables, No. of equations, field size
Elementary operation (op)	$\mathbb{F}_q$ multiplication
Memory unit (el)	$\mathbb{F}_q$ element
Bit complexity factor time	$(\log_2 q)^\theta$, where $\theta \geq 0$
Bit complexity factor memory	$\log_2 q$
No. of algorithms	12
Available modes	Estimate and TildeO

Table 3: Profile of the MQP estimator

of $\mathbb{F}_q$. We assume that storing one $\mathbb{F}_q$ element requires $\log_2 q$ bits of memory and that one multiplication in $\mathbb{F}_q$ can be executed in $(\log_2 q)^\theta$ binary operations. Here $\theta \geq 0$ is an input to the estimator. A summary of the MQ module of the `CryptographicEstimators` library is given in Table 3.

Deviations from online available scripts. The time estimates match those of [11] for all reasonable parameters. We introduced some performance optimizations, which might lead to insignificant deviations for very small input parameters. We modified the memory estimates so that it is always at least the amount of memory needed to store the input polynomials. This modification might lead to deviations in the memory estimates in some corner cases.

4.3 Code Equivalence Problem

Two codes $\mathcal{C}, \mathcal{C}'$ are said to be equivalent, if there exists an isometry mapping one code into the other. In this work, we only cover the more established code equivalence problem in the *Hamming metric*, while recent works also initiated the study of rank-metric equivalences [26, 55].

This means, in our case, an isometry is a map which preserves the Hamming weight. Commonly, there are two flavours of the problem, the *linear* equivalence problem (LEP) and the *permutation* equivalence problem (PEP). In the case of PEP the isometry is a permutation, while in the case of LEP it is a monomial transformation.[9]

The codes are usually represented via their generator matrices $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k \times n}$. Now, if $\mathbf{G}' = \mathbf{GQ}$ for some permutation (or monomial) matrix $\mathbf{Q}$, checking if both codes are equivalent would be trivial by comparing columns of $\mathbf{G}$ and $\mathbf{G}'$, as they would be (scaled) permutations of each other. Therefore, to harden the problem, a change of basis is applied, such that $\mathbf{G}' = \mathbf{SGQ}$ for an invertible matrix $\mathbf{S} \in \mathbb{F}_q^{k \times k}$. Recovering $\mathbf{S}$ and $\mathbf{Q}$ from given $\mathbf{G}, \mathbf{G}'$ is known as the permutation (or linear, when Q is a monomial permutation) equivalence problem.

[9] A monomial transformation is a permutation with additional scaling factors.

Permutation Equivalence. In the case of the permutation equivalence problem the matrix $\mathbf{Q}$ in the equivalence is a permutation matrix. Let us formally define the problem.

Definition 4.3 (Permutation Equivalence Problem (PEP)). *Let* $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k\times n}$ *be generator matrices of linear codes of dimension* k *and length* n*. The* Permutation Equivalence Problem *asks, given* $\mathbf{G}, \mathbf{G}'$*, to find an invertible* $\mathbf{S} \in \mathbb{F}_q^{k\times k}$ *and a permutation matrix* $\mathbf{P} \in \mathbb{F}_q^{n\times n}$ *such that* $\mathbf{G}' = \mathbf{SGP}$*.*

Generally, PEP is known to be not NP-complete unless the polynomial hierarchy collapses [53]. Further, random instances of the problem are solvable in polynomial time due to an attack by Sendrier called *Support Splitting Algorithm* [58]. However, still there is a large set of easy to construct instances for which no efficient algorithms are known.

Support Splitting Algorithm (SSA). This algorithm introduces the concept of a signature of a code, which is invariant under permutations, i.e., the codes $\mathcal{C}, \mathcal{C}'$ associated with the generator matrices $\mathbf{G}, \mathbf{G}'$ have the same signature. The algorithm then punctures both codes in random coordinates, i.e., removes columns from their generator matrices, and checks if the codes still have the same signature, which gives information on the permutation. However, the computation of the proposed signature is exponential in the *hull* of the given codes. The hull of a linear code is the intersection with its dual, i.e., all those codewords that are in $\mathbf{G}$ and $\mathbf{G}^\perp$. For random codes it is known that the hull has constant dimension with high probability [57], making the SSA an efficient algorithm for random instances. Therefore, specific codes, namely (weakly) self-dual codes, with maximal hull dimension $h = \min(k, n-k)$ are used for cryptographic constructions. The complexity of the SSA can be stated as

$$T_{\text{SSA}} = \mathcal{O}(n^3 + n^2 q^h \ln(n)),$$

$\mathbb{F}_q$ operations. For more details on the algorithm the reader is referred to [58].

For trivial hull dimension, i.e., $h = 0$, the SSA is not applicable. But still, there is an efficient strategy to solve the problem via a reduction to the weighted graph isomorphism problem [4]. However, since this case is rather uninteresting for cryptographic applications we do not cover it in the estimator.

Therefore, besides the usual code-parameters the estimator allows for an additional parameter h as an input, specifying the hull dimension. If not provided, maximal hull dimension is assumed.

Low-Weight Codeword-Based Algorithms. The most efficient strategy for solving PEP for codes with large hull is based on finding low-weight codewords. From a sufficiently large list of low-weight codewords $L \subset \mathcal{C}$ and the corresponding permuted set $L' = L\mathbf{P} \subset \mathcal{C}'$ the permutation can be recovered in time linear in $|L| = |L'|$ [15, 48]. However, the algorithms running time is dominated by the creation of the lists L, L', i.e., by finding those low-weight codewords. The two algorithms that fall into this category are by Leon [48] and more recently by

Beullens [15]. They differentiate in the weight w of the codewords searched for and the amount of weight-w codewords needed for success, detailed explanations of both algorithms and their complexity can be found in [7, 15]. We incorporate the estimation scripts[10] derived in [15] into our estimator. Note that we exchanged the estimate for finding a single weight-w code word with a call to our `SDFqEstimator` module.

We normalize estimation scripts to the elementary operation of $\mathbb{F}_q$ additions and choose as memory unit $\mathbb{F}_q$ elements. We summarize the profile of the current PEP estimator in Table 4.

Permutation Equivalence Estimator

Name	`PEEstimator`
Parameters	(n, k, q, h): length, dimension, field size, hull-dimension
Elementary operation (op)	$\mathbb{F}_q$ addition
Memory unit (el)	$\mathbb{F}_q$ element
Bit complexity factor time	$\log_2 q$
Bit complexity factor memory	$\log_2 q$
No. of algorithms	3
Available modes	Estimate

Table 4: Profile of the PEP estimator.

Deviations from online available scripts. In comparison to the online available scripts, there are slight deviations when using the `CryptographicEstimators` library. This is mostly due to the consideration of more advanced algorithms for finding low-weight code words, the correction of the Lee-Brickell procedure mentioned in Section 4.1 and the normalization to $\mathbb{F}_q$ operations.

Linear Equivalence. The linear equivalence problem differs from the permutation equivalence problem by allowing for monomial transformations. We give a formal definition in the following.

Definition 4.4 (Linear Equivalence Problem). *Let* $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k\times n}$ *be generator matrices of linear codes of dimension* k *and length* n*. The* Linear Equivalence Problem *asks, given* $\mathbf{G}, \mathbf{G}'$*, to find an invertible* $\mathbf{S} \in \mathbb{F}_q^{k\times k}$ *and a monomial matrix* $\mathbf{Q} \in \mathbb{F}_q^{n\times n}$ *such that* $\mathbf{G}' = \mathbf{SGQ}$*.*

Support Splitting Algorithm. The Support Splitting Algorithm can be adapted to linear equivalences. However, this adaptation has to be applied to the closure of the code [59]. The closure of an $\mathbb{F}_q$ linear code is known to be always weakly self-dual, i.e., $h = \min(n-k, k)$, as long as $q > 4$. Therefore, to avoid this line of

[10] The code is accessible at https://github.com/WardBeullens/LESS_Attack.

attacks, all known constructions [6,19] restrict to $q \geq 5$ in the case of LEP, which we adopt in `CryptographicEstimators`. Therefore, no additional parameter h is required in case of LEP.

Low-Weight Codeword-Based Algorithms. The remaining algorithms for solving linear equivalences are adaptations of Leon's and Beullens' algorithm for permutation equivalences and a recent improvement made by Barenghi, Biasse, Persichetti and Santini (BBPS) [7]. More precisely, Leon's algorithm can be applied without any changes.

In the case of Beullens algorithm instead of finding weight-w codewords, the algorithm instead searches for 2-dimensional subcodes with support size w. For finding these subcodes, Beullens adapts an early information set decoding algorithm by Lee-Brickell. The more recent BBPS algorithm restricts the algorithm to subcodes of special structure; that is subcodes of support size w formed by two codewords of weight w'. This allows to find weight-w' codewords via more advanced ISD techniques like the $\mathbb{F}_q$ variant of Stern, given by Peters [52].

We integrated the script computing the complexity of Beullens' algorithm[11] for LEP [15] and the BBPS algorithm[12] given in [7] with some adaptations into our estimator. The adaptations include for example exchanging the estimation for finding weight-w' codewords (needed by the BBPS algorithm) by a call to the `SDFqEstimator` module of the `CryptographicEstimators` library.

As in the case of PEP, for LEP we use additions in $\mathbb{F}_q$ as elementary operations and $\mathbb{F}_q$ elements as memory unit. The state of the estimator for LEP is summarized in Table 5.

Linear Equivalence Estimator	
Name	`LEEstimator`
Parameters	(n, k, q): length, dimension, field size
Elementary operation (op)	$\mathbb{F}_q$ addition
Memory unit (el)	$\mathbb{F}_q$ element
Bit complexity factor time	$\log_2 q$
Bit complexity factor memory	$\log_2 q$
No. of algorithms	3
Available modes	Estimate

Table 5: Profile of the LEP estimator

Deviations from online available scripts. In the case of the estimation of Beullens' algorithm, admissible parameters are found via a small scale experiment. The variance of this experiment can lead to slight deviations in the found optimal parameters and, hence, also in the complexities. However, those deviations already

[11] The code is accessible at https://github.com/WardBeullens/LESS_Attack/

[12] The code is accessible at https://github.com/paolo-santini/LESS_project/.

occur in Beullens' original script. For the estimation of the BBPS algorithm we use a slight approximation for the coupon collector, that is, we approximate the number of times to sample L distinct elements from a list of size N as $L \log L$. This leads to slight deviations for some corner cases. Further deviations might be caused by consideration of different algorithms for the SDP subroutine and normalization to $\mathbb{F}_q$ additions.

4.4 Subcode Equivalence Problem and Permuted Kernel Problem

The subcode equivalence problem (SEP) is a problem closely related to PEP. However, in contrast to PEP, SEP was introduced and proven NP-complete in [12]. Because of the short time since its introduction in 2017, SEP has not been the foundation of many primitives yet. However, recently a connection between SEP and the permuted kernel problem (PKP), which is also proven to be NP-complete and the basis of many cryptographic constructions [18, 20, 39], was observed [56].

More precisely, SEP and PKP are equivalent problems, in the sense that PKP is the dual formulation of SEP. However, usually, when PKP is used in cryptographic constructions very specific instances are considered. This made the relation between PKP and SEP less obvious. For the sake of completeness we define the problem in its general version.

Definition 4.5 (Permuted Kernel Problem). *Let* $\mathbf{A} \in \mathbb{F}_q^{m \times n}$ *and* $\mathbf{V} \in \mathbb{F}_q^{\ell \times n}$. *The* Permuted Kernel Problem *asks, given* $\mathbf{A}, \mathbf{V}$*, to find a permutation* $\mathbf{P} \in \mathbb{F}_q^{n \times n}$ *such that* $\mathbf{A}(\mathbf{VP})^\top = \mathbf{0}$.

The specific PKP instances used in cryptographic constructions are *one-dimensional instances*, which refers to the case of $\ell = 1$, i.e., the case where $\mathbf{V}$ is a vector.

On the other hand, the SEP is very similar to PEP with the main difference that the two given codes have different dimensions and the task is to determine a subcode of the larger one, to which the smaller one is permutation equivalent. More formally we give the following definition.

Definition 4.6 (Subcode Equivalence Problem). *Let* $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ *be a generator matrix of a linear code of dimension* k *and length* n *and* $\mathbf{G}' \in \mathbb{F}_q^{k' \times n}$ *generator matrix of a linear code of dimension* $k' < k$. *The* Subcode Equivalence Problem *asks, given* $\mathbf{G}, \mathbf{G}'$*, to find a full rank matrix* $\mathbf{S} \in \mathbb{F}_q^{k' \times k}$ *and a permutation matrix* $\mathbf{P} \in \mathbb{F}_q^{n \times n}$ *such that* $\mathbf{G}' = \mathbf{SGP}$.

PKP is usually not formulated in a coding perspective. But to see the relation between PKP and SEP consider $\mathbf{A}$ of the PKP to be the parity-check matrix of a code $\mathcal{C}$ of dimension $k = n - m$ and length n. Now finding a permutation, such that $\mathbf{VP}$ lies in the right Kernel of $\mathbf{A}$ means finding a permutation such that the rows of $\mathbf{VP}$ are codewords in $\mathcal{C}$. Now, let $\mathbf{G}_\mathcal{C}$ be the generator matrix of $\mathcal{C}$, then the permutation $\mathbf{P}$ solves the subcode equivalence problem defined by $\mathbf{G} = \mathbf{G}_\mathcal{C}$, $\mathbf{G}' = \mathbf{V}$, if $\ell < n - m$. For $\ell > n - m$ the role of $\mathbf{G}$ and $\mathbf{G}'$ are reversed, while

for $\ell = n - m$ PKP (and the corresponding SEP instance) are equivalent to PEP. A formal proof of these equivalence relations is given in [56, Proposition 4].

In turn, any algorithm solving the above formulation of PKP can be used to solve the SEP with parameters $k = \max(n - m, \ell)$ and $k' = \min(n - m, \ell)$. Therefore, keeping this equivalence in mind, the `CryptographicEstimators` library only offer the general formulation of PKP, omitting a separate problem instantiation for SEP.

Algorithms for solving the PKP are mostly of combinatorial nature. Therefore, assume without loss of generality that $\mathbf{A} = (\mathbf{I}_m \mid \mathbf{A}')$ is in systematic form. Now the algorithms consider only a sub-instance, namely the last u rows of $\mathbf{A}$, i.e., $\mathbf{A}_u = (\mathbf{I}_u \mid \mathbf{A}'_u) \in \mathbb{F}_q^{u \times (n-m+u)}$ (omitting the zero columns in front). The algorithms then search for selections of $n - m + u$ columns of $\mathbf{V}$ that lie in the right kernel of $\mathbf{A}_u$, by enumerating possible selections.[13] That means they solve the PKP with respect to the subinstance $\mathbf{A}_u$. In a final step, the algorithms check if such a sub-solution can be extended to a full permutation that solves the original PKP instance.

Initial combinatorial algorithms were proposed by Georgiades [41] and several follwing works [8, 43, 51]. Koussa, Macario-Rat and Patarin [46] redefined those approaches with small modifications which is now known as the KMP algorithm. Recently, Santini, Baldi and Chiaraluce [56] gave an improved algorithm that exchanges how the matrix $\mathbf{A}_u$ is generated. Essentially, the algorithm searches for a u-dimensional subcode of A with small support size $w < n - m + u$, note that for the KMP we have $w = n - m + u$. To accomplish this subcode search, they use the adaptation of the Lee-Brickell ISD used by Beullens in the context of his LEP algorithm (compare to Section 4.3).

The recent treatment by Santini et al. [56] also provides estimation scripts for the KMP algorithm as well as their improvement, which we incorporated into `CryptographicEstimators`. However, the concrete complexity estimates provided, similar to all previous estimates in the literature, is not very consistent, by assigning the same cost to different unit operations. Precisely [56] specifies "Coherently with all the PKP literature, we measure the running time [...] as the number of matrix-matrix multiplications and list operations.".

In order to obtain a rough estimate, in form of a lower bound, on the involved $\mathbb{F}_q$ additions we assign to each "matrix-matrix multiplication or list operation" by default a cost of $n-m$. This means that, in the estimation, we multiply the amount of these operations by the factor of $n - m$. The library allows to customize this scaling factor. For matrix-matrix multiplications this underestimates the effort, while for list elements in the case of $\ell = 1$ this slightly overestimates the effort by a factor of about 2. For larger ℓ, the effort is in both cases underestimated.

We summarize the current profile of the PKP estimator in Table 6.

Deviations from online available scripts. The estimations in the `CryptographicEstimators` library match the original scripts. Deviations

[13] More recent algorithms use a meet-in-the-middle strategy.

Permuted Kernel Estimator	
Name	`PKEstimator`
Parameters	(n, m, q, ℓ): columns, rows, field size, dimension
Elementary operation (op)	$\mathbb{F}_q$ addition
Memory unit (el)	$\mathbb{F}_q$ element
Bit complexity factor time	$\log_2 q$
Bit complexity factor memory	$\log_2 q$
No. of algorithms	3
Available modes	Estimate

Table 6: Profile of the PKP estimator

are only due to normalization and consideration of a wider range of different ISD algorithms in the optimization.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
2. Baldi, M., Barenghi, A., Chiaraluce, F., Pelosi, G., Santini, P.: A finite regime analysis of information set decoding algorithms. Algorithms **12**(10), 209 (2019)
3. Bardet, M., Faugère, J.C., Salvy, B., Spaenlehauer, P.J.: On the complexity of solving quadratic Boolean systems. Journal of Complexity **29**(1), 53–75 (2013). https://doi.org/https://doi.org/10.1016/j.jco.2012.07.001
4. Bardet, M., Otmani, A., Saeed-Taha, M.: Permutation code equivalence is not harder than graph isomorphism when hulls are trivial. In: 2019 IEEE International Symposium on Information Theory (ISIT). pp. 2464–2468. IEEE (2019)
5. Barenghi, A., Biasse, J.F., Ngo, T., Persichetti, E., Santini, P.: Advanced signature functionalities from the code equivalence problem. International Journal of Computer Mathematics: Computer Systems Theory **7**(2), 112–128 (2022)
6. Barenghi, A., Biasse, J.F., Persichetti, E., Santini, P.: LESS-FM: Fine-tuning signatures from the code equivalence problem. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021. pp. 23–43. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-81293-5_2
7. Barenghi, A., Biasse, J.F., Persichetti, E., Santini, P.: On the computational hardness of the code equivalence problem in cryptography. Cryptology ePrint Archive, Report 2022/967 (2022), https://eprint.iacr.org/2022/967
8. Baritaud, T., Campana, M., Chauvaud, P., Gilbert, H.: On the security of the permuted kernel identification scheme. In: Brickell, E.F. (ed.) Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings. Lecture Notes in Computer Science, vol. 740, pp. 305–311. Springer (1992). https://doi.org/10.1007/3-540-48071-4_21, https://doi.org/10.1007/3-540-48071-4_21
9. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_31

10. Bellini, E., Esser, A.: Syndrome decoding estimator (2021), https://github.com/Crypto-TII/syndrome_decoding_estimator
11. Bellini, E., Makarim, R.H., Sanna, C., Verbel, J.A.: An estimator for the hardness of the MQ problem. pp. 323–347. LNCS (2022). https://doi.org/10.1007/978-3-031-17433-9_14
12. Berger, T.P., Gueye, C.T., Klamti, J.B.: A np-complete problem in coding theory with application to code based cryptography. In: Hajji, S.E., Nitaj, A., Souidi, E.M. (eds.) Codes, Cryptology and Information Security - Second International Conference, C2SI 2017, Rabat, Morocco, April 10-12, 2017, Proceedings - In Honor of Claude Carlet. Lecture Notes in Computer Science, vol. 10194, pp. 230–237. Springer (2017). https://doi.org/10.1007/978-3-319-55589-8_15, https://doi.org/10.1007/978-3-319-55589-8_15
13. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. In: Buchmann, J., Ding, J. (eds.) Post-quantum cryptography, second international workshop, PQCRYPTO 2008. pp. 31–46. Springer, Heidelberg (Oct 2008). https://doi.org/10.1007/978-3-540-88403-3_3
14. Bettale, L., Faugère, J., Perret, L.: Hybrid approach for solving multivariate systems over finite fields. J. Mathematical Cryptology **3**(3), 177–197 (2009)
15. Beullens, W.: Not enough LESS: An improved algorithm for solving code equivalence problems over $\mathbb{F}_q$. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 387–403. Springer, Heidelberg (Oct 2020). https://doi.org/10.1007/978-3-030-81652-0_15
16. Beullens, W.: Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 183–211. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45727-3_7
17. Beullens, W.: MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In: AlTawy, R., Hülsing, A. (eds.) SAC 2021. LNCS, vol. 13203, pp. 355–376. Springer, Heidelberg (Sep / Oct 2022). https://doi.org/10.1007/978-3-030-99277-4_17
18. Beullens, W., Faugère, J.C., Koussa, E., Macario-Rat, G., Patarin, J., Perret, L.: PKP-based signature scheme. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) INDOCRYPT 2019. LNCS, vol. 11898, pp. 3–22. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-35423-7_1
19. Biasse, J.F., Micheli, G., Persichetti, E., Santini, P.: LESS is more: Code-based signatures without syndromes. In: Nitaj, A., Youssef, A.M. (eds.) AFRICACRYPT 20. LNCS, vol. 12174, pp. 45–65. Springer, Heidelberg (Jul 2020). https://doi.org/10.1007/978-3-030-51938-4_3
20. Bidoux, L., Gaborit, P.: Shorter signatures from proofs of knowledge for the sd, mq, PKP and RSD problems. CoRR **abs/2204.02915** (2022). https://doi.org/10.48550/arXiv.2204.02915, https://doi.org/10.48550/arXiv.2204.02915
21. Björklund, A., Kaski, P., Williams, R.: Solving systems of polynomial equations over GF(2) by a parity-counting self-reduction. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.) International Colloquium on Automata, Languages and Programming – ICALP 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.26
22. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 633–666. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_22

23. Both, L., May, A.: Optimizing bjmm with nearest neighbors: full decoding in 22/21n and mceliece security. In: WCC workshop on coding and cryptography. vol. 214 (2017)
24. Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security. In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018. pp. 25–46. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-79063-3_2
25. Bouillaguet, C., Chen, H., Cheng, C., Chou, T., Niederhagen, R., Shamir, A., Yang, B.: Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In: Cryptographic Hardware and Embedded Systems, CHES 2010. pp. 203–218 (2010)
26. Chou, T., Niederhagen, R., Persichetti, E., Randrianarisoa, T.H., Reijnders, K., Samardjiska, S., Trimoska, M.: Take your MEDS: Digital signatures from matrix code equivalence. Cryptology ePrint Archive, Report 2022/1559 (2022), https://eprint.iacr.org/2022/1559
27. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. EUROCRYPT 2000, LNCS **1807**, 392–407 (2000)
28. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_27
29. Cox, D., Little, J., O'Shea, D.: Ideals, varieties, and algorithms. an introduction to computational algebraic geometry and commutative algebra (2007), https://link.springer.com/book/10.1007/978-0-387-35651-8
30. Dinur, I.: Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology - EUROCRYPT 2021. pp. 374–403. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_14
31. Dinur, I.: Improved algorithms for solving polynomial systems over GF(2) by multiple parity-counting. In: ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 2550–2564 (2021). https://doi.org/10.1137/1.9781611976465.151
32. Dumer, I.: On minimum distance decoding of linear codes. In: Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory. pp. 50–52 (1991)
33. Esser, A.: Revisiting nearest-neighbor-based information set decoding. Cryptology ePrint Archive, Report 2022/1328 (2022), https://eprint.iacr.org/2022/1328
34. Esser, A., Bellini, E.: Syndrome decoding estimator. In: PKC 2022, Part I. pp. 112–141. LNCS, Springer, Heidelberg (May 2022). https://doi.org/10.1007/978-3-030-97121-2_5
35. Esser, A., May, A., Zweydinger, F.: McEliece needs a break - solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 433–457. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_16
36. Esser, A., Zweydinger, F.: New time-memory trade-offs for subset sum – improving ISD in theory and practice. Cryptology ePrint Archive, Report 2022/1329 (2022), https://eprint.iacr.org/2022/1329
37. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra **139**(1), 61–88 (1999)
38. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation. p. 75–83. ISSAC '02, Association for Computing Machinery, New York, NY, USA (2002)

39. Feneuil, T.: Building MPCitH-based signatures from MQ, MinRank, rank SD and PKP. IACR Cryptol. ePrint Arch. p. 1512 (2022), https://eprint.iacr.org/2022/1512
40. Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 541–572. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_19
41. Georgiades, J.: Some remarks on the security of the identification scheme based on permuted kernels. J. Cryptol. **5**(2), 133–137 (1992). https://doi.org/10.1007/BF00193565, https://doi.org/10.1007/BF00193565
42. Hamdaoui, Y., Sendrier, N.: A non asymptotic analysis of information set decoding. Cryptology ePrint Archive, Report 2013/162 (2013), https://eprint.iacr.org/2013/162
43. Jaulmes, É., Joux, A.: Cryptanalysis of PKP: A new approach. In: Kim, K. (ed.) Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings. Lecture Notes in Computer Science, vol. 1992, pp. 165–172. Springer (2001). https://doi.org/10.1007/3-540-44586-2_12, https://doi.org/10.1007/3-540-44586-2_12
44. Joux, A., Vitse, V.: A crossbred algorithm for solving boolean polynomial systems. In: Kaczorowski, J., Pieprzyk, J., Pomykała, J. (eds.) Number-Theoretic Methods in Cryptology. pp. 3–21. Springer International Publishing, Cham (2018)
45. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced Oil and Vinegar signature schemes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 206–222. Springer, Heidelberg (May 1999). https://doi.org/10.1007/3-540-48910-X_15
46. Koussa, E., Macario-Rat, G., Patarin, J.: On the complexity of the permuted kernel problem. Cryptology ePrint Archive, Report 2019/412 (2019), https://eprint.iacr.org/2019/412
47. Lee, P.J., Brickell, E.F.: An observation on the security of mceliece's public-key cryptosystem. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 275–280. Springer (1988)
48. Leon, J.: Computing automorphism groups of error-correcting codes. IEEE Transactions on Information Theory **28**(3), 496–511 (1982)
49. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_6
50. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 203–228. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_9
51. Patarin, J., Chauvaud, P.: Improved algorithms for the permuted kernel problem. In: Stinson, D.R. (ed.) Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings. Lecture Notes in Computer Science, vol. 773, pp. 391–402. Springer (1993). https://doi.org/10.1007/3-540-48329-2_33, https://doi.org/10.1007/3-540-48329-2_33
52. Peters, C.: Information-set decoding for linear codes over F_q. In: Sendrier, N. (ed.) The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010. pp. 81–94. Springer, Heidelberg (May 2010). https://doi.org/10.1007/978-3-642-12929-2_7

53. Petrank, E., Roth, R.M.: Is code equivalence easy to decide? IEEE Trans. Inf. Theory **43**(5), 1602–1604 (1997). https://doi.org/10.1109/18.623157, https://doi.org/10.1109/18.623157
54. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
55. Reijnders, K., Samardjiska, S., Trimoska, M.: Hardness estimates of the code equivalence problem in the rank metric. Cryptology ePrint Archive, Report 2022/276 (2022), https://eprint.iacr.org/2022/276
56. Santini, P., Baldi, M., Chiaraluce, F.: Computational hardness of the permuted kernel and subcode equivalence problems. Cryptology ePrint Archive, Report 2022/1749 (2022), https://eprint.iacr.org/2022/1749
57. Sendrier, N.: On the dimension of the hull. SIAM Journal on Discrete Mathematics **10**(2), 282–293 (1997)
58. Sendrier, N.: Finding the permutation between equivalent linear codes: The support splitting algorithm. IEEE Transactions on Information Theory **46**(4), 1193–1203 (2000)
59. Sendrier, N., Simos, D.E.: How easy is code equivalence over fq? In: International Workshop on Coding and Cryptography-WCC 2013 (2013)
60. Stern, J.: A method for finding codewords of small weight. In: International Colloquium on Coding Theory and Applications. pp. 106–113. Springer (1988)
61. Thomae, E., Wolf, C.: Solving underdetermined systems of multivariate quadratic equations revisited. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 156–171. Springer, Heidelberg (May 2012). https://doi.org/10.1007/978-3-642-30057-8_10