Concrete Time/Memory Trade-Offs in Generalised Stern's ISD Algorithm

Sreyosi Bhattacharyya and Palash Sarkar

Indian Statistical Institute
203, B.T. Road
Kolkata
India 700108
bhattacharyya.sreyosi@gmail.com, palash@isical.ac.in

Abstract. The first contribution of this work is a generalisation of Stern's information set decoding (ISD) algorithm. Stern's algorithm, a variant of Stern's algorithm due to Dumer, as well as a recent generalisation of Stern's algorithm due to Bernstein and Chou are obtained as special cases of our generalisation. Our second contribution is to introduce the notion of a set of effective time/memory trade-off (TMTO) points for any ISD algorithm for given ranges of values of parameters of the algorithm. Such a set succinctly and uniquely captures the entire landscape of TMTO points with only a minor loss in precision. We further describe a method to compute a set of effective TMTO points. As an application, we compute sets of effective TMTO points for the five variants of the Classic McEliece cryptosystem corresponding to the new algorithm as well as for Stern's, Dumer's and Bernstein and Chou's algorithms. The results show that while Dumer's and Bernstein and Chou's algorithms do not provide any interesting TMTO points beyond what is achieved by Stern's algorithm, the new generalisation that we propose provide about twice the number of effective TMTO points that is obtained from Stern's algorithm. Consequences of the obtained TMTO points to the classification of the variants of Classic McEliece in appropriate NIST categories are discussed.
Keywords: information set decoding, code based cryptography, Classic McEliece cryptosystem, concrete security, time/memory trade-off.

1 Introduction

Code based cryptography [20] is one of the approaches to building post-quantum secure cryptosystems. There are three code-based cryptosystems, namely BIKE [1], Classic McEliece [5] and HQC [21], in Round 4 of the NIST PQC competition [24].

The best known attack against general code-based cryptosystems is based on information set decoding (ISD). The first ISD algorithm was proposed by Prange [25] and since then there have been many important developments. The early ISD algorithms by Prange [25], Lee and Brickell [16] and Leon [17] do not require any memory beyond what is needed to store the input.

A cornerstone of ISD algorithms is the algorithm by Stern [27] which introduced a meet-in-the-middle approach to ISD algorithms. The minimum time complexity of Stern's algorithm, however, is achieved by using a large amount of memory. In fact,

Section 8.2 of the specification of Classic McEliece (available from [5]) mentions the following. "A closer look shows that the attack in [11] is bottlenecked by random access to a huge array (much larger than the public key being attacked)." The reference [11] in the quotation is [6] which introduced certain important practical efficiency improvements to Stern's algorithm. A key point to note in the above criticism of Stern's algorithm is that the memory requirement is much larger than the size of the public key of the cryptosystem being attacked.

Our first contribution is to introduce a generalisation of Stern's algorithm. Stern's algorithm as well as a variant of Stern's algorithm due to Dumer [9] (and later by Finiasz and Sendrier [13]) are obtained as special cases of the generalisation. Further, a recent generalisation of Stern's algorithm due to Bernstein and Chou [4] is also obtained as a special case. A justification for focusing on Stern's algorithm is that it is a watershed in the literature on ISD algorithms. While later work led to more advanced algorithms, the community still continues to use Stern's algorithm as the baseline ISD algorithm to evaluate code based cryptosystems. This is exemplified by the discussion on the PQC forum available at [14]. We also briefly sketch how our generalisation can be further extended to obtain a unified algorithm from which Prange's, Lee and Brickell's, Leon's as well as Stern's, Dumer's, and Bernstein and Chou's algorithms are obtained as special cases.

The basic idea of our generalisation of Stern's algorithm is the following. In Stern's algorithm, there is an enumeration step which builds a list. Storing this list requires a large amount of memory. Our generalisation allows the list to be significantly smaller by ignoring some elements which would otherwise be stored. This reduces the storage requirement of the algorithm. On the flip side, it also reduces the success probability, thus requiring more iterations. So the generalisation does not improve the runtime of Stern's algorithm. Rather it provides a larger number of time/memory trade-off (TMTO) points than what would be achieved by Stern's algorithm.

Varying the parameters of an ISD algorithm provides a large number of TMTO points. It is quite difficult to directly analyse the entire set of all TMTO points. Our second contribution tackles this issue. We introduce the notion of a set of effective TMTO points of an ISD algorithm with respect to a range of values of the parameters of the algorithm. Such a set succinctly and uniquely captures the entire TMTO landscape at only a minor loss in precision. Further, we describe a method to compute a set of effective TMTO points for any ISD algorithm.

As an application, we have obtained sets of effective TMTO points corresponding to the five variants of the Classic McEliece cryptosystem for the generalisation of Stern's algorithm that we propose as well as for Stern's, Dumer's and Bernstein and Chou's algorithms. The results show that Dumer's and Bernstein and Chou's algorithms do not provide any interesting TMTO points beyond what is achieved by Stern's algorithm. On the other hand, the sets of effective TMTO points obtained from the generalisation that we introduce are about twice the sizes of the corresponding sets of effective TMTO points obtained from Stern's algorithm. Further, in each case, the set of effective TMTO points obtained from the new generalised algorithm essentially subsumes the corresponding set obtained from Stern's algorithm. In particular, there are certain TMTO points which are achieved by the new generalised algorithm, but not by either of

Stern's, Dumer's and Bernstein and Chou's algorithms. The TMTO points themselves are quite interesting. For example, these points show that in certain cases, by letting the time estimates increase by factors which are at most 2, it is possible to reduce the memory requirements by factors of about 2^8 to 2^{10}.

By obtaining sets of effective TMTO points, we are able to address the question of what time complexity can be achieved without requiring memory much larger than the public key size. For `mceliece-4608-096`, the size of the public key is about 2^{22} and the NIST target is 2^{207} gates. For this variant, it is possible to obtain TMTO points having time complexities less than 2^{207} with memory requirement only about 1.5 times the size of the public key for both the cases of constant memory access cost and logarithmic memory access cost. For `mceliece-6688-128` and `mceliece-6960-119`, the sizes of the public key are also about 2^{22} bits and the NIST target for both is 2^{272} gates. With constant memory access cost, time complexities of about 2^{265} can be obtained for both these variants. The corresponding memory complexities are $2^{57.97}$ and $2^{67.37}$ respectively which are much larger than the size of the public key. By choosing values of the parameters appropriately, we find that it is possible to obtain time complexities slightly less than 2^{272} while keeping the memory complexities to be less than 2^{30}. While the memory complexities are still larger than the size 2^{22} of the public key, they are not too large. If, on the other hand, logarithmic cost of memory access is incorporated into the time estimates, then with memory restricted to be less than 2^{30} bits, the minimum time estimates for both the variants turn out to be more than the target value of 2^{272} by factors which are less than 8. For the other two variants of the Classic McEliece cryptosystem, we do not obtain any TMTO point, even with constant memory access cost, whose time complexity is less than what is required for NIST classification.

1.1 Previous and related works. A number of works [18,3,19,8,11] described advanced ISD algorithms with improved asymptotic time complexities. The advanced ISD algorithms require much more memory compared to Stern's algorithm. It has been proved in [28] that for codes which are relevant to the McEliece cryptosystem, all known ISD algorithms have essentially the same asymptotic complexity as that of Prange's algorithm.

TMTO for ISD algorithms has not received much attention in the literature. A recent work on this topic is [12] which mentions that "there has been very limited work on time-memory trade-offs for ISD algorithms." The work [12] considered TMTO with respect to the MMT [18] algorithm and for Classic McEliece, reported time complexities with an upper bound (2^{60} or 2^{80}) on the memory complexities. Considering the time estimates for memory at most 2^{60} bits and comparing with the time estimates of the (generalised) Stern's algorithm also with memory at most 2^{60} bits, we find that the MMT algorithm is faster than Stern's algorithm by factors which are less than 8. We note, though, that the gap could be somewhat wider if certain techniques for improving practical efficiency are incorporated into the method used in [12] for obtaining the time estimates.

A systematic work to assess the concrete security of various code based cryptosystems against a number of important ISD algorithm was reported in [2]. A work along the same line and the corresponding code was provided in [10]. A recent work [4] provides a rigorous approach to obtaining time complexity estimates by comprehensively automating the effects of various practical improvements as well as hidden costs. The

works [2,10,4] focus on obtaining the minimum time complexities achievable by the different ISD algorithms. Unlike our approach, they do not provide any method to analyse the entire landscape of all TMTO points.

2 Preliminaries

Let $\mathbb{F}_2$ denote the finite field of two elements. The cardinality of a finite set S will be denoted as $\#S$. For a positive integer k, the set $\{1,\ldots,k\}$ will be denoted as $[k]$.

Vectors will be considered to be row vectors and will be denoted by bold lower case letters. Matrices will be denoted by bold upper case letters. By $\mathbf{M}^\top$ we will denote the transpose of the matrix $\mathbf{M}$. The identity matrix of order m will be denoted as $\mathbf{I}_m$. If $\mathbf{M}_1$ and $\mathbf{M}_2$ are two matrices having the same number of rows, then by $[\mathbf{M}_1|\mathbf{M}_2]$ we will denote the matrix obtained by juxtaposing $\mathbf{M}_1$ and $\mathbf{M}_2$. For a positive integer k, $\mathbf{0}_k$ and $\mathbf{1}_k$ will denote the all-zero and all-one vectors of length k respectively.

Vectors with elements from $\mathbb{F}_2$ will also be called binary strings. For a binary string $\mathbf{x}$, by $\mathsf{wt}(\mathbf{x})$ we will denote the number of positions where $\mathbf{x}$ is 1, i.e. for $\mathbf{x}=(x_1,\ldots,x_m)$, $\mathsf{wt}(\mathbf{x})=\#\{i : x_i=1\}$. For a subset S of $[k]$, by $\chi(S)$ we will denote the k-bit string $\mathbf{x}=(x_1,\ldots,x_k)$ such that $x_i=1$ if and only if $i\in S$.

Let n and k be positive integers such that $k<n$. We will consider codes over $\mathbb{F}_2$. A linear code $\mathscr{C}$ is a subspace of dimension k of $\mathbb{F}_2^n$. Elements of $\mathscr{C}$ are called codewords. A basis for $\mathscr{C}$ is given by a matrix $\mathbf{G}_0$ of order $k\times n$ and so $\mathscr{C}=\{\mathbf{x}\mathbf{G}_0 : \mathbf{x}\in\mathbb{F}_2^k\}$. Such a matrix $\mathbf{G}_0$ is called a generator matrix for $\mathscr{C}$. The null space of $\mathbf{G}_0$ has dimension $r=n-k$. Let $\mathbf{H}_0$ be a matrix of order $r\times n$ such that the rows of $\mathbf{H}_0^\top$ form a basis for the null space of $\mathbf{G}_0$, i.e. $\mathbf{G}_0\mathbf{H}_0^\top=\mathbf{0}$. Such a matrix $\mathbf{H}_0$ is called a parity check matrix for $\mathscr{C}$. For any codeword $\mathbf{y}\in\mathscr{C}$, we have $\mathbf{H}_0\mathbf{y}^\top=\mathbf{0}$.

Let $w<n$ be a positive integer and $\mathbf{e}\in\mathbb{F}_2^n$ be a vector of weight w. Let $\mathbf{y}\in\mathscr{C}$ be a codeword and define $\mathbf{z}=\mathbf{y}+\mathbf{e}$. The vector $\mathbf{e}$ is called an error vector. Note that $\mathbf{s}^\top=\mathbf{H}_0\mathbf{z}^\top=\mathbf{H}_0(\mathbf{y}^\top+\mathbf{e}^\top)=\mathbf{H}_0\mathbf{e}^\top$, since $\mathbf{H}_0\mathbf{y}^\top=\mathbf{0}$. The vector $\mathbf{s}\in\mathbb{F}_2^r$ is called a syndrome.

The computational problem to be solved is the following.

Definition 1 (Syndrome decoding problem (SDP)). *Let n, k, r and w be positive integers such that $k,w<n$ and $r=n-k$. Let $\mathbf{H}_0$ be a parity check matrix for a linear code $\mathscr{C}\subseteq\mathbb{F}_2^n$ of dimension k. Given a syndrome $\mathbf{s}_0\in\mathbb{F}_2^r$, the goal is to find an error vector $\mathbf{e}\in\mathbb{F}_2^n$ of weight w such that $\mathbf{H}_0\mathbf{e}^\top=\mathbf{s}_0^\top$. The triplet $(\mathbf{H}_0,\mathbf{s}_0,w)$ is an instance of the SDP.*

2.1 ISD algorithms from Prange to Stern. The first ISD algorithm which can be used to solve SDP was proposed by Prange [25]. Let $(\mathbf{H}_0,\mathbf{s}_0,w)$ be an instance of SDP. The requirement is to obtain a vector $\mathbf{e}_0\in\mathbb{F}_2^n$ of weight w such that $\mathbf{H}_0\mathbf{e}_0^\top=\mathbf{s}_0^\top$. The algorithm proceeds as follows. Choose a random permutation matrix $\mathbf{P}$ of order $n\times n$ and apply Gaussian elimination using row operations to the augmented matrix $[\mathbf{H}_0\mathbf{P}\,|\,\mathbf{s}_0^\top]$ to obtain a matrix $[\mathbf{H}\,|\,\mathbf{s}^\top]$, where $\mathbf{H}=\mathbf{U}\mathbf{H}_0\mathbf{P}$ is such that $\mathbf{H}$ can be written as $\mathbf{H}=[\mathbf{A}\,|\,\mathbf{I}_r]$ and $\mathbf{s}^\top=\mathbf{U}\mathbf{s}^\top$. Here $\mathbf{U}$ is the invertible matrix of order $r\times r$ which corresponds to the sequence of row operations. (This may not always be possible; see Remark 2 in Section 3.) Note that $\mathbf{A}$ is a matrix of order $r\times k$.

To solve SDP on instance $(\mathbf{H}_0, \mathbf{s}_0, w)$, it is sufficient to obtain a vector $\mathbf{e} \in \mathbb{F}_2^n$ of weight w such that $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$, since in this case $\mathbf{e}_0^\top = \mathbf{P}\mathbf{e}^\top$ satisfies the relation $\mathbf{H}_0\mathbf{e}_0^\top = \mathbf{s}_0^\top$.

The structure of the algorithm is the following. In each iteration, it chooses an independent and uniform random permutation matrix $\mathbf{P}$ and obtains $\mathbf{H}$ and $\mathbf{s}$ as mentioned above. Then it checks the weight of $\mathbf{s}$ and returns $\mathbf{e} = [\mathbf{0}_k \,|\, \mathbf{s}]$ if the weight is equal to w. (Note: $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$.) Iterations are repeated until a solution is obtained.

Conceptually, each iteration has two steps, namely the linear algebra step and the search step. The linear algebra step yields $[\mathbf{H} \,|\, \mathbf{s}^\top]$, while the search step looks for a solution using $\mathbf{H}$ and $\mathbf{s}$. The search step in Prange's algorithm is trivial and amounts to checking the weight of $\mathbf{s}$. Subsequent algorithms introduced more sophisticated ideas to perform the search step.

Lee and Brickell [16] described an algorithm which can be considered to be a modification of Prange's algorithm. In each iteration, the linear algebra step remains the same as in Prange's algorithm, i.e. $[\mathbf{H} \,|\, \mathbf{s}^\top]$, where $\mathbf{H} = [\mathbf{A} \,|\, \mathbf{I}_r]$ is obtained as in Prange's algorithm. The search step in the Lee-Brickell algorithm uses a parameter p which is a positive integer satisfying $p \leq w$. This step proceeds over all possible subsets of $[k]$ of size $p/2$. For each such p-element subset S, let $\mathbf{e}_1 = \chi(S)$ and $\mathbf{x} = \mathbf{A}\mathbf{e}_1^\top$. If $\mathsf{wt}(\mathbf{x}+\mathbf{s}) = w - p$, then $\mathbf{e} = [\mathbf{e}_1 \,|\, \mathbf{s}]$ satisfies $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$.

Leon [17] put forward an algorithm which can be considered to be a further modification of Prange's algorithm. Apart from the parameter p in Lee and Brickell's algorithm, Leon's algorithm introduced an additional parameter ℓ with $1 \leq \ell \leq r$. The linear algebra step yielding $[\mathbf{H} \,|\, \mathbf{s}^\top]$ remains the same as in Prange's algorithm. In Leon's algorithm, $\mathbf{H}$ and $\mathbf{s}$ are written as

$$\mathbf{H} = \begin{array}{|c|c|c|} \hline \mathbf{A}_{\ell\times k} & \mathbf{I}_\ell & \mathbf{0}_{\ell\times(r-\ell)} \\ \hline \mathbf{B}_{(r-\ell)\times k} & \mathbf{0}_{(r-\ell)\times\ell} & \mathbf{I}_{r-\ell} \\ \hline \end{array}, \qquad \mathbf{s}^\top = \begin{array}{|c|} \hline \mathbf{u}^\top \\ \hline \mathbf{v}^\top \\ \hline \end{array} \tag{1}$$

The search step uses the parameter p as in Lee and Brickell's algorithm and proceeds over all possible subsets of size of $[k]$ of size p. For each p-element subset S, let $\mathbf{e}_1 = \chi(S)$ and $\mathbf{x} = \mathbf{A}\mathbf{e}_1^\top$. (Note that in this case $\mathbf{A}$ is an $\ell \times k$ matrix and not an $r \times k$ matrix as in Lee and Brickell's algorithm.) If $\mathbf{x} = \mathbf{u}$, then let $\mathbf{y} = \mathbf{B}_1\mathbf{e}_1^T$; if $\mathsf{wt}(\mathbf{y}+\mathbf{v}) = w - p$, then $\mathbf{e} = [\mathbf{e}_1 \,|\, \mathbf{0}^\ell \,|\, \mathbf{v}]$ satisfies $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$.

Stern [27] introduced the idea of using the meet-in-the-middle technique to perform the search step. The algorithm uses the two parameters ℓ and p. For simplicity of the basic description, assume that p and also k are even. The output $[\mathbf{H} \,|\, \mathbf{s}^\top]$ of the linear algebra step is written as in (1). Next write

$$\mathbf{A} = [\mathbf{A}_1 | \mathbf{A}_2] \text{ and } \mathbf{B} = [\mathbf{B}_1 | \mathbf{B}_2], \tag{2}$$

where $\mathbf{A}_1$, $\mathbf{A}_2$ are $\ell \times k/2$ matrices and $\mathbf{B}_1$, $\mathbf{B}_2$ are $(r-\ell) \times k/2$ matrices.

The search step has two phases. In the first phase, all possible subsets of $[k/2]$ of size $p/2$ are considered. For each such subset S, let $\mathbf{e}_1 = \chi(S)$ and $\mathbf{a}_1 = \mathbf{A}_1\mathbf{e}_1^\top$. All such pairs

$(\mathbf{a}_1, S)$ are stored in a list $\mathscr{L}$ and $\mathscr{L}$ is indexed on the first components of the entries. In the second phase, again all possible subsets of $[k/2]$ of size $p/2$ are considered. For each such subset T, let $\mathbf{e}_2 = \chi(T)$ and $\mathbf{a}_2 = \mathbf{A}_2\mathbf{e}_2^\top$. Next, for each such pair $(\mathbf{a}_2, T)$, find all entries $(\mathbf{a}_1, S)$ in $\mathscr{L}$ such that $\mathbf{a}_1 + \mathbf{u} = \mathbf{a}_2$. Let $\mathbf{b}_1 = \mathbf{B}_1\mathbf{e}_1^\top$ and $\mathbf{b}_2 = \mathbf{B}_2\mathbf{e}_2^\top$. If $\mathsf{wt}(\mathbf{y}_1 + \mathbf{y}_2 + \mathbf{v}) = w - p$, then $\mathbf{e} = [\mathbf{e}_1 \,|\, \mathbf{e}_2 \,|\, \mathbf{0}_\ell \,|\, \mathbf{v}]$ satisfies $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$.

3 A generalisation of Stern's ISD algorithm

We introduce a generalisation of Stern's algorithm. The generalisation is obtained through the use of two parameters λ and δ in addition to the parameters ℓ and p used in Stern's algorithm. The maximum possible ranges of values of the parameters are as follows.

$$0 \le \ell \le r, \quad -(k-2) \le \lambda \le \ell, \quad 0 \le p \le \min(k+\lambda, w), \quad \delta \in (0,1]. \tag{3}$$

Remark 1. A generalisation of Stern's algorithm (called isd1) has been given by Bernstein and Chou [4, Section 4.8] through the use of a parameter z, which is related to the parameter λ that we use by $z + \lambda = \ell$. The only values of z mentioned in [4] are $z = 0$ and $z = \ell$. It is not clear whether [4] allows z to be greater than ℓ (corresponding to λ less than 0). Note that due to the use of the additional parameter δ, our generalisation of Stern's algorithm subsumes the generalisation in [4]. As we will see later (Remark 7 in Section 5), it is the parameter δ that turns out to determine the non-triviality of the generalisation that we propose.

The basic structure of the algorithm is shown in Algorithm 1. It consists of a linear algebra step and a search step as in Stern's algorithm. These two steps are explained below. The input to the algorithm is $(\mathbf{H}_0, \mathbf{s}_0, w)$ and the requirement is to find a vector $\mathbf{e}_0 \in \mathbb{F}_2^n$ such that $\mathbf{H}_0\mathbf{e}_0^\top = \mathbf{s}_0^\top$.

Algorithm 1: A general formulation of Stern's ISD algorithm.

Input: $(\mathbf{H}_0, \mathbf{s}_0, w)$
Output: $\mathbf{e}_0$ such that $\mathbf{H}_0\mathbf{e_0}^\top = \mathbf{s}_0^\top$ and $\mathsf{wt}(\mathbf{e}_0) = w$.

```
while true do
    Choose a random permutation matrix P of order n × n
    (A1, A2, B1, B2, u, v) ← LinAlg(H0, P, s0, ℓ, λ)
    (flg, e) ← Srch(A1, A2, B1, B2, u, v, w, p, δ)
    if flg = yes then
        return Pe^⊤
```

Linear algebra Apply Gaussian elimination using row operations to the augmented matrix $[\mathbf{H}_0\mathbf{P} \,|\, \mathbf{s}_0^\top]$ to obtain a matrix $[\mathbf{H} \,|\, \mathbf{s}^\top]$, where $\mathbf{H} = \mathbf{U}\mathbf{H}_0\mathbf{P}$ and $\mathbf{s}^\top = \mathbf{U}\mathbf{s}^\top$ are of the

following forms

$$\mathbf{H} = \begin{array}{|c|c|c|} \hline \mathbf{A}_{\ell\times(k+\lambda)} & \mathbf{C}_{\ell\times(\ell-\lambda)} & \mathbf{0}_{\ell\times(r-\ell)} \\ \hline \mathbf{B}_{(r-\ell)\times(k+\lambda)} & \mathbf{D}_{(r-\ell)\times(\ell-\lambda)} & \mathbf{I}_{r-\ell} \\ \hline \end{array}, \quad \mathbf{s}^\top = \begin{array}{|c|} \hline \mathbf{u}^\top \\ \hline \mathbf{v}^\top \\ \hline \end{array} \tag{4}$$

Here $\mathbf{U}$ is the invertible matrix of order $r \times r$ which corresponds to the sequence of row operations, $\mathbf{u} \in \mathbb{F}_2^\ell$ and $\mathbf{v} \in \mathbb{F}_2^{r-\ell}$.

Remark 2. Consider the event E_1 that $\mathbf{H}_0\mathbf{P}$ can be reduced to the form shown in (4), i.e. the bottom right sub-matrix of $\mathbf{H}_0\mathbf{P}$ of order $(r-\ell)\times(r-\ell)$ is invertible. So E_1 is the event that the linear algebra step succeeds, and let π_1 be the probability of E_1. Under the heuristic assumption that the entries of $\mathbf{H}_0$ are independent and uniform random bits, $\pi_1 = \prod_{i=1}^{r-\ell}(1-2^{-i})$, which is about 0.288 for large enough values of $r-\ell$.

Write

$$\mathbf{A} = [\mathbf{A}_1|\mathbf{A}_2] \text{ and } \mathbf{B} = [\mathbf{B}_1|\mathbf{B}_2], \tag{5}$$

where $\mathbf{A}_1$ (resp. $\mathbf{A}_2$) is an $\ell \times \lfloor (k+\lambda)/2 \rfloor$ (resp. $\ell \times \lceil (k+\lambda)/2 \rceil$) matrix, and $\mathbf{B}_1$ (resp. $\mathbf{B}_2$) is an $(r-\ell)\times\lfloor (k+\lambda)/2 \rfloor$ (resp. $(r-\ell)\times\lceil (k+\lambda)/2 \rceil$) matrix. The call $\mathsf{LinAlg}(\mathbf{H}_0, \mathbf{P}, \mathbf{s}_0, \ell, \lambda)$ in Algorithm 1 returns $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{u}, \mathbf{v})$.

Search. As explained in Section 2.1, it is sufficient to obtain a vector $\mathbf{e} \in \mathbb{F}_2^n$ such that $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$. The search step looks for such an $\mathbf{e}$. The parameter δ is used in the search step. As in Stern's algorithm, the search step is done in two parts. The first part prepares a list $\mathscr{L}$ and the second part searches for a collision. The difference with Stern's algorithm is that instead of storing all the $\binom{\lfloor (k+\lambda)/2 \rfloor}{\lfloor p/2 \rfloor}$ ℓ-bit strings arising from considering all $\lfloor p/2 \rfloor$ possible combinations of the $\lfloor (k+\lambda)/2 \rfloor$ columns of $\mathbf{A}_1$, only $\binom{\lfloor (k+\lambda)/2 \rfloor}{\lfloor p/2 \rfloor}^\delta$ of such combinations are considered and the corresponding ℓ-bit strings stored in $\mathscr{L}$. The second part of the search step proceeds more or less in the same manner as that of Stern's algorithm. The complete search algorithm is shown in Algorithm 2. Note that the list $\mathscr{L}$ is stored indexed on the first component of its entries.

Remark 3. From the description of the algorithm, we note that for the error vector $\mathbf{e}$ returned by the algorithm, the matrix vector product $\mathbf{H}\mathbf{e}^\top$ is of the following form.

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{C} & \mathbf{0}_{\ell\times(r-\ell)} \\ \mathbf{B}_1 & \mathbf{B}_2 & \mathbf{D} & \mathbf{I}_{(r-\ell)} \end{bmatrix} \begin{bmatrix} \mathbf{e}_1^\top \\ \mathbf{e}_2^\top \\ \mathbf{0}_{\ell-\lambda}^\top \\ \mathbf{e}_3^\top \end{bmatrix}. \tag{6}$$

Note that there are $\ell - \lambda$ positions where $\mathbf{e}$ has the value 0.

Correctness. It is easy to check that any solution returned by Algorithm 1 is correct.

Algorithm 2: The Srch procedure.

Input: $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{u}, \mathbf{v}, w, p, \delta)$ (see (4) and (5))
Output: $(\mathsf{yes}, \mathbf{e})$ such that $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$ and $\mathsf{wt}(\mathbf{e}) = w$, or $(\mathsf{no}, \perp)$.

$c \leftarrow k + \lambda$; $L_1 \leftarrow \binom{\lfloor c/2 \rfloor}{\lfloor p/2 \rfloor}$
$\mathscr{L} \leftarrow ()$; $i = 0$
for $S \subseteq [\lfloor c/2 \rfloor]$ *with* $\#S = \lfloor p/2 \rfloor$ **do**
  if $i < \lceil L_1^\delta \rceil$ **then**
    $\mathbf{e}_1 = \chi(S)$; $\mathbf{a}_1^\top = \mathbf{A}_1 \mathbf{e}_1^\top$; $\mathbf{b}_1^\top = \mathbf{B}_1 \mathbf{e}_1^\top$
    $\mathscr{L} \leftarrow \mathscr{L} \cup \{(\mathbf{a}_1, \mathbf{b}_1, S)\}$
    $i \leftarrow i + 1$
  else
    break
for $T \subseteq [\lceil c/2 \rceil]$ *with* $\#T = \lceil p/2 \rceil$ **do**
  $\mathbf{e}_2 = \chi(T)$; $\mathbf{a}_2^\top = \mathbf{A}_2 \mathbf{e}_2^\top$; $\mathbf{b}_2^\top = \mathbf{B}_2 \mathbf{e}_2^\top$
  for *all* $(\mathbf{a}_1, \mathbf{b}_1, S) \in \mathscr{L}$ *such that* $\mathbf{a}_1 + \mathbf{u} = \mathbf{a}_2$ **do**
    if $\mathsf{wt}(\mathbf{b}_1 + \mathbf{b}_2 + \mathbf{v}) = w - p$ **then**
      $\mathbf{e}_1 = \chi(S)$; $\mathbf{e}_3 = \mathbf{b}_1 + \mathbf{b}_2 + \mathbf{v}$
      Return $(\mathsf{yes}, [\mathbf{e}_1 | \mathbf{e}_2 | \mathbf{0}_{\ell - \lambda} | \mathbf{e}_3])$
Return $(\mathsf{no}, \perp)$

Special cases. If we take $\delta = 1$, then we obtain the generalisation of Stern's algorithm by Bernstein and Chou [4] (see Remark 1). If we take $\delta = 1$ and $\lambda = 0$, then we essentially get back Stern's algorithm, and if we take $\delta = 1$ and $\lambda = \ell$, then we essentially obtain Dumer's algorithm.

3.1 Further generalisation. The matrix $\mathbf{A}$ has been divided into $\mathbf{A}_1$ and $\mathbf{A}_2$, where $\mathbf{A}_1$ (resp. $\mathbf{A}_2$) has $\lfloor (k+\lambda)/2 \rfloor$ (resp. $\lceil (k+\lambda)/2 \rceil$) columns. We may instead let $\mathbf{A}_1$ and $\mathbf{A}_2$ to have κ_1 and κ_2 columns respectively, where κ_1 and κ_2 are two parameters which are non-negative integers satisfying $\kappa_1 + \kappa_2 = k + \lambda$. Correspondingly, we divide $\mathbf{B}$ into matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ having κ_1 and κ_2 columns respectively. Let p_1 and p_2 be two additional parameters which are non-negative integers such that $p_1 + p_2 = p$.

The generalisation is the following. The first part of the search step will consider column combinations of $\mathbf{A}_1$ taken p_1 at a time and the second part will consider column combinations of $\mathbf{A}_2$ taken p_2 at a time. The description of the search step given in Algorithm 2 can be easily modified to give effect to this generalisation: replace $\lfloor c/2 \rfloor$ by κ_1, $\lceil c/2 \rceil$ by κ_2, $\lfloor p/2 \rfloor$ by p_1 and $\lceil p/2 \rceil$ by p_2.

With the above generalisation, we obtain a *unified* algorithm which provides as special cases all the algorithms from Prange's to Dumer's. It is clear that the generalised Stern's algorithm is obtained by taking $\kappa_1 = \lfloor (k+\lambda)/2 \rfloor$, $\kappa_2 = \lceil (k+\lambda)/2 \rceil$, $p_1 = \lfloor p/2 \rfloor$ and $p_2 = \lceil p/2 \rceil$. Prange's algorithm is obtained by setting $\ell = \lambda = \kappa_1 = p = p_1 = p_2 = 0$, $\delta = 1$ and $\kappa_2 = k$; Lee and Brickell's algorithm is obtained by setting $\ell = \lambda = \kappa_1 = p_1 = 0$, $\delta = 1$, $\kappa_2 = k$ and $p_2 = p$; and Leon's algorithm is obtained by setting $\lambda = \kappa_1 = p_1 = 0$, $\delta = 1$, $\kappa_2 = k$ and $p_2 = p$.

3.2 Practical efficiency improvements. There are known techniques for improving the practical efficiency of Stern's algorithms. Below we briefly describe two of these techniques which we will use in estimating the expected number of bit operations.

Chase's sequence: For each choice S, the Srch procedure obtains $\mathbf{e}_1 = \chi(S)$ and computes $\mathbf{a}_1^\top = \mathbf{A}_1\mathbf{e}^\top$. The latter computation involves adding together $\lfloor p/2 \rfloor$ columns of $\mathbf{A}_1$. This requires a total of $\lfloor p/2 \rfloor - 1$ additions of ℓ-bit vectors. Since L_1 subsets S are considered, the total number of ℓ-bit vector additions is $L_1 \cdot (\lfloor p/2 \rfloor - 1)$. An alternative way to perform the entire computation is to use Chase's sequence (see Section 7.2.1.3 of [15]) as was done in [29,30]. In this technique, the subsets S are generated incrementally, where the next subset is obtained from the present subset by removing one element and including a new one. So from the vector $\mathbf{a}_1$ corresponding to the present subset, the vector $\mathbf{a}_1$ corresponding to the next subset can be obtained using exactly two ℓ-bit vector additions. As a result, the total number of ℓ-bit vector additions required for all the subsets becomes $2 \cdot L_1$. This is an improvement over the naive method if $p > 5$. Similar efficiency improvements are obtained for the computations of $\mathbf{b}_1$, $\mathbf{a}_2$ and $\mathbf{b}_2$. Even though the use of Chase's sequence is advantageous only for $p > 5$, for the sake of obtaining a single expression for the time complexity, we will assume $2 \cdot L_1$ vector additions are required even for $p < 5$, and similarly for the computations of $\mathbf{b}_1$, $\mathbf{a}_2$ and $\mathbf{b}_2$. This makes the time complexity estimates slightly worse for $2 \leq p \leq 5$.

Early abort: The final check for a solution is to compare the weight of $\mathbf{x} = \mathbf{b}_1 + \mathbf{b}_2 + \mathbf{v}$ with $w - p$. Note that the length of $\mathbf{x}$ is $r - \ell$, which in general is substantially greater than $w - p$. This observation forms the basis of the technique of early abort in [6]. Instead of first computing $\mathbf{x}$ and then comparing its weight to $w - p$, it is faster to compute $\mathbf{x}$ incrementally and abort once the weight of the partially computed $\mathbf{x}$ exceeds $w - p$. If $\mathbf{x}$ does not correspond to a solution, it is reasonable to assume that it will behave like a random binary string of length $r - \ell$. So the first $2(w - p + 1)$ positions is likely to have weight $w - p + 1$ and such a vector $\mathbf{x}$ can be discarded without computing the other bits. For most vectors $\mathbf{x}$, this brings down the cost of checking $\mathsf{wt}(\mathbf{x}) = w - p$ from $r - \ell$ bit operations to an expected number $2(w - p + 1)$ of bit operations.

There are several other sophisticated techniques to improve both the linear algebra step and the search step of Stern's algorithm [6,4]. All of these techniques also apply to the generalised Stern's algorithm. For the sake of simplicity we do not include the effect of these techniques in the present analysis. Nonetheless, time estimates for the variants of Classic McEliece obtained from our simple model are quite close to the time estimates obtained using the more detailed techniques of [4]; see Section 5.1.

3.3 Time complexity. We estimate the number of bit operations required by Algorithm 1. The quantity L_1 is defined in Algorithm Srch. Recall that $c = k + \lambda$ and let

$$L_2 = \binom{\lceil c/2 \rceil}{\lceil p/2 \rceil}. \tag{7}$$

Since the algorithm is probabilistic, first we calculate the success probability of the algorithm in a single iteration. We assume that there is one solution $\mathbf{e}_0$ (of weight w) to the ISD instance $(\mathbf{H}_0, \mathbf{s}_0, w)$. By success probability we mean the probability of the event that the algorithm returns this solution. (If there are more than one solutions, then the success probability will be higher.)

We assume that the permutation matrix $\mathbf{P}$ is chosen uniformly and independently in each iteration. So in each iteration, a uniform random permutation is applied to the columns of the parity check matrix $\mathbf{H}_0$. The total number of such permutations is $n!$. Let $\mathscr{P}$ be the set of 'good' permutations, i.e. if any permutation from $\mathscr{P}$ is applied to the columns of $\mathbf{H}_0$, then a solution is obtained in the search step. So the success probability π of the search step of a single iteration is

$$\pi = \frac{\#\mathscr{P}}{n!}. \tag{8}$$

The set $\mathscr{P}$ can be constructed in the following manner. Let $i_1, \ldots, i_w$ be the one-positions of $\mathbf{e}_0$ (i.e. the positions where $\mathbf{e}_0$ is 1). Call the other positions of $\mathbf{e}_0$ to be zero-positions. It is helpful to visualise the construction of the permutations in $\mathscr{P}$ as distributing the one-positions and the zero-positions to the cells of an array of length n. Distribute the first $\lfloor p/2 \rfloor$ one-positions to a subset of the cells $1, \ldots, \lfloor c/2 \rfloor$ of size $\lfloor p/2 \rfloor$ in a manner such that these cells form some subset S in $\mathscr{L}$ (there are $\#\mathscr{L}$ ways to make this distribution); distribute the next $\lceil p/2 \rceil$ one-positions to some subset of the cells $(\lfloor c/2 \rfloor + 1), \ldots, c$ of size $\lceil p/2 \rceil$ (there are L_2 ways to make this distribution); distribute the remaining $w-p$ one-positions to some subset of the cells $(c+1), \ldots, n$ of size $w-p$ (there are $\binom{n-k-\ell}{w-p}$ ways to make this distribution); and then fill the remaining $n-w$ cells with the zero-positions in some particular order. This fixes the positions for the one-positions and the zero-positions in the array. This fixing can be done in

$$\#\mathscr{L} \cdot L_2 \cdot \binom{n-k-\ell}{w-p}$$

ways. Now permute the cells filled with the one-positions among themselves and also permute the cells filled with the zero-positions among themselves, which can be done in $w!(n-w)!$ ways. So the size of $\mathscr{P}$ is

$$\#\mathscr{P} = \#\mathscr{L} \cdot L_2 \cdot \binom{n-k-\ell}{w-p} \cdot w!(n-w)!. \tag{9}$$

Algorithm Srch ensures that the size of $\mathscr{L}$ is $\lceil L_1^{\delta} \rceil$. Using (8) and (9), we have

$$\pi = \frac{\lceil L_1^{\delta} \rceil \cdot L_2 \cdot \binom{n-k-\ell}{w-p}}{\binom{n}{w}}. \tag{10}$$

Let N be the number of iterations required to obtain success. Then N follows a geometric distribution with parameter π. Consequently,

$$\mathbb{E}[N] = \frac{1}{\pi}. \tag{11}$$

Let X_i be the random variable whose value is given by the number of bit operations performed in the i-th iteration. The total number of bit operations is $X_1 + X_2 + \cdots + X_N$. Let T be the expected value of the total number of bit operations. Under the heuristic assumption that the X_i's are independent and identically distributed, and N is independent of the X_i's, by Wald's equation (see Page 300 of [22]),

$$T = \mathbb{E}[X_1 + X_2 + \cdots + X_N] = \mathbb{E}[X_1] \cdot \mathbb{E}[N]. \tag{12}$$

Next we obtain an estimate for $\mathbb{E}[X_1]$, i.e. the expected number of bit operations in each iteration. This has two components, the number of bit operations due to linear algebra step and the number of bit operations due to the search step. Denoting by T_{LA} and T_{SR} the expected number of bit operations required for linear algebra and search steps respectively, we have

$$\mathbb{E}[X_1] = T_{\mathrm{LA}} + T_{\mathrm{SR}}. \tag{13}$$

Using (11), (12) and (13), we obtain

$$T = \frac{1}{\pi} \cdot (T_{\mathrm{LA}} + T_{\mathrm{SR}}) = \frac{\binom{n}{w}}{\lceil L_1^{\delta} \rceil \cdot L_2 \cdot \binom{n-k-\ell}{w-p}} \cdot (T_{\mathrm{LA}} + T_{\mathrm{SR}}). \tag{14}$$

Remark 4. The above analysis ignores the effect of E_1, i.e. the event that the linear algebra step succeeds (see Remark 2). We briefly consider this effect. Let E_2 be the event that the search step succeeds and so $\pi = \Pr[E_2]$. Let M be a random variable whose value is the number of iterations required by Algorithm 1 to achieve success. For $i = 1, \ldots, M$, let T_i be the binary valued random variable which takes the value 1 if and only if E_1 occurs in the i-th step. So $\mathbb{E}[T_i] = \pi_1$. Let $N = \sum_{i=1}^{M} T_i$, i.e. N is the number of times the search step is repeated until success is obtained (i.e. E_2 occurs), and so $\mathbb{E}[N] = 1/\pi$. Hence, by an application of Wald's equation, $\mathbb{E}[M] = (\pi \cdot \pi_1)^{-1}$. Let Y_i and Z_i be random variables whose values are the numbers of bit operations required for the linear algebra and the search step respectively. As above, let X_i be the number of bit operations required in the i-th step, and so $X_i = Y_i + T_i Z_i$. Note that $\mathbb{E}[Y_i] = T_{\mathrm{LA}}$ and $\mathbb{E}[Z_i] = T_{\mathrm{SR}}$. Letting T denote the expected value of the total number of bit operations, we have $T = \mathbb{E}[\sum_{i=1}^{M} X_i] = \mathbb{E}[\sum_{i=1}^{M} Y_i] + \mathbb{E}[\sum_{i=1}^{M} T_i Z_i]$. Applying Wald's equation separately to the two terms and heuristically assuming that T_i and Z_i are independent, we obtain $T = \pi^{-1} \cdot (T_{\mathrm{LA}} \cdot \pi_1^{-1} + T_{\mathrm{SR}})$. Note the difference with the expression for T given by (14). This difference, however, does not cause noticeable difference in the concrete time estimates and so for simplicity, we consider T to be given by (14).

Next we describe estimates of T_{LA} and T_{SR}. If $\lambda \leq 0$, then the last ℓ columns of the matrices $\mathbf{C}$ and $\mathbf{D}$ in (4) are $\mathbf{I}_\ell$ and the all-zero matrices respectively. This corresponds to the row operations required in Stern's algorithm. For $0 < \lambda \leq \ell$, the matrices $\mathbf{C}$ and $\mathbf{D}$ are smaller and the linear algebra step possibly requires less number of bit operations. For simplicity of analysis, we assume that the number of bit operations required for the linear algebra step is equal to that of Stern's algorithm. Following [6], we estimate the expected number of bit operations required for the linear algebra step in Stern's algorithm to be

$$T_{\mathrm{LA}} = \frac{k^2(n-k)(n-k-1)(3n-k)}{4n^2}. \tag{15}$$

In the search step, bit operations are required to compute the quantities $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{b}_1 + \mathbf{b}_2 + \mathbf{v}$. Using the early abort technique (see Section 3.2) requires computing only the first $2(w-p+1)$ bits of the last three quantities. Using Chase's sequence (again see Section 3.2) the computation of all the $\lceil L_1^{\delta} \rceil$ $\mathbf{a}_1$'s and $\mathbf{b}_1$'s require a total of $(2\ell +$

$4(w-p+1))\lceil L_1^{\delta}\rceil$ bit operations. Similarly, using Chase's sequence the computation of all the L_2 $\mathbf{a}_2$'s and $\mathbf{b}_2$'s require a total of $(2\ell+4(w-p+1))L_2$ bit operations. Finally, we consider the number of bit operations in the collision phase. We make the heuristic assumption that the $\mathbf{a}_1$'s that arise due to the different choices of S are independent and uniformly distributed ℓ-bit strings. So for any fixed ℓ-bit string $\mathbf{x}$, the probability it arises as $\mathbf{a}_1$ due to any particular choice of S is equal to $1/2^{\ell}$; consequently, the total number of times $\mathbf{x}$ occurs as a first component in the list $\mathscr{L}$ follows the binomial distribution with parameters $1/2^{\ell}$ and $\#\mathscr{L}$. So the expected number of times $\mathbf{x}$ occurs as a first component in the list $\mathscr{L}$ is $\#\mathscr{L}/2^{\ell} = \lceil L_1^{\delta}\rceil/2^{\ell}$. It then follows that for each of the L_2 $\mathbf{a}_2$'s (which are also ℓ-bit strings) generated in the collision phase, on an average there will be about $\lceil L_1^{\delta}\rceil/2^{\ell}$ $\mathbf{a}_1$'s in $\mathscr{L}$ such that the condition $\mathbf{a}_1+\mathbf{u}=\mathbf{a}_2$ holds. So the computation of $\mathbf{b}_1+\mathbf{b}_2+\mathbf{v}$ has to be done a total of about $L_2\cdot(\lceil L_1^{\delta}\rceil/2^{\ell})$ times. Using the early abort technique, this requires about $L_2\cdot(\lceil L_1^{\delta}\rceil/2^{\ell})\cdot(2(w-p+1))$ bit operations. Putting the calculations together, we have

$$\begin{aligned} T_{\mathrm{SR}} \approx\ & (2\ell+4(w-p+1))\lceil L_1^{\delta}\rceil \\ & +(2\ell+4(w-p+1))L_2+L_2\cdot(\lceil L_1^{\delta}\rceil/2^{\ell})\cdot(2(w-p+1)). \end{aligned} \tag{16}$$

3.4 Memory complexity. An instance of the SDP is a triplet $(\mathbf{H}_0,\mathbf{s}_0,w)$. So any ISD algorithm has to store the matrix $\mathbf{H}_0$ and the syndrome $\mathbf{s}_0$. Let M_{mat} be the number of bits required to store $\mathbf{H}_0$ and $\mathbf{s}_0$. Then

$$M_{\mathrm{mat}} = (n-k)(n+1). \tag{17}$$

The additional memory requirement of Algorithm 1 arises from the memory requirement of Algorithm Srch, which needs to store the list $\mathscr{L}$. The size of $\mathscr{L}$ is $\lceil L_1^{\delta}\rceil$. Each entry of $\mathscr{L}$ is of the form $(\mathbf{a}_1,\mathbf{b}_1,S)$, where $\mathbf{a}_1$ is an ℓ-bit vector, $\mathbf{b}_1$ is an $(r-\ell)$-bit vector and M is a subset of $[\lfloor(k+\lambda)/2\rfloor]$ of size $\lfloor p/2\rfloor$. Since we consider the early abort technique, only the first $2(w-p+1)$ bits of $\mathbf{b}_1$ are stored. Let M_{lst} be the number of bits required to store $\mathscr{L}$. Then

$$M_{\mathrm{lst}} = \lceil L_1^{\delta}\rceil\cdot(\ell+2(w-p+1)+\lfloor p/2\rfloor\cdot\lceil\log_2\lfloor(k+\lambda)/2\rfloor\rceil). \tag{18}$$

The total memory required by Algorithm 1 is M, where

$$M = M_{\mathrm{mat}}+M_{\mathrm{lst}}. \tag{19}$$

3.5 Cost of memory access. If M is large, then accessing memory will require non-negligible time. The logarithmic memory access cost model has been suggested in previous works [2,11]. In this model, the time estimate is increased by a factor which is equal to the logarithm to the base two of the memory estimate. So the time estimate T_{ma} taking logarithmic memory access cost into consideration is given by

$$T_{\mathrm{ma}} = T\cdot\log_2 M. \tag{20}$$

Remark 5. Estimating the cost of memory access using (20) is rather adhoc. For one thing it assumes that each bit operation requires a memory access which is not the case. Secondly, a large fraction of the bit operations are on the matrix $\mathbf{H}$ and not on the list $\mathscr{L}$. Compared to $\mathbf{H}$, the list $\mathscr{L}$ can require much more memory to store. So assigning the same cost of memory access to memory operations on $\mathbf{H}$ and $\mathscr{L}$ may not be justified.

Remark 6. A Boolean circuit model based analysis of time estimates of ISD algorithms has been performed in [4]. Such an analysis inherently incorporates cost of memory access which would otherwise be ignored in an analysis assuming constant time for memory access. Note, however, that incorporating logarithmic memory access cost as in (20) captures memory access cost in a manner which is different from that in [4].

4 A set of effective TMTO points

We describe what we mean by a set of effective TMTO points and a procedure to compute such a set. Our initial description is based on Algorithm 1. Later we mention how the procedure can be applied to any ISD algorithm.

For fixed values of n,k and w, it is possible to evaluate the expressions for T and M given by (14) and (19) respectively for every possible choice of the values of the parameters ℓ, p, λ and δ. This leads to a large number of TMTO points (T,M). For example, using the range of parameters given by (23) in Section 5 leads to more than 3 million TMTO points. There are, however, large clusters of values of T and M. As an example, for the variant `mceliece-3488-064` of Classic McEliece, we have $n = 3488$, $k = 2720$ and $w = 64$. In this case, the minimum value of $\log_2 T$ required by Algorithm 1 is 147.70988 and there are 819 values which are less than 148, the largest of which is 147.99983. Similar clustering also occurs for $\log_2 M$. Based on these observations, we define two time complexities T and T' to be equivalent if $\lceil \log_2 T \rceil = \lceil \log_2 T' \rceil$, and two memory complexities M and M' to be equivalent if $\lceil \log_2 M \rceil = \lceil \log_2 M' \rceil$. Extending to TMTO points, we define two TMTO points (T,M) and (T',M') to be equivalent if $\lceil \log_2 T \rceil = \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil = \lceil \log_2 M' \rceil$. So the time and memory complexities of a TMTO point differ from the respective complexities of an equivalent TMTO point by factors which are less than 2.

For fixed values of n, k and w, let $\mathscr{T}_0$ be the set of all tuples

$$(\lceil \log_2 T \rceil, \lceil \log_2 M \rceil, \ell, p, \lambda, \delta, \log_2 T, \log_2 M) \tag{21}$$

corresponding to a pre-determined range of values of the parameters ℓ, p, λ and δ. The list $\mathscr{T}_0$ captures all the TMTO points arising from the chosen range of values of the parameters. We say that a TMTO point (T,M) is represented in $\mathscr{T}_0$ if $(\log_2 T, \log_2 M)$ occurs as the last two components of some tuple in $\mathscr{T}_0$. Let $\mathscr{V}$ be a non-empty set of TMTO points represented in $\mathscr{T}_0$ satisfying the following two properties.

1. *Minimality:* If $(T,M) \neq (T',M')$ are in $\mathscr{V}$, then $\lceil \log_2 T \rceil \neq \lceil \log_2 T' \rceil$, $\lceil \log_2 M \rceil \neq \lceil \log_2 M' \rceil$, and either $\lceil \log_2 T \rceil > \lceil \log_2 T' \rceil$ or $\lceil \log_2 M \rceil > \lceil \log_2 M' \rceil$.
2. *Completeness:* If (T',M') is represented in $\mathscr{T}_0$, then there is a point (T,M) in $\mathscr{V}$ such that $\lceil \log_2 T \rceil \leq \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil \leq \lceil \log_2 M' \rceil$.

We say that the set $\mathscr{V}$ is a *set of effective TMTO points* with respect to the chosen range of parameters. A consequence of the first condition is that two distinct points in $\mathscr{V}$ are inequivalent. Let

$$\mathsf{CL}(\mathscr{V}) = \{(\lceil \log_2 T \rceil, \lceil \log_2 M \rceil) : (T,M) \in \mathscr{V}\}. \tag{22}$$

Proposition 1. *If $\mathscr{V}$ and $\mathscr{V}'$ are two sets of effective TMTO points for the same ranges of values of the parameters, then* $\mathsf{CL}(\mathscr{V}) = \mathsf{CL}(\mathscr{V}')$.

Proof. Suppose $(\lceil \log_2 T' \rceil, \lceil \log_2 M' \rceil)$ is in $\mathsf{CL}(\mathscr{V}')$ corresponding to some point (T', M') in $\mathscr{V}'$. Since $\mathscr{V}'$ is a set of effective TMTO points, by definition, (T', M') is represented in $\mathscr{T}_0$. Since $\mathscr{V}$ is also a set of effective TMTO points, by completeness of $\mathscr{V}$, there is a point (T, M) in $\mathscr{V}$ which is represented in $\mathscr{T}_0$ satisfying $\lceil \log_2 T \rceil \leq \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil \leq \lceil \log_2 M' \rceil$. Further, $(\lceil \log_2 T \rceil, \lceil \log_2 M \rceil)$ is in $\mathsf{CL}(\mathscr{V})$. Since (T, M) is represented in $\mathscr{T}_0$ and $\mathscr{V}'$ is a set of effective TMTO points, by completeness of $\mathscr{V}'$, there is a point (T'', M'') in $\mathsf{CL}(\mathscr{V}')$ satisfying $\lceil \log_2 T'' \rceil \leq \lceil \log_2 T \rceil$ and $\lceil \log_2 M'' \rceil \leq \lceil \log_2 M \rceil$. So it follows that $\lceil \log_2 T'' \rceil \leq \lceil \log_2 T' \rceil$ and $\lceil \log_2 M'' \rceil \leq \lceil \log_2 M' \rceil$. By minimality of $\mathscr{V}'$, the condition in the previous sentence is only possible if $\lceil \log_2 T'' \rceil = \lceil \log_2 T' \rceil$ and $\lceil \log_2 M'' \rceil = \lceil \log_2 M' \rceil$, which implies that $\lceil \log_2 T \rceil = \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil = \lceil \log_2 M' \rceil$, and so $(\lceil \log_2 T' \rceil, \lceil \log_2 M' \rceil)$ is in $\mathsf{CL}(\mathscr{V})$. This shows that $\mathsf{CL}(\mathscr{V}')$ is a subset of $\mathsf{CL}(\mathscr{V})$. Reversing the argument, we have $\mathsf{CL}(\mathscr{V})$ to be a subset of $\mathsf{CL}(\mathscr{V}')$. □

Proposition 1 shows that a set of effective TMTO points uniquely captures the entire landscape of all TMTO points up to loss of precision by factors which are less than 2. Later we will see examples which show that a set of effective TMTO points can be much smaller than the size of all TMTO points.

We describe a method to compute a set of effective TMTO points. The idea is to progressively process a list of tuples $\mathscr{T}$ which is initially set to be equal to $\mathscr{T}_0$. The following steps are then performed successively on $\mathscr{T}$.

1. *Sorting:* Perform an ascending order sort of the tuples in $\mathscr{T}$, where the usual lexicographic ordering of tuples is assumed, i.e. a tuple is considered to be less than another if for some $i \geq 1$, the first $i-1$ components of the two tuples are equal and the i-th component of the first tuple is less than that of the second.
2. *First filtering:* Perform a filtering on $\mathscr{T}$ to ensure that for any particular value of $\lceil \log_2 T \rceil$, only the first tuple with the given value is retained while all other tuples with the same value of $\lceil \log_2 T \rceil$ are dropped.
3. *Second filtering:* Perform a filtering on $\mathscr{T}$ to ensure that for any particular value of $\lceil \log_2 M \rceil$, only the first tuple with the given value is retained while all other tuples with the same value of $\lceil \log_2 M \rceil$ are dropped.
4. *Pruning:* Discard all tuples from $\mathscr{T}$ starting from the point where $\lceil \log_2 T \rceil$ increases but $\lceil \log_2 M \rceil$ does not decrease.

Let $\mathscr{T}_1$ be the final state of $\mathscr{T}$ and let $\mathscr{U}$ be the set of all TMTO points (T, M) which are represented in $\mathscr{T}_1$. We have the following result.

Proposition 2. *$\mathscr{U}$ is a set of effective TMTO points.*

Proof. Since $\mathscr{T}_1$ is obtained from $\mathscr{T}_0$ by removing tuples, any tuple in $\mathscr{T}_1$ is also in $\mathscr{T}_0$. So any TMTO point in $\mathscr{U}$ is represented in $\mathscr{T}_0$.

Suppose (T, M) and (T', M') are two TMTO points in $\mathscr{U}$. The first and the second filtering steps ensure that $\lceil \log_2 T \rceil \neq \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil \neq \lceil \log_2 M' \rceil$ respectively. The pruning step ensures that if $\lceil \log_2 T \rceil < \lceil \log_2 T' \rceil$, then $\lceil \log_2 M \rceil > \lceil \log_2 M' \rceil$, as otherwise the tuple representing (T', M') would be dropped in the pruning step. This shows the minimality of $\mathscr{U}$.

Suppose (T',M') is a TMTO point represented in $\mathscr{T}_0$. If it is not in $\mathscr{U}$, then the tuple which represents it in $\mathscr{T}_0$ was dropped in one of the two filtering steps or in the pruning step. If it was dropped in the first filtering step, then there is a TMTO point (T,M) in $\mathscr{U}$ such that $\lceil \log_2 T \rceil = \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil \leq \lceil \log_2 M' \rceil$. If it was dropped in the second filtering step, then there is a TMTO point (T,M) in $\mathscr{U}$ such that $\lceil \log_2 T \rceil \leq \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil = \lceil \log_2 M' \rceil$. Finally, if it was dropped in the pruning step, then there is a TMTO point (T,M) in $\mathscr{U}$ such that $\lceil \log_2 T \rceil < \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil < \lceil \log_2 M' \rceil$. This shows the completeness of $\mathscr{U}$. □

A simple SAGE [26] code to compute a set of effective TMTO points for Algorithm 1 for the range of parameters in (23) is given in Appendix A. Setting $\delta = 1$ in the code provides a set of effective TMTO points for the Bernstein-Chou generalisation of Stern's algorithm; setting $\lambda = 0$ and $\delta = 1$ in the code provides a set of effective TMTO points for Stern's algorithm; setting $\lambda = \ell$ and $\delta = 1$ in the code provides a set of effective TMTO points for Dumer's algorithm.

The discussion above for obtaining a set of effective TMTO points was with reference to Algorithm 1. Changing the procedure to any other ISD algorithm is simply to change the parameters and to use the appropriate expressions (or algorithms) to compute the time and memory complexities of the algorithm. This leads to changing (21) and affects the construction of the initial state $\mathscr{T}_0$ of $\mathscr{T}$. The sorting, filtering and pruning steps are applied to $\mathscr{T}$ as described above. A set $\mathscr{U}$ of effective TMTO points (with respect to the chosen range of values of the parameters) can then be defined from the final state $\mathscr{T}_1$ of $\mathscr{T}$ in the same manner as described above.

4.1 Comparison to previous TMTO analysis of ISD algorithms. The TMTO analysis considered in [12] is of the following type. Fix an upper bound M on the memory complexity and then obtain the minimum time required by the algorithm utilising at most M bits of memory. Let us call this a memory bounded approach to TMTO analysis. Such an approach provides less information in comparison to the analysis using the notion of an effective set of TMTO points. We illustrate this using an example. Consider Table 2a which provides certain TMTO points for `mceliece-3488-064` achieved by Algorithm 1 using the notion of effective set of TMTO points. Now consider the memory bounded analysis. Suppose we fix the memory bound M to be 2^{50} and ask what is the minimum time that can be achieved by Algorithm 1 utilising at most 2^{50} bits of memory? From Table 2a the answer is $2^{147.93}$. However, from the table we also find that the time $2^{147.93}$ can be achieved using $2^{44.55}$ bits of memory. Similarly, if we fix M to be 2^{40}, then the memory bounded analysis will provide minimum time estimate of $2^{148.93}$, while the analysis based on effective set of TMTO points will provide the additional information that the time estimate of $2^{148.93}$ can be achieved using $2^{34.63}$ bits of memory. In both the above cases, the actual memory requirements $2^{44.55}$ or $2^{34.63}$ are lower than the upper bounds 2^{50} and 2^{40} respectively. So while the memory bounded analysis obtains the minimum time subject to an upper bound on the memory, it does not yield the actual memory that is required to achieve the minimum time. In contrast, the analysis based on the notion of effective set of TMTO points provides for each time estimate the minimum memory required to achieve the time estimate. More generally, due to the completeness property an effective set of TMTO points captures (up to a minor loss in precision) the entire landscape of TMTO points. In particular, for any TMTO

point (T',M') obtained using a memory bounded analysis, the completeness property assures us that there is a TMTO point (T,M) in an effective set of TMTO points such that $\lceil \log_2 T \rceil \leq \lceil \log_2 T' \rceil$ and $\lceil \log_2 M \rceil \leq \lceil \log_2 M' \rceil$.

5 Application to Classic McEliece

The NIST call for proposals [23] for post-quantum cryptosystems outlines five categories. Of these, Categories 1, 3 and 5 require cryptosystems to be secure under attacks using 2^{143}, 2^{207} and 2^{272} classical gates respectively.

The expected security categories of the five variants are given in Section 7 of the Classic McEliece specification [5]. For the five variants, our abbreviations of the names, the values of n, k and w for the five variants, and their categories are shown in Table 1. Section 2.2.3 of the Classic McEliece specification [5], states that the public key is an $r \times k$ binary matrix. Table 1 shows $\log_2 P$, where $P = r \cdot k$ is the size of the public key in bits. An ISD instance is given by $(\mathbf{H}_0, \mathbf{s}_0, w)$. So M_{mat} bits are required to store $\mathbf{H}_0$ and $\mathbf{s}_0$. The values of $\log_2 M_{\text{mat}}$ for the variants of the Classic McEliece are shown in Table 1.

category	name	n	k	w	$\log_2 P$	$\log_2 M_{\text{mat}}$
1	`mceliece-3488-064` (`m3488`)	3488	2720	64	20.99	21.35
3	`mceliece-4608-096` (`m4608`)	4608	3360	96	22.00	22.46
5	`mceliece-6688-128` (`m6688`)	6688	5024	128	22.00	23.41
	`mceliece-6960-119` (`m6960`)	6960	5413	119	22.00	23.36
	`mceliece-8192-128` (`m8192`)	8192	6528	128	23.37	23.70

Table 1: Parameters for Classic McEliece and the corresponding values of $\log_2 P$ and $\log_2 M_{\text{mat}}$.

For Algorithm 1, instead of using the maximum possible ranges of values of parameters given by (3), we have restricted to the following ranges of values of the parameters. The concrete results obtained using these choices indicate that there are no interesting TMTO points for values of parameters outside these ranges.

$$0 \leq \ell \leq 100, \quad 2 \leq p \leq 30, \quad -\ell \leq \lambda \leq \ell, \quad \delta = 1 - 0.025 \cdot i,\ i = 0, \ldots, 12. \tag{23}$$

The total number of choices of values for the parameters ℓ, p, λ and δ in the above ranges is 3712800. We have used the code in Appendix A to compute the sets of effective TMTO points for the variants of Classic McEliece corresponding to Algorithm 1 for both the cases where memory access cost is taken to be constant and the logarithmic memory access cost model is assumed. We have also computed similar sets of effective TMTO points corresponding to Stern's (i.e. with $\delta = 1$ and $\lambda = 0$) and Dumer's (i.e. with $\delta = 1$ and $\lambda = \ell$) algorithms for the ranges of ℓ and p given by (23). For every case, the sizes of the sets of effective TMTO points corresponding to Stern's and Dumer's are the same and the corresponding time complexities are equivalent. The memory complexities are also mostly equivalent, though in a few cases they vary a little. We have similarly computed the sets of effective TMTO points corresponding to the Bernstein and Chou's (i.e. with $\delta = 1$) algorithm for the ranges of ℓ, p and λ given by (23). Again

the sizes of the sets of effective TMTO points are the same as those obtained for Stern's algorithm and the corresponding time complexities are equivalent, while the memory complexities are mostly equivalent and vary a little for a small number of cases.

Remark 7. The generalisation of Algorithm 1 over Stern's algorithm arises from the use of two parameter λ and δ. From the above discussion, we see that it is the parameter δ which provides the non-triviality of the generalisation. If we set $\delta = 1$ (obtaining Bernstein and Chou's algorithm isd1), then we do not obtain any interesting TMTO points beyond what is achieved by Stern's algorithm. It is necessary to allow δ to be less than 1 to explore the TMTO landscape not covered by Stern's algorithm.

The sets of effective TMTO points for the variants of Classic McEliece obtained from Algorithm 1 are shown in Tables 2 and 3. The tables also provide the values of the parameters which achieve the corresponding TMTO points. In these tables, rows marked with (*) indicate that Stern's algorithm achieves TMTO points which are equivalent to the corresponding TMTO points achieved by Algorithm 1. Further, Stern's algorithm does not achieve any TMTO point whose time complexity is equivalent to the time complexity of any row not marked with (*). The values in the tables show that the maximum and minimum values of $\lceil \log_2 T \rceil$ remain the same for Algorithm 1 and Stern's algorithm. For Algorithm 1, $\lceil \log_2 T \rceil$ achieves every value between the maximum and the minimum, while for Stern's algorithm only about half of these values are achieved. So Algorithm 1 provides a finer time/memory trade-off compared to Stern's algorithm. Note that for Algorithm 1 sets of 3712800 TMTO points reduce to sets of 6 to 13 effective TMTO points. This underlines the usefulness of the notion of a set of effective TMTO points.

The minimum memory TMTO points for the five Classic McEliece variants have $\log_2 M$ to be equal to 21.45, 22.54, 23.49, 23.45 and 23.79. Comparing with the values of $\log_2 M_{\text{mat}}$ in Table 1, one may observe that the minimum memory is marginally greater than $\log_2 M_{\text{mat}}$. The time estimates of the minimum memory trade-off points are less than that of the minimum time estimates of Leon's algorithm (whose memory complexity is M_{mat}) given in Table 6 (provided in Appendix A.1) by factors of about 8 to 20.

It is interesting to observe that there are sharp drops in memory requirement at only a moderate increase in the time complexity. As an example, if we consider the estimates which take memory access cost to be constant, increasing $\log_2 T$ by the amount indicated below leads to $\log_2 M$ dropping by the stated amount.

`m3488`: $\log_2 T$ from 147.93 to 148.93; $\log_2 M$ from 44.55 to 34.63;
`m4608`: $\log_2 T$ from 189.98 to 190.80; $\log_2 M$ from 46.22 to 37.70;
`m6688`: $\log_2 T$ from 265.00 to 265.95; $\log_2 M$ from 57.97 to 48.89;
`m6960`: $\log_2 T$ from 265.00 to 265.96; $\log_2 M$ from 67.37 to 58.38;
`m8192`: $\log_2 T$ from 300.00 to 300.96; $\log_2 M$ from 77.99 to 68.96.

This shows that allowing the time complexity to increase by factors which are at most 2 result in the memory requirement dropping by factors varying from 2^8 to 2^{10}. In general, one may observe that the drops in the memory requirement are sharper for smaller values of T and become less sharp for larger values of T. Similar observations hold when we consider the estimates which include logarithmic memory access cost.

We consider the implications of the new TMTO points to the classification in NIST categories of the variants of Classic McEliece. Time estimates of Stern's algorithm from [2] have been used in a discussion [14] regarding the classifications of the five variants. A criticism forwarded against these time estimates was that the corresponding memory requirements are much larger than the size of the public key being attacked.

Since we have obtained the sets of effective TMTO points, we may consider points where the memory size is not much larger than the size of the public key.

Let us first consider `m4608`. In this case the gate count requirement is 2^{207} and the size of the public key is about 2^{22} bits. The time estimates of all the TMTO points for this variant given by Tables 2a and 2b are below the target 2^{207}. The TMTO point with highest of these time estimates is equal to $(2^{197.99}, 2^{22.52})$ when memory access cost is taken to be constant, and is equal to $(2^{202.00}, 2^{22.54})$ when logarithmic memory access cost is considered. So for `m4608`, it is possible to achieve time complexity less than 2^{207} by a factor of about 2^9 (for constant memory access cost) and about 2^5 (for logarithmic memory access cost) while requiring memory which is only about 1.5 times the size of the public key.

Let us now consider `m6688` and `m6960`. The NIST gate count requirement is 2^{272} and the size of the public key is about 2^{22} bits. From Table 3a, we see that for `m6688`, Algorithm 1 has the TMTO points $(2^{269.99}, 2^{29.25})$ and $(2^{271.00}, 2^{27.69})$; and for `m6960`, Algorithm 1 has the TMTO points $(2^{270.00}, 2^{29.93})$ and $(2^{270.85}, 2^{28.83})$. All of these time estimates are slightly smaller than 2^{272}. The corresponding memory requirements, while still being larger than the size of the public key, are not too large. On the other hand, if we consider logarithmic memory access cost, then with memory requirement less than 2^{30}, we obtain time complexities $2^{274.97}$ and $2^{274.93}$ respectively, both of which are greater than the NIST requirement of 2^{272} by factors which are less than 8.

In contrast to the above, for both `m3488` and `m8192`, there is no TMTO point in Tables 2 and 3, even with constant memory access cost, which has time estimate less than the NIST requirement for classification in the respective categories.

Remark 8. For `m4608` and `m6688`, the minimum gate count estimates obtained in [4] for `isd1` (which subsumes Stern's and Dumer's algorithms) are $2^{198.93}$ and $2^{275.41}$ respectively. For `m4608`, the count of $2^{198.93}$ is lower than the NIST target of 2^{207} while for `m6688`, the count of $2^{275.41}$ is about 10 times the NIST target of 2^{272}. The methodology used in [4] for obtaining gate count estimates incorporates cost of memory access though in a manner which is different from the logarithmic memory access cost considered here (see Remark 6). From Tables 2b and 3b, the minimum bit complexity estimates for `m4608` and `m6688` obtained in the present work are not too far from those obtained in [4]. The main difference with the results reported in [4] is that, as discussed above, the bit complexity estimate for `m4608` falls below the NIST target and the bit complexity estimate for `m6688` (and also `m6960`) is a little above the NIST target *even* if we restrict the memory requirement to be not too larger than the size of the public key.

5.1 Previous estimates. Previous works on concrete estimates of the performance of ISD algorithms have focused on the minimum time that can be achieved for a fixed set of values of n, k and w, where the minimum is over the appropriate choices of the parameters.

		$(\log_2 T,\ \log_2 M)$	ℓ	p	λ	δ
m3488	(*)	(147.93, 44.55)	36	8	−36	1.000
	(*)	(148.93, 34.63)	27	6	−27	0.950
	(*)	(149.89, 27.15)	18	4	−18	1.000
		(150.97, 25.71)	17	4	−17	0.925
		(151.96, 24.77)	15	4	−14	0.875
		(152.81, 23.90)	14	4	−14	0.825
		(153.99, 22.80)	13	4	−13	0.750
	(*)	(154.98, 21.44)	13	4	−13	0.975
m4608	(*)	(189.98, 46.22)	38	8	−38	1.000
	(*)	(190.88, 37.70)	28	6	−28	0.975
		(191.81, 35.95)	26	6	−26	0.950
	(*)	(192.95, 27.75)	18	4	−18	0.975
		(193.99, 26.76)	16	4	−16	0.925
		(194.85, 25.81)	15	4	−15	0.875
		(195.74, 24.92)	14	4	−14	0.825
		(196.95, 23.81)	13	4	−13	0.750
	(*)	(197.99, 22.52)	7	3	−7	0.975

(a) Constant memory access cost.

		$(\log_2 T_{\text{ma}},\ \log_2 M)$	ℓ	p	λ	δ
m3488	(*)	(154.00, 34.69)	27	6	16	0.950
	(*)	(154.94, 26.67)	18	4	−18	0.975
		(155.81, 25.70)	16	4	−16	0.925
		(156.91, 24.76)	14	4	−14	0.875
		(157.94, 23.50)	13	4	−13	0.800
	(*)	(158.93, 21.45)	8	3	-8	1.000
m4608	(*)	(195.94, 36.70)	29	6	−29	0.975
	(*)	(196.98, 35.95)	26	6	−26	0.950
	(*)	(197.94, 27.75)	17	4	−17	0.975
		(198.73, 26.76)	16	4	−16	0.925
		(199.92, 25.80)	14	4	−14	0.875
		(200.97, 24.51)	13	4	−13	0.800
	(*)	(202.00, 22.54)	8	3	−2	1.000

(b) Logarithmic memory access cost.

Table 2: TMTO points and the corresponding values of the parameters achieved by Algorithm 1 for m3488 and m4608. Stern's algorithm provides equivalent TMTO points for only the rows marked with (*).

For Classic McEliece, previous estimates of Stern's algorithm in [2,4] are shown in Table 4. To make the comparison clear, we have also included the minimum time estimates and the corresponding memory estimates for Algorithm 1 for both constant and logarithmic memory access costs; equivalent TMTO points can also be obtained from Stern's algorithm (see Tables 2 and 3). In the column headed by [2], T_1 is the minimum time estimate for Stern's algorithm given in that work and M_1 is the memory required to achieve the corresponding time estimate. In the column headed by [4], T_2 is the minimum time estimate of either Stern's or Dumer's algorithm given in that work, the specific algorithm for each time estimate is not mentioned in [4]. Also, [4] does not provide the memory estimates. The time estimates in [4] include the cost of memory access (see, however, Remark 6), while those in [2] do not[1]. From Table 4, we note that the estimates T_1 in [2] are higher than the estimates T achieved by Algorithm 1. The reason is that the techniques of Chase's sequence and early abort were not used in the estimates in [2]. On the other hand, the estimates in [4] are higher than T_{ma}. The values of $\log_2 T_2 - \log_2 T_{\text{ma}}$ for m3488, m4608, m6688 and m8192 are 2.96, 2.99, 4.45 and 5.44 respectively. So the time estimates obtained from our simple cost model are fairly accurate; in particular they are within 1.5% to 1.9% of the time estimates obtained from the sophisticated methodology used in [4].

For Classic McEliece, bit complexity estimates of the MMT [18] algorithm were given in [12] for various cases. We focus on the case where the memory requirement is restricted to at most 2^{60} bits. The bit complexity estimates from [12] for this case are shown in Table 5 both for the cases where memory access time is taken to be constant and for logarithmic memory access time. Table 5 also provides the time estimates for Algorithm 1 with memory restricted to at most 2^{60} bits. These estimates are obtained from Tables 2 and 3. Since our methodology permits obtaining the memory estimates

[1] While [2] mentions the logarithmic memory access cost model, to the best of our understanding Table 4 of [2] provides estimates of bit operations assuming constant memory access time. The values in Table 4 in the column headed by [2] are from Table 4 of [2].

		$(\log_2 T,\ \log_2 M)$	ℓ	p	λ	δ
m6688	(*)	(265.00, 57.97)	50	10	−16	1.000
	(*)	(265.95, 48.89)	39	8	−39	1.000
	(*)	(267.00, 38.82)	31	6	30	0.975
		(268.80, 36.41)	28	6	−28	0.900
	(*)	(269.99, 29.25)	18	4	−18	0.975
		(271.00, 27.69)	18	4	7	0.900
		(271.93, 26.68)	17	4	−17	0.850
		(273.00, 25.76)	15	4	15	0.800
		(273.95, 24.95)	14	4	−14	0.750
	(*)	(274.89, 23.48)	9	3	−9	0.975
m6960	(*)	(265.00, 67.37)	59	12	19	1.000
	(*)	(265.96, 58.38)	48	10	−48	1.000
	(*)	(267.00, 48.29)	39	8	29	0.975
	(*)	(267.91, 39.00)	30	6	−30	0.975
		(268.92, 37.41)	29	6	−29	0.925
	(*)	(270.00, 29.93)	20	4	20	1.000
		(270.85, 28.83)	19	4	−19	0.950
		(271.96, 27.77)	17	4	−17	0.900
		(272.87, 26.75)	16	4	−16	0.850
		(273.80, 25.80)	15	4	−15	0.800
		(274.74, 24.98)	14	4	−14	0.750
	(*)	(275.87, 23.45)	8	3	−8	1.000
m8192	(*)	(300.00, 77.99)	68	14	16	1.000
	(*)	(300.96, 68.96)	58	12	-58	1.000
	(*)	(301.90, 58.55)	50	10	−50	0.975
	(*)	(302.90, 49.38)	40	8	−40	0.975
	(*)	(303.96, 39.88)	31	6	−31	0.975
		(305.00, 38.26)	30	6	−30	0.925
		(305.95, 37.44)	28	6	−28	0.900
	(*)	(306.92, 29.99)	19	4	−19	0.975
		(307.80, 28.89)	18	4	−18	0.925
		(308.72, 27.81)	17	4	−17	0.875
		(309.66, 26.79)	16	4	−16	0.825
		(310.96, 25.85)	14	4	−14	0.775
	(*)	(311.85, 23.79)	10	3	−10	1.000

(a) Constant memory access cost.

		$(\log_2 T_{\text{ma}},\ \log_2 M)$	ℓ	p	λ	δ
m6688	(*)	(270.96, 57.91)	49	10	−49	1.000
	(*)	(271.99, 39.55)	31	6	−31	1.000
		(272.96, 37.98)	29	6	−29	0.950
		(273.99, 36.41)	28	6	−28	0.900
	(*)	(274.97, 28.72)	19	4	−19	0.950
		(275.97, 27.68)	17	4	−17	0.900
		(276.83, 26.67)	16	4	−16	0.850
		(278.00, 25.75)	14	4	−7	0.800
		(278.88, 24.95)	13	4	−13	0.750
	(*)	(279.86, 23.45)	8	3	−8	0.950
m6960	(*)	(271.00, 75.99)	68	14	−4	1.000
	(*)	(272.00, 49.33)	40	8	32	1.000
	(*)	(272.94, 39.79)	30	6	−30	1.000
		(274.00, 37.45)	30	6	13	0.925
	(*)	(274.93, 29.91)	20	4	−20	1.000
		(275.91, 28.83)	18	4	−18	0.950
		(276.75, 27.77)	17	4	−17	0.900
		(277.97, 26.75)	15	4	−15	0.850
		(278.84, 25.80)	14	4	−14	0.800
	(*)	(279.89, 23.45)	10	3	−10	1.000
m8192	(*)	(306.00, 86.78)	77	16	46	1.000
	(*)	(307.00, 59.88)	52	10	−9	1.000
	(*)	(307.93, 50.43)	41	8	−41	1.000
	(*)	(309.00, 40.70)	31	6	−31	1.000
		(309.97, 39.88)	29	6	−29	0.975
		(310.90, 37.45)	29	6	−29	0.900
	(*)	(311.82, 29.99)	19	4	−19	0.975
		(312.95, 28.35)	18	4	−18	0.900
		(313.90, 27.81)	16	4	−16	0.875
		(314.77, 26.78)	15	4	−15	0.825
		(315.90, 25.44)	14	4	−14	0.750
	(*)	(316.84, 23.78)	9	3	−9	0.975

(b) Logarithmic memory access cost.

Table 3: TMTO points and the corresponding values of the parameters achieved by Algorithm 1 for m6688, m6960 and m8192. Stern's algorithm provides equivalent TMTO points for only the rows marked with (*).

name	[2] $(\log_2 T_1, \log_2 M_1)$	[4] $\log_2 T_2$	Algo 1 $(\log_2 T, \log_2 M)$	Algo 1 $(\log_2 T_{\text{ma}}, \log_2 M)$
m3488	(152.51, 34.68)	156.96	(147.93, 44.55)	(154.00, 34.69)
m4608	(194.36, 35.66)	198.93	(189.98, 46.22)	(195.94, 36.70)
m6688	(270.46, 37.48)	275.41	(265.00, 57.97)	(270.96, 57.91)
m6960	(271.18, 47.58)	-	(265.00, 67.37)	(271.00, 75.99)
m8192	(306.63, 67.64)	311.44	(300.00, 77.99)	(306.00, 86.78)

Table 4: Previous estimates of the expected number of bit operations and the corresponding memory for Stern's/Dumer's algorithm. The minimum time estimates required by Algorithm 1 are also provided.

	MMT		Algo 1	
	$(\log_2 T_3, \log_2 M_3)$	$(\log_2 T_4, \log_2 M_4)$	$(\log_2 T, \log_2 M)$	$(\log_2 T_{\text{ma}}, \log_2 M)$
m3488	$(145.47, \leq 60)$	$(151.06, \leq 60)$	$(147.93, 44.55)$	$(154.00, 34.69)$
m4608	$(188.16, \leq 60)$	$(193.59, \leq 60)$	$(189.98, 46.22)$	$(195.94, 36.70)$
m6688	$(263.16, \leq 60)$	$(268.66, \leq 60)$	$(265.00, 57.97)$	$(270.96, 57.91)$
m6960	$(263.64, \leq 60)$	$(269.17, \leq 60)$	$(265.96, 58.38)$	$(272.00, 49.33)$
m8192	$(298.65, \leq 60)$	$(304.19, \leq 60)$	$(301.90, 58.55)$	$(307.00, 59.88)$

Table 5: Comparison of time estimates from [12] achieved by the MMT algorithm [18] with the time estimates for Algorithm 1. In both cases, memory is restricted to at most 2^{60} bits. T_3 and M_3 respectively denote the time and memory estimates from [12] for the MMT algorithm in the case where the memory access cost is taken to be constant; T_4 and M_4 respectively denote the time and memory estimates from [12] for the MMT algorithm in the case where logarithmic memory access cost is considered.

required for achieving the corresponding time estimates, we provide these values in Table 5. The time estimates for the MMT algorithm are lower than the time estimates for Algorithm 1, which confirms the well known fact that the MMT algorithm is faster than Stern's algorithm. Note, on the other hand, that the actual gain in speed by the MMT algorithm over Stern's algorithm is by a factor which is less than 8 in all the cases. This small loss of speed by Stern's algorithm is to be contrasted with the relative simplicity of Stern's algorithm in comparison with the MMT algorithm. We note, however, that a direct comparison between the estimates in [12] and the estimates obtained here may be misleading; [12] does not describe the details of the methodology used to obtain the estimates and it is possible that techniques such as early abort were not incorporated in obtaining the estimates in [12]. Incorporating such techniques may lower the estimates in [12] making the gap between the time estimates of the MMT algorithm and (generalised) Stern's algorithm wider than what is suggested by the values in Table 5.

6 Conclusion

We have proposed a generalised version of Stern's ISD algorithm. By appropriately selecting the values of the parameters of the new algorithm, it is possible to obtain Stern's, Dumer's, and Bernstein and Chou's algorithms as special cases. We also introduced the notion of a set of effective TMTO points and showed how to compute such a set for any ISD algorithm. The sets of effective TMTO points for the new ISD algorithm corresponding to the variants of the Classic McEliece cryptosystems have been obtained and their significance discussed in details.

The present work has focussed on obtaining concrete estimates using the generalised Stern's algorithm. From a more theoretical point of view, it would be of interest to perform a comprehensive asymptotic analysis of the generalised Stern's algorithm following the methodology used in [7]. A more compact asymptotic analysis along the lines of [18,3] or the more recent [12] would also be of interest. We leave this as a possible future work.

Acknowledgement. We thank the reviewers for their kind comments which have helped in improving the discussion at several points in the paper.

References

1. Nicolas Aragon, Paulo L. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE, Round 4 submission. https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/BIKE-Round4.zip (accessed on 9th August, 2023), 2022.
2. 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.
3. 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, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012.
4. Daniel J. Bernstein and Tung Chou. CryptAttackTester: formalizing attack analyses. https://cat.cr.yp.to/papers.html#cryptattacktester (accessed on 9th August, 2023), 2023.
5. Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece, Round 4 submission. https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/mceliece-Round4.tar.gz (accessed on 9th August, 2023), 2022.
6. 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, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008.
7. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011.
8. 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, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 25–46. Springer, 2018.
9. Ilya Dumer. On minimum distance decoding of linear codes. In *Proceedings of the 5th Joint Soviet-Swedish International Workshop on Information Theory*, pages 50–52, 1991.
10. Andre Esser and Emanuele Bellini. Syndrome decoding estimator. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 112–141. Springer, 2022.

11. 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, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 433–457. Springer, 2022.
12. 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, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 360–390. Springer, 2023.
13. Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009.
14. Kirk Fleming. ROUND 3 OFFICIAL COMMENT: Classic McEliece, started on November 10, 2020, 7:05:28 AM. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/EiwxGnfQgec?pli=1 (accessed on 9th August, 2023), 2020.
15. Donald E. Knuth. *Art of Computer Programming, Volume 4A, The Combinatorial Algorithms, Part 1*.
16. Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In Christoph G. Günther, editor, *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988.
17. Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Inf. Theory*, 34(5):1354–1359, 1988.
18. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011.
19. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228. Springer, 2015.
20. Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. Jet Propulsion Laboratory DSN Progress Report 42–44. http://ipnpr.jpl.nasa.gov/progressreport2/42-44/44N.PDF, 1978.
21. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jurjen Bos, Jean-Christophe Deneuville, Arnaud Dion, Philippe Gaborit, Jérôme Lacan, Edoardo Persichetti, Jean-Marc Robert, Pascal Véron, and Gilles Zémor. HQC, Round 4 submission. https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/HQC-Round4.zip (accessed on 9th August, 2023), 2022.
22. Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

23. NIST. Call for proposals. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf (accessed on 9th August, 2023), 2016.
24. NIST. Round 4 submissions. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions (accessed on 9th August, 2023), 2022.
25. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962.
26. W.A. et al Stein. Sage Mathematics Software (Version 8.1), The Sage Development Team. http://www.sagemath.org (accessed on 9th August, 2023), 2017.
27. Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.
28. 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, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2016.
29. Valentin Vasseur. Information set decoding implementation. https://github.com/vvasseur/isd/blob/master/src/dumer.c (accessed on 9th August, 2023), 2019.
30. Floyd Zweydinger. Decoding. https://github.com/FloydZ/decoding/blob/master/src/dumer.h (accessed on 9th August, 2023), 2022.

A SAGE code

```
import sys
ZZ=IntegerRing()
RR=RealField(100)
def getTuple(n,k,w,l,p,lam,delta):
    L1 = RR(binomial(floor((k+lam)/2),floor(p/2))); Lsz = RR(ceil(L1^delta))
    L2 = RR(binomial(ceil((k+lam)/2),ceil(p/2)))
    PS = (Lsz * L2 * RR(binomial(n-k-l,w-p))) / RR(binomial(n,w))
    TLA = RR(k^2*(n-k)*(n-k-1)*(3*n-k)) / RR((4*n^2))
    TSR = (Lsz+L2)*(2*l+4*(w-p+1)) + L2*(Lsz/RR(2^l))*(2*(w-p+1))
    Mmat = (n-k)*(n+1)
    Mlst = Lsz*(l+2*(w-p+1)+RR(floor(p/2))*log(RR(floor((k+lam)/2)),2));
    M = Mmat + Mlst; logM = log(M,2)
    Titer = (TLA+TSR)*log(RR(M),2) # Titer = TLA + TSR
    logT = log(Titer,2) - log(PS,2)
    tuple = [ceil(logT),ceil(logM),l,p,lam,delta,logT,logM]
    return tuple
def FilterTime(resLst): # first filtering
    tresLst = []; tresLst.append(resLst[0]); val = resLst[0][0]
    for tuple in resLst:
        if (tuple[0] != val):
            tresLst.append(tuple)
            val = tuple[0]
    return tresLst
def FilterMemory(resLst): # second filtering and pruning
    tresLst = []; tresLst.append(resLst[0]); val = resLst[0][1]
    for tuple in resLst:
        if (tuple[1] > val):
            return tresLst
        if (tuple[1] != val):
            tresLst.append(tuple)
            val = tuple[1]
    return tresLst
# start of main routine: takes three command line parameters: n,k,w
```

```
n=int(sys.argv[1]); k=int(sys.argv[2]); w=int(sys.argv[3])
lmax = 100; pmin = 2; pmax = 30
resLst = []
for i in range(0,13):
    delta = 1.0-RR(i)*0.025
    for l in range(1,lmax+1):
        for lam in range(-l,l+1):
            for p in range(pmin,pmax):
                tuple = getTuple(n,k,w,l,p,lam,delta); resLst.append(tuple)
resLst.sort(); tresLst = FilterTime(resLst); tresLst = FilterMemory(tresLst)
for tuple in tresLst:
    print tuple
```

A.1 Complexity of Prange's, Lee-Brickell's and Leon's algorithms. Let C_1, C_2 and C_3 be the expected number of bit operations required required by Prange's, Lee and Brickell's and Leon's algorithms respectively. We have

$$C_1 = \frac{\binom{n}{w}}{\binom{n-k}{w}} \cdot (T_{\mathrm{LA}} + 2(w+1)), \tag{24}$$

$$C_2 = \frac{\binom{n}{w}}{\binom{k}{p}\binom{n-k}{w-p}} \cdot \left(T_{\mathrm{LA}} + \binom{k}{p} \cdot (4(w-p+1)) \right), \tag{25}$$

$$C_3 = \frac{\binom{n}{w}}{\binom{k}{p}\binom{n-k-\ell}{w-p}} \cdot \left(T_{\mathrm{LA}} + \binom{k}{p} \cdot \left(2\ell + \frac{2p(w-p+1)}{2^\ell} \right) \right). \tag{26}$$

All the three algorithms store only $\mathbf{H}_0$ and $\mathbf{s}_0$ and hence the number of bits required to be stored is M_{mat}.

Justifications for the estimates in (24), (25) and (26). A generalisation of Algorithm 1 has been outlined in Section 3.1 which provides Prange's, Lee-Brickell's and Leon's algorithms as special cases. The analysis of time and memory complexities of Algorithm 1 can also be easily generalised to obtain expressions for the time and memory complexities of the unified algorithm sketched in Section 3.1. The expressions for C_1, C_2 and C_3 can then be obtained as special cases. Instead of adopting this approach, we briefly sketch the arguments required to directly obtain the required expressions.

As in the case of Algorithm 1, applying (12), the expected number of bit operations is given by the product of the expected number of bit operations in each iteration and the inverse of the success probability. The success probabilities of the three algorithms can be obtained using standard arguments and we refer to [2] for the details. Each iteration consists of the linear algebra step and the search step. The number of bit operations in the linear algebra step is the same as that in Stern's algorithm which we take to be T_{LA}. It only remains to consider the expected number of bit operations in the search step. The difference with the analysis in [2] arises from our use of Chase's sequence and the early abort technique.

In Prange's algorithm, the search step is simply to check that the weight of $\mathbf{s}$ is w. Using the early abort technique, this requires about $2(w+1)$ bit operations in almost all cases.

In the Lee-Brickell algorithm, the search step computes all possible p-column combinations of $\mathbf{A}$ (which in this case is an $r \times k$ matrix) and checks whether the sum of $\mathbf{s}$ and any such p-column combination $\mathbf{x}$ is equal to $w-p$. Using Chase's sequence, the

next p-column combination from the previous one can be generated using two vector additions. Further, using the early abort technique, it is sufficient to compute only the first $2(w-p+1)$ bits $\mathbf{s}+\mathbf{x}$ in almost all cases. So the expected number of bit operations is $4(w-p+1)$.

In Leon's algorithm, the search step computes all possible p-column combinations of $\mathbf{A}$ (which in this case is an $\ell \times k$ matrix) and checks whether any such $\mathbf{x}$ is equal to $\mathbf{u}$. Using Chase's sequence, the number of bit operations required to generate the next p-column combination from the previous one is 2ℓ. If some $\mathbf{x}$ is equal to $\mathbf{u}$, then the corresponding p-column combination $\mathbf{y}$ of $\mathbf{B}$ is computed and it is checked whether the weight of $\mathbf{y}+\mathbf{v}$ is equal to $w-p$. We make the usual heuristic assumption that the $\mathbf{x}$'s generated by all the p-column combinations of $\mathbf{A}$ are independent and uniformly distributed. Under this assumption, any particular $\mathbf{x}$ is equal to $\mathbf{u}$ with probability $1/2^{\ell}$ and so the number of $\mathbf{x}$'s which are equal to $\mathbf{u}$ follows the binomial distribution with parameters $1/2^{\ell}$ and $\binom{k}{p}$. So the expected number of times $\mathbf{x}$ is equal to $\mathbf{u}$ is equal to $\binom{k}{p}/2^{\ell}$. For each such match, the sum $\mathbf{y}+\mathbf{v}$ is to be computed. Using the early abort technique, it is sufficient to compute the first $2(w-p+1)$ bits of this sum in almost all cases. This gives us the required expression.

Concrete estimates of time complexity of Leon's algorithm for Classic McEliece. Table 6 provides the minimum value of C_3, the expected number of bit operations required by Leon's algorithm, where the minimum is taken over the appropriate ranges of ℓ and p. The column headed by $\log_2(C_3 \cdot \log_2 M_{\text{mat}})$ reports the time estimates of Leon's algorithm adjusted for logarithmic memory access cost. We used a SAGE code to compute the minimum values of C_3 and $\log_2(C_3 \cdot \log_2 M_{\text{mat}})$. Since Leon's algorithm requires M_{mat} bits of memory for all parameter choices, the code is simpler than the code given in Appendix A. The entries in the column headed by A are the binary logarithms of the minimum time estimates of Lee-Brickell or Leon's algorithm from [4]; the work does not specify the particular algorithm corresponding to a particular time estimate. While [4] captures a wide range of optimisations as well as costs, including memory access costs, it may be noted that the values in column A are in-between our estimates of Leon's algorithm with constant and logarithmic memory access costs, which are quite close.

	$\log_2 C_3$	$\log_2(C_3 \cdot \log_2 M_{\text{mat}})$	A [4]
m3488	157.91	162.33	159.93
m4608	200.71	205.20	202.30
m6688	278.28	282.82	279.88
m6960	279.33	283.87	-
m8192	316.05	320.62	317.66

Table 6: Concrete time estimates of Leon's algorithm. The entries in Column-A are estimates of either Leon's or Lee-Brickell's algorithms.