Multilevel Amortized Gaussian Elimination in Information-Set Decoding: Applications to HQC and PCG

Kévin Carrier[1,2], Valerian Hatey[1], Laura Luzzi[1,3], and Jean-Pierre Tillich[3]

[1] ETIS UMR 8051 - Cergy-Paris Université, ENSEA, CNRS,
[kevin.carrier,valerian.hatey]@cyu.fr,laura.luzzi@ensea.fr
[2] LIX, CNRS, École Polytechnique, IPP, Inria Saclay,
kevin.carrier@polytechnique.edu
[3] Project COSMIQ, Inria de Paris, [laura.luzzi,jean-pierre.tillich]@inria.fr

Abstract. For cryptosystems whose security relies on the hardness of decoding in the sublinear regime, the best known attacks are based on Information Set Decoding (ISD). In this regime, which is particularly relevant to HQC and Pseudorandom Correlation Generators (PCG), the cost of Gaussian elimination is no longer negligible and significantly affects the overall attack complexity.
In this work, we revisit the Reduce-and-Prange technique of Kim and Lee, which reduces the cost of Gaussian elimination by reusing partial pivots. We refine its complexity analysis using branching-process techniques, thereby obtaining a more accurate assessment of its performance. We then extend partial pivot reuse to Stern's algorithm and introduce *MAGE-Stern*, a multilevel amortized Gaussian elimination variant of Stern's algorithm.
Under a consistent logic-gate cost model, MAGE-Stern improves upon the best previously known attack against HQC by approximately 3 bits in time complexity, while reducing the memory complexity by about 12 bits. In particular, we estimate the security of the standardized HQC Category I parameter set at approximately 140 bits, about 3 bits below its NIST security target. We further combine multilevel amortized Gaussian elimination with the projective decoding framework of Carrier, Hatey, and Tillich, and investigate its application to regular decoding. Applied to the reference Pseudorandom Correlation Generator (PCG) parameter sets of Boyle, Couteau, Gilboa, and Ishai, the resulting algorithms improve upon the best previously known attacks by up to 6 bits across a broad range of practical parameters.

Keywords: Code-based cryptography · Information Set Decoding · HQC · Pseudorandom Correlation Generators · Branching Processes

1 Introduction

1.1 Decoding in the Sublinear Regime and the HQC Cryptosystem

A large class of code-based cryptosystems derives its security from the hardness of the decoding problem.

Problem 1.1 (Decoding problem $\mathsf{Dec}_q(n,k,t)$). *Let $\mathbf{G} \in \mathbb{F}_q^{n\times k}$ be a generator matrix of a code $\mathcal{C}$, and let $\boldsymbol{y} = \mathbf{G}\boldsymbol{u} + \boldsymbol{e}$ be a noisy codeword with $\boldsymbol{u} \in \mathbb{F}_q^k$ and $\boldsymbol{e} \in \mathbb{F}_q^n$ of Hamming weight t. The goal is to recover the error vector $\boldsymbol{e}$.*

This is notably the case for the HQC cryptosystem [2], which was recently selected by NIST for standardization. More precisely, the security of HQC relies on the hardness of a variant of the decoding problem over $\mathbb{F}_2$, namely the Decoding-One-Out-of-Many (DOOM) problem, in which an attacker is given k noisy codewords $\boldsymbol{y}_i$ and is required to decode at least one of them. Moreover, the error distribution has a regular structure and the parameters lie in the sublinear regime, where $k = Rn,\ R \in [0,1]$ and $t = \Theta(\sqrt{n}) = o(n)$.

The most effective attacks against the decoding problem are based on Information Set Decoding (ISD). In the sublinear regime, Canto-Torres and Sendrier [15] showed that for most ISD variants, the asymptotic complexity is the same:

$$T_{\mathrm{ISD}} = \mathsf{poly}(n)\ 2^{-t\log_2(1-R)+o(t)}. \tag{1.1}$$

Further analyses by Peters [34] established that, for several ISD variants, the second-order term in the complexity exponent is of order $\frac{t^2}{n}$. For the HQC parameters, where $t = \Theta(\sqrt{n})$, this second-order contribution is asymptotically constant[4]. Consequently, the asymptotic behavior of existing ISD algorithms is essentially identical, making improvements to the polynomial factors particularly relevant. This observation is further supported by the results of [22, Table 4], which evaluate the non-asymptotic complexity of the main ISD variants for the HQC parameter sets. While these ISD variants mainly aim at improving the asymptotic complexity exponent, their complexities differ by at most one bit for the HQC parameters, with Stern's algorithm providing the best attack.

More recently, building on Sieving-ISD [28], Progressive Sieving-ISD was introduced in [45] and reported to outperform classical ISD algorithms for several HQC parameter sets, with an improvement of approximately 8 bits over Stern's algorithm for the Category I parameters. Our re-evaluation shows, however, that most of this reported gain stems from discrepancies in the complexity assessment rather than from an intrinsic algorithmic improvement. First, Stern and Progressive Sieving-ISD are not evaluated under the same random-access cost assumptions in the reference implementation: memory accesses are assigned a logarithmic cost for Stern, but a unit cost for Progressive Sieving-ISD. Second, the complexity estimate for Stern includes an additional multiplicative factor that, according to our analysis, is already accounted for in its success probability and is therefore counted twice. Third, Stern is first evaluated in the standard decoding setting and its complexity is then divided by $\sqrt{k}$ to account for the DOOM setting, whereas a direct adaptation using the classical $(p, p-1)$ trick

[4] We also investigated dual attacks as a possible approach to assessing the security of HQC. However, our preliminary results suggest that these techniques are not competitive in the sublinear regime relevant to the HQC parameters. In particular, their second-order complexity term appears to be larger than that of ISD algorithms.

of [24] yields a slightly larger speedup for the HQC parameters. Finally, the comparison does not incorporate several standard optimizations of Stern's algorithm, including Peter's techniques [34]. After correcting these discrepancies and evaluating both algorithms consistently in a properly defined logic-gate model, the advantage of Progressive Sieving-ISD over Stern is reduced to less than one bit, as shown in Tables 6.3 and 6.4.

As discussed above, the results of [22] suggest that, for HQC, further improvements are more likely to come from reducing the polynomial factors than from refining the asymptotic exponent. In this context, reducing the cost of Gaussian elimination becomes especially important, as linear algebra constitutes one of the dominant costs of an ISD iteration. A first approach, proposed by Canteaut and Chabaud [13], and later refined by Bernstein, Lange and Peters [7], updates the information set by exchanging only one coordinate between consecutive iterations. This reduces the cost of Gaussian elimination from cubic to quadratic but introduces strong dependencies between successive iterations, which offset the expected linear speedup in the sublinear regime.

More recently, Kim and Lee [30] introduced a different strategy for Prange's algorithm [36], called Reduce-and-Prange (RaP). Instead of recomputing a full Gaussian elimination at each iteration, their algorithm progressively constructs the information set through a sequence of partial reductions, each requiring a smaller Gaussian elimination. Each partial reduction is repeated sufficiently many times to validate the corresponding partial guess. The resulting tree structure provides a much finer control over the dependencies introduced by these partial reductions and leads to the best known practical Gaussian elimination reduction for Prange's algorithm.

Beyond HQC, reducing the cost of linear algebra is also of practical interest for decoding over large finite fields, where Gaussian elimination remains a dominant component of the overall complexity.

1.2 Decoding over Large Fields and the Security of PCG Protocols

The security of many Pseudorandom Correlation Generator (PCG) protocols relies on the hardness of the decoding problem or its regular counterpart over large finite fields [39,44,27,4,20,19,38,12,37]. In particular, [10] proposed reference parameter sets over the field $\mathbb{F}_{2^{128}}$, which are widely used to evaluate the security of PCG constructions.

The large-field setting fundamentally changes the behavior of Information Set Decoding algorithms. Indeed, an ISD algorithm enumerating vectors of weight p incurs an enumeration cost containing the factor $(q-1)^p$. When q is large, this factor rapidly dominates the overall complexity, making the enumeration of vectors of positive weight prohibitively expensive.

As a consequence, it was shown in [14] that, as $q \to +\infty$, all known ISD algorithms become asymptotically equivalent to Prange's algorithm. Accordingly, the security of current PCG parameter sets is primarily evaluated using Prange's

algorithm and its practical improvements, such as the Reduce-and-Prange algorithm of [30]. In the regular setting, these attacks also compete with algebraic and hybrid attacks [11,42,30].

Recently, [18] showed that ISD algorithms can be adapted to the projective space. By enumerating projective equivalence classes rather than vectors themselves, one factor of $(q-1)$ is removed from the enumeration cost. Consequently, projective ISD algorithms can consider non-zero enumeration weights while avoiding the prohibitive penalty associated with large fields.

This observation calls into question the current security evaluation of PCG protocols and suggests that projective ISD algorithms should also be taken into account when calibrating their parameters.

1.3 Our Contributions

We first revisit the theoretical analysis of the Reduce-and-Prange algorithm of [30]. Its success probability was previously estimated as a product of independent events. We show that these events are in fact correlated and that the evolution of the algorithm is more accurately described by the survival of a multitype branching process. This yields a corrected expression for the success probability and shows that the analysis of [30] overestimates it. We also refine the algorithm by allowing the number of repetitions to vary across reduction levels. The resulting parameters govern a tradeoff between the cost of each execution and the survival probability of the branching process, and their joint optimization leads to a lower overall complexity.

Building on this analysis, we introduce Multilevel Amortized Gaussian Elimination for Stern's algorithm (*MAGE-Stern*), which incorporates the multilevel Gaussian-elimination technique of [30] into Stern's algorithm. We analyze its success probability through the same branching-process framework and apply it to the security evaluation of HQC under a theoretical cost model aligned with the NIST security requirements. In this model, Gaussian elimination accounts for a substantial fraction of the computational cost, making its amortization particularly effective. Compared with classical Stern, MAGE-Stern reduces the attack complexity by approximately 3 bits and the memory complexity by approximately 12 bits. The reduction in Gaussian-elimination cost also shifts the optimal enumeration weight from $p = 5$ to $p = 3$, thereby reducing both the running-time and memory contributions associated with list enumeration from cubic to quadratic in n. Applied to the official HQC parameter sets, MAGE-Stern yields the security estimates reported in Tables 6.3 and 6.4, making it the most efficient known attack against HQC. In particular, our analysis estimates the security of the current HQC Category I parameters at approximately 140 bits, about 3 bits below the NIST target of 143 bits.

Extending our approach to the non-binary setting over large finite fields, we introduce *Projective-MAGE-Stern* by combining the projective decoding framework of [18] with multilevel amortized Gaussian elimination. We then extend this

approach to decoding problems with regular error structure by incorporating the partial-guessing strategy of [25,30], resulting in *Regular-Projective-MAGE-Stern*. For the large-field parameter sets proposed in [10], using projective representations in the meet-in-the-middle step of ISD allows vectors of weight $p = 2$ to be enumerated without paying the usual factor of $(q-1)$. Projective-MAGE-Stern therefore combines the increased success probability of Projective-Stern with the reduced Gaussian elimination cost of Reduce-and-Prange. Its application, together with that of its regular counterpart, to the reference PCG parameter sets of [10] yields the estimates reported in Tables 8.1, 8.2, 8.3, and 8.4. Projective-MAGE-Stern improves upon Reduce-and-Prange by up to 10 bits and upon the attacks selected by the estimator of [46] by up to 6 bits. Overall, Projective-MAGE-Stern and Regular-Projective-MAGE-Stern provide the best known attacks for a broad range of the PCG parameter sets proposed in [10].

2 Preliminaries

2.1 Notation

Vectors and matrices are denoted by bold lowercase and uppercase letters, respectively, such as $\boldsymbol{v}$ and $\mathbf{M}$. Given a matrix $\mathbf{M}$, we write $\mathbf{M}^\top$ for its transpose. Throughout the paper, all vectors are regarded as column vectors; consequently, $\boldsymbol{v}^\top$ denotes the corresponding row vector.

Given a vector $\boldsymbol{v}$, we denote its i-th coordinate by v_i. The support of $\boldsymbol{v}$, denoted by $\mathsf{supp}(\boldsymbol{v})$, is the set of indices of its nonzero entries: $\mathsf{supp}(\boldsymbol{v}) \stackrel{\text{def}}{=} \{i : v_i \neq 0\}$. The Hamming weight of $\boldsymbol{v}$ is the size of its support, and is denoted by $|\boldsymbol{v}|$.

We denote integer intervals using double brackets: for instance, $[\![a, b]\!]$ is the set of integers between a and b. If $I \subseteq [\![1, n]\!]$, we denote its complement in $[\![1, n]\!]$ by $\overline{I} \stackrel{\text{def}}{=} [\![1, n]\!] \setminus I$.

Given a set of indices $I \subseteq [\![1, n]\!]$ and a vector $\boldsymbol{v} \in \mathbb{F}_q^n$, the notation $\boldsymbol{v}_I$ stands for the subvector of $\boldsymbol{v}$ obtained by restricting $\boldsymbol{v}$ to the coordinates indexed by I, namely $\boldsymbol{v}_I = (v_i)_{i \in I}$. Similarly, given two sets of indices $I \subseteq [\![1, m]\!]$ and $J \subseteq [\![1, n]\!]$ and a matrix $\mathbf{M} \in \mathbb{F}_q^{m \times n}$, we denote by $\mathbf{M}_{[I:J]}$ the submatrix of $\mathbf{M}$ obtained by selecting the rows indexed by I and the columns indexed by J, namely $\mathbf{M}_{[I:J]} \stackrel{\text{def}}{=} (M_{i,j})_{i \in I, j \in J}$, where $M_{i,j}$ denotes the entry of $\mathbf{M}$ at position (i, j). In particular, $\mathbf{M}_{[I:]}$ denotes the submatrix obtained by selecting the rows whose indices belong to I, while $\mathbf{M}_{[:J]}$ denotes the submatrix obtained by selecting the columns whose indices belong to J.

2.2 Coding Background

We first recall basic definitions related to error-correcting codes.

Definition 2.1 (Linear Code, Generator Matrix). *A $[n, k]_q$ linear code $\mathcal{C}$ of length n and dimension k over $\mathbb{F}_q$ is a k-dimensional subspace of $\mathbb{F}_q^n$. The elements of $\mathcal{C}$ are called codewords.*

A matrix $\mathbf{G} \in \mathbb{F}_q^{n\times k}$ *whose columns form a basis of* $\mathcal{C}$ *is called a* generator matrix *of* $\mathcal{C}$*; that is,* $\mathcal{C} = \{\mathbf{G}\boldsymbol{u} \,:\, \boldsymbol{u} \in \mathbb{F}_q^k\}$.

Most code-based cryptosystems rely on the hardness of the decoding problem or its variants.

Problem 2.1 (Decoding Problem $\mathsf{Dec}_q(n,k,t)$**).** *Let* $\mathcal{C}$ *be a* $[n,k]_q$ *linear code, let* $\boldsymbol{e} \in \mathbb{F}_q^n$ *be a vector of Hamming weight* t*, and let* $\boldsymbol{y} \in \mathcal{C} + \boldsymbol{e}$*. Given* $(\mathcal{C}, \boldsymbol{y})$*, recover* $\boldsymbol{e}$*.*

Throughout this paper, we restrict our attention to parameter sets for which $\boldsymbol{e}$ is the unique error vector of Hamming weight t satisfying $\boldsymbol{y} - \boldsymbol{e} \in \mathcal{C}$. The decoding problem admits two equivalent formulations, depending on whether the code is represented by a generator matrix or by a generator matrix of its orthogonal space (that is a parity-check matrix). In this work, we adopt the generator-matrix formulation, in which solving $\mathsf{Dec}_q(n,k,t)$ amounts to finding a vector $\boldsymbol{u} \in \mathbb{F}_q^k$ such that $|\boldsymbol{y} - \mathbf{G}\boldsymbol{u}| = t$.

In HQC, the quasi-cyclic structure of the code can be exploited to derive several related decoding instances. The attacker is then allowed to solve any one of them. This leads to the following *Decoding-Out-Of-Many* problem[5]:

Problem 2.2 (Decoding-Out-Of-Many $\mathsf{DOOM}_q(n,k,t)$**).** *Let* $\mathcal{C}$ *be a* $[n,k]_q$ *linear code, let* $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k \in \mathbb{F}_q^n$ *be vectors of Hamming weight* t*, and let* $\boldsymbol{y}_i \in \mathcal{C} + \boldsymbol{e}_i$ *for each* $i \in [\![1,k]\!]$*. Given* $(\mathcal{C}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)$*, find one pair* $(i, \boldsymbol{e}_i)$*.*

We will also consider regular variants of the Decoding and DOOM problems, in which the target error vectors are split into d blocks, each containing the same number of nonzero entries:

Problem 2.3 (d**-**$\mathsf{Dec}_q(n,k,t)$ **and** d**-**$\mathsf{DOOM}_q(n,k,t)$**).** *Assume that* d *divides both* n *and* t*. For each* $j \in [\![1,d]\!]$*, define* $[j] \stackrel{\text{def}}{=} [\![(j-1)n/d + 1, jn/d]\!]$*. The problems* d-$\mathsf{Dec}_q(n,k,t)$ *and* d-$\mathsf{DOOM}_q(n,k,t)$ *are the variants of* $\mathsf{Dec}_q(n,k,t)$ *and* $\mathsf{DOOM}_q(n,k,t)$ *in which the target error vectors are additionally constrained to have the same Hamming weight in each of the* d *blocks defined above:*

- *in* d-$\mathsf{Dec}_q(n,k,t)$*, the error vector* $\boldsymbol{e}$ *satisfies* $\left|\boldsymbol{e}_{[j]}\right| = \frac{t}{d}$ *for all* $j \in [\![1,d]\!]$*;*
- *in* d-$\mathsf{DOOM}_q(n,k,t)$*, each error vector* $\boldsymbol{e}_i$ *satisfies* $\left|\boldsymbol{e}_{i,[j]}\right| = \frac{t}{d}$ *for all* $i \in [\![1,k]\!]$ *and all* $j \in [\![1,d]\!]$*.*

The above problems can be attacked using the following basic observation: once the entries of an error vector on an information set are known, all its remaining entries can be recovered by linear algebra. This idea is the starting point of *Information Set Decoding* algorithms.

[5] In this work, we define DOOM with exactly k instances, as this is the setting relevant to our analysis. In the literature, however, DOOM is often formulated with an exponential number of instances, notably in the context of forgery attacks against signature schemes.

Definition 2.2 (Information Set). *Let $\mathcal{C}$ be an $[n,k]_q$ linear code. A subset $I \subseteq [\![1,n]\!]$ of size k is called an* information set *if*

$$\mathcal{C}_I \stackrel{\text{def}}{=} \{\boldsymbol{c}_I \ : \ \boldsymbol{c} \in \mathcal{C}\} = \mathbb{F}_q^k. \tag{2.1}$$

Equivalently, if $\mathbf{G}$ is a generator matrix of $\mathcal{C}$, then I is an information set if and only if $\mathbf{G}_{[I:]}$ is invertible.

Prange's algorithm [36] can be seen as the precursor of Information Set Decoding (ISD) algorithms. It consists in choosing uniformly at random an information set $I \subseteq [\![1,n]\!]$ and making the bet that the target error vector $\boldsymbol{e}$ has no support on this set, namely $\boldsymbol{e}_I = \mathbf{0}$. If this guess is correct, then the remaining part $\boldsymbol{e}_{\overline{I}}$ can be recovered by simple linear algebra:

$$\boldsymbol{e}_{\overline{I}} \ = \ \boldsymbol{y}_{\overline{I}} - \mathbf{G}_{[\overline{I}:]}\mathbf{G}_{[I:]}^{-1}\boldsymbol{y}_I \tag{2.2}$$

Thus, for a given information set I, one obtains a candidate error vector by setting $\boldsymbol{e}_I = \mathbf{0}$ and computing $\boldsymbol{e}_{\overline{I}}$ as above. To check whether the guess was correct, it is enough to verify that this candidate has Hamming weight t. If this is not the case, the procedure is repeated with a new information set I until a valid solution is found.

Remark 2.3. To avoid overloading the presentation, we will often refrain from spelling out the d-regular variants of our algorithms. Most ISD-type algorithms admit a natural adaptation to the d-regular setting: the information set is chosen so as to respect the block structure, and the corresponding Prange-type bets are imposed regularly across the d blocks of indices.

2.3 Branching Processes in a Varying Environment

The Reduce-and-Prange algorithm [30], which will be discussed later, improves upon Prange's algorithm by factoring parts of the Gaussian elimination step. Roughly speaking, it proceeds recursively by making partial Prange bets: at a given level, the algorithm tries several small sets of positions and bets that the error vector vanishes on these positions. Whenever such a bet is correct, the original decoding problem can be reduced to a smaller decoding problem, to which the same strategy is then applied recursively.

The analysis of this recursive structure can be refined beyond the treatment of [30] by using classical probabilistic tools. Indeed, the algorithm naturally gives rise to a tree of partial bets: successful bets generate new subproblems, whereas unsuccessful branches die out. Such a structure is naturally modeled by a *branching process*; see, for instance, the classical references [29,3]. Moreover, since the distribution governing the number of surviving branches may vary from one reduction level to the next, the appropriate framework is that of branching processes in varying environments.

In this section, we formally define these processes and recall the classical results that will be used in the sequel. They will provide the main probabilistic

tool for analyzing the Reduce-and-Prange algorithm [30], as well as our improved variant.

We start with the relevant definitions:

Definition 2.4 (Generation-dependent offspring distributions). *For every generation $i \geq 1$, let $(p_k^{(i)})_{k\geq 0}$ be a sequence of non-negative real numbers such that $\sum_{k=0}^{\infty} p_k^{(i)} = 1$. A random variable $X^{(i)}$ taking values in $\mathbb{N}$ and satisfying*

$$\mathbb{P}(X^{(i)} = k) = p_k^{(i)}, \qquad k \in \mathbb{N}, \tag{2.3}$$

is called the offspring distribution *at generation i.*

Definition 2.5 (Generating functions). *For every generation $i \geq 1$, the probability* generating function *associated with the offspring distribution at generation i is defined by*

$$f_i(s) = \mathbb{E}\left(s^{X^{(i)}}\right) = \sum_{k=0}^{\infty} p_k^{(i)} s^k, \qquad s \in [0,1]. \tag{2.4}$$

Definition 2.6 (Branching process in varying environment). *Let $\left(X_j^{(i)}\right)_{\substack{i\geq 1\\ j\geq 1}}$ be a family of independent random variables such that, for every fixed generation i, the random variables $\left(X_j^{(i)}\right)_{j\geq 1}$ are identically distributed with common distribution $X^{(i)}$. Given an initial population size $Z_0 \in \mathbb{N}$, the associated* branching process *$(Z_i)_{i\geq 0}$ is defined recursively by*

$$Z_{i+1} = \sum_{j=1}^{Z_i} X_j^{(i+1)}. \tag{2.5}$$

The random variable Z_i represents the population size at generation i.

Definition 2.7 (Survival probability). *For every $i \geq 0$, we call $\mathbb{P}(Z_i > 0)$ the survival probability at generation i. We also call $\mathbb{P}(Z_i = 0) = 1 - \mathbb{P}(Z_i > 0)$ the extinction probability at generation i.*

We now recall some useful properties of the extinction probability. First, the survival probability can be expressed in terms of the offspring generating functions:

Proposition 2.8. *Assume that $Z_0 = 1$. Then, for every $i \geq 1$,*

$$\mathbb{P}(Z_i > 0) = 1 - f_1 \circ f_2 \circ \cdots \circ f_i(0). \tag{2.6}$$

Proof. From [1], we have that the generating function of Z_i is

$$g_i(s) \stackrel{\text{def}}{=} \Big(f_1 \circ f_2 \circ \cdots \circ f_i(s)\Big)^{Z_0}, \qquad s \in [0,1]$$

So with $Z_0 = 1$, we have

$$\mathbb{P}(Z_i = 0) = g_i(0) = f_1 \circ f_2 \circ \cdots \circ f_i(0). \qquad \square$$

Multitype Branching Processes in Varying Environments. In our setting, the branches generated by the algorithm may have different types, and their offspring distributions may depend on these types. To capture this additional heterogeneity, we use the framework of *multitype branching processes in varying environments*, where individuals are assigned types and the offspring distribution may depend both on the generation and on the type of the parent; see, for instance, [3, Chapter 5] or [33, Section 2].

Definition 2.9 (Multitype branching process in varying environment). *Let $\mathcal{P}$ be a finite set of types. A* multitype branching process in varying environment *is a process $(\boldsymbol{Z}_i)_{i\geq 0}$ where $\boldsymbol{Z}_i = (Z_i(\tau))_{\tau\in\mathcal{P}}$ and $Z_i(\tau)$ denotes the number of individuals of type τ at generation i.*

For every generation $i \geq 1$ and every type $\tau \in \mathcal{P}$, let $\boldsymbol{X}^{(i,\tau)} = \left(X^{(i,\tau)}(\sigma)\right)_{\sigma\in\mathcal{P}}$ be a random vector, where $X^{(i,\tau)}(\sigma)$ denotes the number of children of type σ produced at generation i by one individual of type τ at generation $i-1$.

The process is defined by the initial condition $\boldsymbol{Z}_0$ and, for every $i \geq 0$ and every $\sigma \in \mathcal{P}$, $Z_{i+1}(\sigma) = \sum_{\tau\in\mathcal{P}} \sum_{j=1}^{Z_i(\tau)} X_j^{(i+1,\tau)}(\sigma)$, where, for every fixed i and τ, the random vectors $(\boldsymbol{X}_j^{(i,\tau)})_{j\geq 1}$ are independent copies of $\boldsymbol{X}^{(i,\tau)}$, and all these vectors are mutually independent.

Definition 2.10 (Offspring generating functions). *For every generation $i \geq 1$, let $\boldsymbol{X}^{(i)} = (X_1^{(i)}, \cdots, X_m^{(i)})$ be a random vector taking values in $\mathbb{N}^m$.*

The generating function associated with the transition from generation $i-1$ to generation i is the function $F_i : [0,1]^m \to \mathbb{R}$ defined by

$$F_i(s_1, \ldots, s_m) = \mathbb{E}\left(\prod_{j=1}^{m} s_j^{X_j^{(i)}}\right). \tag{2.7}$$

Proposition 2.11. *Let $(\boldsymbol{Z}_i)_{i\geq 0}$ be a multitype branching process in varying environment with finite type set $\mathcal{P}$. For every $i \geq 1$ and every $\tau \in \mathcal{P}$, let $F_i^{(\tau)}$ be the generating function of the offspring vector $\boldsymbol{X}^{(i,\tau)}$ produced by an individual of type τ at generation $i-1$.*

Fix a starting type $\tau_s \in \mathcal{P}$ and a target type $\tau_t \in \mathcal{P}$. If the process starts from one individual of type τ_s, namely if $\boldsymbol{Z}_0 = \boldsymbol{e}_{\tau_s}$ where $\boldsymbol{e}_{\tau_s}$ denotes the τ_s-th vector of the canonical basis, the probability of having no individual of type τ_t at generation i is

$$\mathbb{P}\left(Z_i(\tau_t) = 0 \mid \boldsymbol{Z}_0 = \boldsymbol{e}_{\tau_s}\right) = q_0(\tau_s), \tag{2.8}$$

where $(q_j(\tau))_{0\leq j\leq i}$ is defined recursively by

$$q_i(\tau) \stackrel{\text{def}}{=} 1 - \mathbb{1}_{\tau=\tau_t}, \tag{2.9}$$

$$q_{j-1}(\tau) \stackrel{\text{def}}{=} F_j^{(\tau)}\big((q_j(\sigma))_{\sigma\in\mathcal{P}}\big), \quad \text{for } 1 \leq j \leq i. \tag{2.10}$$

Proof. We prove by backward induction that, for every $j \in [\![0, i]\!]$ and every $\tau \in \mathcal{P}$, the quantity $q_j(\tau)$ is the probability that, starting from a single individual of type τ at generation j, no individual of type τ_t appears at generation i.

At the terminal generation i, this probability is $q_i(\tau) = 1 - \mathbb{1}_{\tau=\tau_t}$. Indeed, if $\tau = \tau_t$, then the initial individual itself is of type τ_t at generation i, so the probability of having no individual of type τ_t is 0; otherwise, it is 1.

Now consider a single individual of type τ at generation $j-1$, with $1 \leq j \leq i$. Its offspring vector at generation j is $\boldsymbol{X}^{(j,\tau)} = \left(X^{(j,\tau)}(\sigma)\right)_{\sigma\in\mathcal{P}}$ where $X^{(j,\tau)}(\sigma)$ denotes the number of children of type σ. Conditionally on this offspring vector, the descendants of all children evolve independently. Therefore, the probability that none of these descendants gives rise to an individual of type τ_t at generation i is

$$\prod_{\sigma\in\mathcal{P}} q_j(\sigma)^{X^{(j,\tau)}(\sigma)}.$$

Taking expectation over the offspring vector gives

$$q_{j-1}(\tau) = \mathbb{E}\left(\prod_{\sigma\in\mathcal{P}} q_j(\sigma)^{X^{(j,\tau)}(\sigma)}\right) = F_j^{(\tau)}\big((q_j(\sigma))_{\sigma\in\mathcal{P}}\big). \qquad \square$$

2.4 Multinomial Distribution

The offspring distributions considered in our branching processes will be binomial distributions. This will allow us to analyze the Reduce-and-Prange algorithm and our variant. However, in Section 5, we will also generalize this approach to Stern's algorithm. In that setting, we will use multitype branching processes, whose offspring distributions are naturally described by multinomial distributions.

The multinomial distribution generalizes the binomial distribution to the case where each trial may result in one of several possible outcomes. More precisely, the random experiment consists of r independent trials: at each trial, outcome i occurs with probability p_i, and X_i counts the number of times outcome i occurs. The distribution of $\boldsymbol{X}$ is given as follows:

Definition 2.12 (Multinomial distribution). *Let $m, r \in \mathbb{N}^*$, and let $\boldsymbol{p} \in [0,1]^m$ be such that $\sum_{i=1}^m p_i = 1$. A random vector $\boldsymbol{X} \in \mathbb{N}^m$ is said to have a* multinomial distribution *with parameters r and $\boldsymbol{p}$, written $\boldsymbol{X} \sim \mathrm{Multinomial}(r;\boldsymbol{p})$, if, for every $\boldsymbol{x} \in \mathbb{N}^m$ such that $\sum_{i=1}^m x_i = r$, we have*

$$\mathbb{P}(\boldsymbol{X} = \boldsymbol{x}) = \frac{r!}{x_1! \cdots x_m!} \prod_{i=1}^m p_i^{x_i}. \tag{2.11}$$

The generating function of $\boldsymbol{X}$ in $\boldsymbol{s} = (s_1, \ldots, s_m)$ is given by

$$F(\boldsymbol{s}) = \mathbb{E}\left(\prod_{i=1}^m s_i^{X_i}\right) = \left(\sum_{i=1}^m p_i s_i\right)^r. \tag{2.12}$$

2.5 Computational Models

We introduce a common notation for the elementary operations appearing in the complexity analyses of this paper. The cost assigned to each operation depends on the computational model under consideration and is specified below.

For q a prime power, we use the following cost functions:

- $\mathsf{VAdd}_q(n)$ and $\mathsf{VProd}_q(n)$ denote the costs of the componentwise addition and multiplication, respectively, of two vectors in $\mathbb{F}_q^n$.
- $\mathsf{RAM}_q(b, N)$ denotes the cost of a random-access query that reads or writes b consecutive symbols in a memory containing N symbols.
- $\mathsf{SVRAM}_q(n, t, M)$ denotes the cost of performing a read or write random-access query on a sparse vector of length n and Hamming weight t stored in a list containing M such sparse vectors. If the vectors are stored in a table whose cells contain T sparse vectors on average, then the table contains $\frac{M}{T}$ cells on average. Hence, the cost of reading or writing all the sparse vectors contained in a randomly accessed cell is $\mathsf{SVRAM}_q\left(n, Tt, \frac{M}{T}\right)$.
- $\mathsf{SVAdd}_q(n, t)$ and $\mathsf{SVProd}_q(n, t)$ denote the costs of an addition and a multiplication, respectively, involving sparse vectors of length n and Hamming weight at most t.
- $\mathsf{MSVProd}_q(m, n, t)$ denotes the cost of multiplying a matrix in $\mathbb{F}_q^{m\times n}$ by a sparse vector of Hamming weight t, while $\mathsf{MVProd}_q(m, n)$ denotes the corresponding cost for a dense vector.
- $\mathsf{GAUSS}_q(m, n, \ell)$ denotes the cost of performing a partial Gaussian elimination with ℓ pivots on the rows of a matrix in $\mathbb{F}_q^{m\times n}$.

We now instantiate these functions in the two computational models used in this work.

Logic-Gate Model for the Calibration of HQC. We use a logic-gate computational model to assess the security of HQC against attacks on the d-$\mathsf{DOOM}_2(n, k, t)$ problem. Our goal is to obtain precise security estimates for HQC, in order to support its designers in preparing the specification of HQC as a new NIST standard. NIST specifies several security categories, including Categories I, III, and V, which are intended to match the security levels of AES-128, AES-192, and AES-256, respectively. It is commonly accepted in the cryptographic community that breaking AES-128, AES-192, and AES-256 requires approximately 2^{143}, 2^{207}, and 2^{272} elementary operations, respectively. However, to compare our estimates with these reference values, one must specify the underlying computational model. Indeed, security estimates are meaningful only when the cost of the considered attacks is measured in a model comparable to the one used for AES.

Following the recommendation of [6], all complexities in this model are therefore expressed as numbers of logic gates. Consequently, we must specify the gate-level cost of the operations that are used repeatedly in the attacks against HQC.

Model 2.1 (Logic-gate cost model). *The costs of the following operations are measured in numbers of logic gates. The elementary Boolean operations AND, OR, XOR, and NOT, as well as bit comparisons, are assigned unit cost. The cost functions introduced above are instantiated as follows:*

$$\mathsf{VProd}_2(n) = \mathsf{VAdd}_2(n) \stackrel{\text{def}}{=} n, \tag{2.13}$$

$$\mathsf{RAM}_2(b, N) \stackrel{\text{def}}{=} \log_2(N)\big(b + \log_2(N)\big), \tag{2.14}$$

$$\mathsf{SVRAM}_2(n, t, M) \stackrel{\text{def}}{=} \mathsf{RAM}_2\left(t\lceil\log_2(n)\rceil, Mt\lceil\log_2(n)\rceil\right), \tag{2.15}$$

$$\mathsf{SVProd}_2(n, t) = \mathsf{SVAdd}_2(n, t) \stackrel{\text{def}}{=} 2t\lceil\log_2(n)\rceil, \tag{2.16}$$

$$\mathsf{MSVProd}_2(m, n, t) \stackrel{\text{def}}{=} t\mathsf{RAM}_2(m, mn) + (t-1)m, \tag{2.17}$$

$$\mathsf{MVProd}_2(m, n) \stackrel{\text{def}}{=} m(2n-1), \tag{2.18}$$

$$\mathsf{GAUSS}_2(m, n, \ell) \stackrel{\text{def}}{=} \frac{(m-1)\ell}{2}\left(n - \frac{\ell-1}{2}\right). \tag{2.19}$$

When the partial Gaussian elimination is performed on the columns rather than on the rows, its cost is $\mathsf{GAUSS}_2(n, m, \ell)$.

We now briefly explain the rationale behind the cost estimates introduced in Model 2.1. The model is deliberately simple and aims at consistently counting the dominant elementary operations in terms of logic gates.

First, elementary Boolean operations are assigned unit cost because each of them can be implemented by a single logic gate. Similarly, comparing two bits is also assigned unit cost.

The cost $\mathsf{RAM}_2(b, N)$ models the overhead of accessing b consecutive bits in a memory containing N bits. Addressing such a memory requires $\log_2(N)$ bits. The factor $\log_2(N)$ accounts for the cost of the addressing and selection mechanism, while the term $b + \log_2(N)$ accounts for the fact that both the requested data and the address information must be routed through this mechanism[6].

A sparse vector of Hamming weight t in $\mathbb{F}_2^n$ is represented by the list of its t nonzero positions. Since each position requires $\log_2(n)$ bits, such a vector requires $t\lceil\log_2(n)\rceil$ bits of storage. A list of M such vectors is therefore viewed as a memory of size $Mt\lceil\log_2(n)\rceil$ bits, and retrieving one sparse vector from this list consists in querying $t\lceil\log_2(n)\rceil$ consecutive bits in this memory. Moreover, bitwise operations on two sparse vectors can be performed directly on their support representations. Since the two supports contain at most $2t$ stored positions in total, and each position is encoded using $\lceil\log_2(n)\rceil$ bits, this gives a simple upper bound for the cost of scanning and combining the two sparse supports.

[6] This estimate appears to be commonly considered in the context of the NIST post-quantum cryptography standardization process (https://csrc.nist.gov/projects/post-quantum-cryptography), as reflected in several discussions on the dedicated PQC forum: https://groups.google.com/a/list.nist.gov/g/pqc-forum.

For the matrix-vector product $\mathbf{M}\boldsymbol{x}$ with a sparse vector $\boldsymbol{x}$ of Hamming weight t, the product is computed as the sum of the t columns of $\mathbf{M}$ indexed by the support of $\boldsymbol{x}$. Each selected column has length m and is read from a memory of size mn, which costs $\mathsf{RAM}_2(m, mn)$ if the columns of $\mathbf{M}$ are laid out consecutively in memory. Reading the t selected columns therefore costs $t\mathsf{RAM}_2(m, mn)$. These t columns are then added over $\mathbb{F}_2$, requiring $(t-1)m$ bitwise XORs. When $\boldsymbol{x}$ is not assumed to be sparse, the product is instead evaluated as a dense matrix-vector multiplication. Each of the m output coordinates is computed as an inner product of length n over $\mathbb{F}_2$, requiring n AND gates and $n-1$ XOR gates.

Finally, a partial Gaussian elimination with ℓ pivots is estimated by counting the row additions performed during the elimination. At pivot step s, with $0 \le s < \ell$, the row operations affect the remaining $n-s$ entries. Since the entries are assumed to behave as random bits, approximately half of the other $m-1$ rows need to be updated. Hence the cost is $\sum_{s=0}^{\ell-1} \frac{m-1}{2}(n-s) = \frac{(m-1)\ell}{2}\left(n - \frac{\ell-1}{2}\right)$.

To avoid additional random-access costs, we assume that, when Gaussian elimination is performed on the rows, the matrix is stored with its rows laid out consecutively in memory. If the rows to be eliminated are not stored consecutively, we neglect the additive cost of rearranging the matrix, as this cost is significantly smaller than that of Gaussian elimination itself. The relevant entries can therefore be accessed sequentially. Similarly, when Gaussian elimination is performed on the columns, we assume that the matrix is stored with its columns laid out consecutively in memory. The corresponding cost is therefore obtained by exchanging the roles of rows and columns, namely $\mathsf{GAUSS}_2(n, m, \ell)$.

***q*-ary Operation-Counting Model for PCG Security Estimates.** For PCG security estimates, we use a simpler computational model in which complexities are expressed in numbers of arithmetic operations over $\mathbb{F}_q$. In contrast with the logic-gate model introduced above for the calibration of HQC, we do not assign any additional cost to memory accesses in this model. In particular, random-access queries do not incur a cost depending on the size of the memory. This choice is motivated by the fact that we are no longer aiming at a fine-grained comparison with exhaustive key-search attacks on AES, where gate-level estimates are needed. Instead, we use a model that provides a convenient and consistent basis for comparing our estimates with the existing literature on PCG protocols [46,30].

Model 2.2 (q-ary operation-counting model). *The costs of the following operations are measured in numbers of arithmetic operations over $\mathbb{F}_q$. Field additions, subtractions, multiplications, and inversions over $\mathbb{F}_q$ have unit cost. The cost functions introduced above are instantiated as follows:*

$$\mathsf{VProd}_q(n) = \mathsf{VAdd}_q(n) \stackrel{\text{def}}{=} n, \tag{2.20}$$

$$\mathsf{RAM}_q(b, N) \stackrel{\text{def}}{=} b, \tag{2.21}$$

$$\mathsf{SVRAM}_q(n, t, M) \stackrel{\text{def}}{=} \mathsf{RAM}_q(t, Mt) = t, \tag{2.22}$$

$$\mathsf{SVProd}_q(n,t) = \mathsf{SVAdd}_q(n,t) \stackrel{\text{def}}{=} 2t, \tag{2.23}$$

$$\mathsf{MSVProd}_q(m,n,t) \stackrel{\text{def}}{=} t\mathsf{RAM}_q(m,mn) + t\mathsf{VProd}_q(m) + (t-1)\mathsf{VAdd}_q(m), \tag{2.24}$$

$$\mathsf{MVProd}_q(m,n) \stackrel{\text{def}}{=} m\mathsf{VProd}_q(n) + (n-1)\mathsf{VAdd}_q(m), \tag{2.25}$$

$$\mathsf{GAUSS}_q(m,n,\ell) \stackrel{\text{def}}{=} \ell\left(1 + \left(1 + 2\left(1 - \frac{1}{q}\right)(m-1)\right)\left(n - \frac{\ell-1}{2}\right)\right). \tag{2.26}$$

When the elimination is performed on the columns of a matrix in $\mathbb{F}_q^{m\times n}$*, the corresponding cost is* $\mathsf{GAUSS}_q(n,m,\ell)$.

The rationale behind the cost estimates in Model 2.2 is essentially the same as in Model 2.1, except that we only count arithmetic operations over $\mathbb{F}_q$. In particular, we abstract away memory-access overheads, data movements, and bit-level implementations of field operations.

For Gaussian elimination, at pivot step s, with $0 \le s < \ell$, we count one inversion for the pivot and $n-s$ multiplications to normalize the pivot row. For each other row, the entry in the pivot column is nonzero with probability $1-1/q$, and eliminating it requires $2(n-s)$ field operations, namely one multiplication and one subtraction per affected coordinate. Summing these costs over the ℓ pivot steps gives the stated expression for

$$\mathsf{GAUSS}_q(m,n,\ell) \stackrel{\text{def}}{=} \sum_{s=0}^{\ell-1}\left(1 + n - s + 2(n-s)\left(1 - \tfrac{1}{q}\right)(m-1)\right),$$

which coincides with the expression given in Equation (2.26). When Gaussian elimination is performed on the columns instead of the rows, the same estimate applies, since the costs of random-access queries are ignored.

3 Relevant Attacks Against HQC and PCG

This section surveys the best-known combinatorial attacks against HQC and PCG. The techniques presented here are well established in the literature. However, they have not been considered together in the context of HQC and PCG, nor analyzed in the computational models that are most relevant for our security estimates.

For the classical asymptotic regime in which the field is $\mathbb{F}_2$ and t grows linearly with n, a long line of work has improved the exponential complexity of Information Set Decoding beyond Prange's algorithm [41,21,32,5,8,9,16,17]. However, HQC and PCG correspond to different parameter regimes: the target weight t is sublinear in n, and PCG is defined over large finite fields. As discussed in the introduction, in these regimes, asymptotic improvements over Stern's algorithm [41] at the exponential level are limited, whereas polynomial-cost operations, notably linear algebra, can have a significant impact on concrete

security estimates. This makes refinements targeting these polynomial terms particularly relevant, including Peters' tricks [34], Projective Stern [18], Progressive Sieving-ISD [28,45], and factored Gaussian elimination [13,7,30]. We review these techniques below, as they currently form the state of the art for attacks against HQC and PCG and serve as the basis for the improvements developed in the remainder of the paper.

3.1 Stern's Algorithm and Improvements.

Stern's algorithm [41] improves over Prange's by allowing a nonzero error of weight $2p$ on the information set and by leveraging the birthday speedup. More precisely, Stern's algorithm is parametrized by two positive integers $p \leq t/2$ and $\ell \in [\![1, n-k]\!]$. At each iteration, it chooses uniformly at random an information set $I \subseteq [\![1, n]\!]$ of size k, together with a subset $Z \subseteq \overline{I}$ of size ℓ. For any error vector $\boldsymbol{e}$ such that $\boldsymbol{y} - \boldsymbol{e} \in \mathcal{C}$, the restriction of $\boldsymbol{e}$ to Z satisfies

$$\begin{aligned} \boldsymbol{e}_Z &= \boldsymbol{y}_Z - \mathbf{G}_{[Z:]}\mathbf{G}_{[I:]}^{-1}(\boldsymbol{y}_I - \boldsymbol{e}_I) && (3.1)\\ &= \boldsymbol{y}'_Z + \mathbf{G}'_L\boldsymbol{e}_{I_L} + \mathbf{G}'_R\boldsymbol{e}_{I_R}, && (3.2) \end{aligned}$$

where $\boldsymbol{y}'_Z \stackrel{\text{def}}{=} \boldsymbol{y}_Z - \mathbf{G}_{[Z:]}\mathbf{G}_{[I:]}^{-1}\boldsymbol{y}_I$, $[\mathbf{G}'_L \mid \mathbf{G}'_R] = \mathbf{G}_{[Z:]}\mathbf{G}_{[I:]}^{-1} \in \mathbb{F}_q^{\ell \times k/2} \times \mathbb{F}_q^{\ell \times k/2}$, and I_L (resp. I_R) are the first (resp. the last) $k/2$ indices of I. Stern's algorithm guesses that $\boldsymbol{e}_Z = \mathbf{0}$ and $|\boldsymbol{e}_{I_L}| = |\boldsymbol{e}_{I_R}| = p$. Under this guess, a candidate pair $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}_q^{k/2} \times \mathbb{F}_q^{k/2}$ for $(\boldsymbol{e}_{I_L}, \boldsymbol{e}_{I_R})$ must satisfy

$$-\mathbf{G}'_L\boldsymbol{u} \;=\; \boldsymbol{y}'_Z + \mathbf{G}'_R\boldsymbol{v} \qquad (3.3)$$

The algorithm therefore searches for collisions between the two lists

$$\mathcal{L}_L \stackrel{\text{def}}{=} \left\{ -\mathbf{G}'_L\boldsymbol{u} \;:\; \boldsymbol{u} \in \mathbb{F}_q^{k/2} \text{ and } |\boldsymbol{u}| = p \right\} \qquad (3.4)$$

$$\mathcal{L}_R \stackrel{\text{def}}{=} \left\{ \boldsymbol{y}'_Z + \mathbf{G}'_R\boldsymbol{v} \;:\; \boldsymbol{v} \in \mathbb{F}_q^{k/2} \text{ and } |\boldsymbol{v}| = p \right\} \qquad (3.5)$$

Each collision gives a candidate error vector, obtained by setting its restrictions to I_L and I_R to the corresponding values $\boldsymbol{u}$ and $\boldsymbol{v}$, setting its restriction to Z to zero, and recovering the remaining coordinates in $\overline{I} \setminus Z$ by linear algebra. The candidate is then checked by verifying that it has Hamming weight t.

Peters' Tricks. Stern's algorithm can be accelerated using the implementation optimizations described by Peters [34]. The first observation is that, by enumerating vectors in revolving-door order, each element of $\mathcal{L}_L$ and $\mathcal{L}_R$ can be computed from the previous one using only one vector addition [35, Remark 6.3]. The second observation is that, when checking whether $\left|\boldsymbol{e}_{\overline{I}\setminus Z}\right| = t - 2p$, it is not necessary to compute the entire vector: on average, only $\frac{q}{q-1}(t-2p+1)$ coordinates have to be computed [35, Remark 6.4].

Adaptation to DOOM. Stern's algorithm can be adapted to solve Problem 2.2, where k decoding instances are given and the goal is to solve any one of them. To obtain the expected $1/\sqrt{k}$ speedup associated with the k instances of the DOOM problem [40], the approach proposed in [24] is to modify the second list $\mathcal{L}_R$, namely the list involving the noisy codeword, by taking all noisy codewords $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_k$ into account. In order to keep the two lists as balanced as possible, we compensate for this factor k by considering candidates of weight $p-\delta$ instead of p in the second list. Thus, the second list becomes

$$\mathcal{L}_R \stackrel{\text{def}}{=} \bigcup_{i=1}^{k} \left\{ (\boldsymbol{y}_i')_Z + \mathbf{G}_R' \boldsymbol{v} \ : \ \boldsymbol{v} \in \mathbb{F}_q^{k/2} \text{ and } |\boldsymbol{v}| = p - \delta \right\} \tag{3.6}$$

where $(\boldsymbol{y}_i')_Z \stackrel{\text{def}}{=} (\boldsymbol{y}_i)_Z - \mathbf{G}_{[Z:]} \mathbf{G}_{[I:]}^{-1} (\boldsymbol{y}_i)_I$. Since the list $\mathcal{L}_L$ remains unchanged, we naturally adapt the bet as follows: $|\boldsymbol{e}_{I_L}| = p$, $|\boldsymbol{e}_{I_R}| = p-\delta$ and $\left|\boldsymbol{e}_{\overline{I} \setminus Z}\right| = t-2p+\delta$. The parameter δ is chosen so as to make the two lists as balanced as possible, namely so that

$$k \binom{k/2}{p-\delta} (q-1)^{p-\delta} \approx \binom{k/2}{p} (q-1)^p. \tag{3.7}$$

In the binary case, taking $\delta = 1$ is the natural minimal modification and keeps the two lists balanced up to a small factor. By contrast, when q is large, decreasing the weight by one introduces an additional factor $(q-1)^{-1}$, which may overcompensate for the k instances. In that case, the most balanced choice can be $\delta = 0$.

Implementing Lookup Tables. Collisions between $\mathcal{L}_L$ and $\mathcal{L}_R$ are found using a lookup table indexed by the addressing function $@(\cdot)$ from the memory-access model. First, for each sparse vector $\boldsymbol{u} \in \mathbb{F}_q^{k/2}$ of weight p, we insert $\boldsymbol{u}$ at address $@(-\mathbf{G}_L' \boldsymbol{u})$. Then, for each noisy codeword $\boldsymbol{y}_i$ and each sparse vector $\boldsymbol{v} \in \mathbb{F}_q^{k/2}$ of weight $p-\delta$, we query the table at address $@(\boldsymbol{y}_i' + \mathbf{G}_R' \boldsymbol{v})$. A hit gives a candidate collision, which is then checked explicitly. The table only stores enough information to recover the corresponding $\boldsymbol{v}$ during the checking phase; the vectors $-\mathbf{G}_L' \boldsymbol{u} = \boldsymbol{y}_i' + \mathbf{G}_R' \boldsymbol{v}$ and $\boldsymbol{y}_i$ need not be stored in the lookup table, since they are already available when the candidate is checked.

Complexity of Stern's Algorithm in the DOOM Setting. We consider the DOOM variant of Stern's algorithm described above. Its expected complexity is given by the following proposition.

Proposition 3.1. *For internal parameters ℓ, p, and δ, the time complexity of Stern's algorithm for solving* $\mathsf{DOOM}_q(n,k,t)$ *is*

$$T_{\text{Stern}} \stackrel{\text{def}}{=} \frac{T_{\text{Gauss}} + T_{\text{list}} + k\left(T_{\boldsymbol{y}_Z'} + T_{\text{check}}\right)}{\mathbb{P}_{\text{success}}(\text{Stern})}. \tag{3.8}$$

Its memory complexity is

$$M_{\text{Stern}} = 2kn + M_{\mathcal{T}} \cdot p \left\lceil \log_q \left(\tfrac{k}{2}\right) \right\rceil \text{ symbols of } \mathbb{F}_q, \tag{3.9}$$

where

$$\begin{aligned}
M_{\mathcal{T}} &\stackrel{\text{def}}{=} \binom{k/2}{p}(q-1)^p, \\
M' &\stackrel{\text{def}}{=} \binom{k/2}{p-\delta}(q-1)^{p-\delta}, \\
\mathbb{P}_{\text{success}}(\text{Stern}) &\stackrel{\text{def}}{=} 1 - \left(1 - \frac{\binom{k/2}{p}\binom{k/2}{p-\delta}\binom{n-k-\ell}{t-(2p-\delta)}}{\binom{n}{t}}\right)^k, \\
T_{\text{Gauss}} &\stackrel{\text{def}}{=} \mathsf{GAUSS}_q\left(k, n, k\right), \\
T_{\text{list}} &\stackrel{\text{def}}{=} k\mathsf{VAdd}_q\left(\ell\right) + \left(M_{\mathcal{T}} - 1 + p\right)\mathsf{VAdd}_q\left(\ell\right) \\
&\qquad + M_{\mathcal{T}} \cdot \left(p\left\lceil \log_q\left(\tfrac{k}{2}\right)\right\rceil + \mathsf{SVRAM}_q\left(\tfrac{k}{2}, p, M_{\mathcal{T}}\right)\right), \\
T_{\boldsymbol{y}'_Z} &\stackrel{\text{def}}{=} \mathsf{MVProd}_q\left(\ell, k\right) + \mathsf{VAdd}_q\left(\ell\right), \\
T_{\text{check}} &\stackrel{\text{def}}{=} \left(M' + p - \delta\right)\mathsf{VAdd}_q\left(\ell\right) + M' \cdot \mathsf{SVRAM}_q\left(\tfrac{k}{2}, \tfrac{M_{\mathcal{T}}}{q^\ell}p, q^\ell\right) \\
&\qquad + \frac{M_{\mathcal{T}} \cdot M'}{q^\ell}\Bigg(\mathsf{MVProd}_q\left(\tfrac{q}{q-1}(t-2p+\delta+1), k\right) \\
&\qquad\qquad + \mathsf{MSVProd}_q\left(\tfrac{q}{q-1}(t-2p+\delta+1), k, 2p-\delta\right) \\
&\qquad\qquad + \mathsf{VAdd}_q\left(\tfrac{q}{q-1}(t-2p+\delta+1)\right)\Bigg).
\end{aligned}$$

Proof. First, we have to iterate the Stern procedure $\frac{1}{\mathbb{P}_{\text{success}}(\text{Stern})}$ times on average to succeed.

The cost of each iteration is essentially the sum of the Gaussian-elimination cost and the meet-in-the-middle cost. The former is given by $\mathsf{GAUSS}_q\left(k, n, k\right)$.

We now consider the meet-in-the-middle cost, beginning with the lookup-table construction cost T_{list}. The first term $k\mathsf{VAdd}_q\left(\ell\right)$ accounts for the preprocessing required by Peters' revolving-door enumeration technique. More precisely, this preprocessing consists in computing the sums of consecutive pairs of columns. It is performed for both matrices $\mathbf{G}'_L$ and $\mathbf{G}'_R$, since the latter is subsequently used during the right-list enumeration.

Using a revolving-door enumeration on all the weight-p vectors $\boldsymbol{u}$, one can compute all the $-\mathbf{G}'_L\boldsymbol{u}$ with p vector additions for the first element, then 1 vector addition for the remaining $M_{\mathcal{T}}-1$ elements. Since the columns of $\mathbf{G}'_L$ have length ℓ, the total cost of computing all the $-\mathbf{G}'_L\boldsymbol{u}$ is $\left(M_{\mathcal{T}} - 1 + p\right)\mathsf{VAdd}_q\left(\ell\right)$.

Each vector $\boldsymbol{u}$ is stored in a lookup table, denoted $\mathcal{T}$, using a sparse representation requiring $p\left\lceil \log_q\left(\frac{k}{2}\right)\right\rceil$ symbols of $\mathbb{F}_q$. Initializing the storage for all $M_{\mathcal{T}}$ entries therefore costs $M_{\mathcal{T}} \cdot p\left\lceil \log_q\left(\frac{k}{2}\right)\right\rceil$. Moreover, inserting a weight-p sparse

vector at a random address in the lookup table costs $\mathsf{SVRAM}_q\left(\frac{k}{2}, p, M_{\mathcal{T}}\right)$. Repeating this insertion for all $M_{\mathcal{T}}$ vectors gives the last contribution to T_{list}.

Once the lookup table has been constructed, it is reused for each of the k noisy codewords. For a fixed noisy codeword $\boldsymbol{y}_j$, the algorithm first computes $\boldsymbol{y}'_Z$. This requires one matrix-vector product and one vector addition, and hence costs $T_{\boldsymbol{y}'_Z}$. The algorithm then enumerates the M' vectors $\boldsymbol{v} \in \mathbb{F}_q^{k/2}$ of Hamming weight $p-\delta$ and computes $\boldsymbol{y}'_Z + \mathbf{G}'_R\boldsymbol{v}$. For the first vector $\boldsymbol{v}$, computing $\mathbf{G}'_R\boldsymbol{v}$ requires $p-\delta$ vector additions, followed by one additional vector addition to add $\boldsymbol{y}'_Z$. Using Peters' revolving-door enumeration technique, each of the remaining $M'-1$ vectors $\boldsymbol{z}$ can then be obtained from the preceding one using a single vector addition. The total cost of this enumeration is therefore $(M'+p-\delta)\,\mathsf{VAdd}_q(\ell)$.

Then, for each of the M' vectors $\boldsymbol{z}$, the algorithm reads the sparse vectors at address $\boldsymbol{z}$ in $\mathcal{T}$. On average, each table entry contains $\frac{M_{\mathcal{T}}}{q^\ell}$ weight-p sparse vectors. Therefore, reading the contents of $\mathcal{T}[\boldsymbol{z}]$ costs $\mathsf{SVRAM}_q\left(\frac{k}{2}, \frac{M_{\mathcal{T}}}{q^\ell}p, q^\ell\right)$.

There are $M_{\mathcal{T}} \cdot M'$ possible pairs $(\boldsymbol{u}, \boldsymbol{v})$. Since their corresponding addresses are uniformly distributed over the q^ℓ possible values, the expected number of collisions for a fixed noisy codeword is $\frac{M_{\mathcal{T}}M'}{q^\ell}$. For each collision, the algorithm computes $\boldsymbol{b} \stackrel{\text{def}}{=} \boldsymbol{y}_{\overline{I}} - \mathbf{G}_{[\overline{I}:]}\mathbf{G}_{[I:]}^{-1}\left(\boldsymbol{y}_I + \begin{pmatrix}\boldsymbol{u}\\ \boldsymbol{v}\end{pmatrix}\right)$ while simultaneously checking whether $|\boldsymbol{b}| = t - 2p + \delta$. By Peters' early-abort technique, an unsuccessful candidate can be rejected after computing $\frac{q}{q-1}(t-2p+\delta+1)$ coordinates of $\boldsymbol{b}$ on average. Computing these coordinates requires one dense matrix-vector product, one matrix/sparse-vector product involving a vector of weight $2p-\delta$, and one vector addition. Multiplying the cost of these operations by the expected number of collisions yields the last term of T_{check}.

The lookup table is constructed once at each information set selection, whereas the computation of $\boldsymbol{y}'_Z$ and the meet-in-the-middle search are repeated for each of the k noisy codewords. Hence, the total cost of the meet-in-the-middle attack is $T_{\text{list}} + k\left(T_{\boldsymbol{y}'_Z} + T_{\text{check}}\right)$.

Finally, the memory complexity follows from the storage required for the matrix, the k noisy codewords, and the lookup table $\mathcal{T}$. □

Working in Projective Space. An additional improvement to Stern's algorithm for large field sizes was proposed in [18]. This technique applies to low-weight codeword search rather than directly to low-weight error recovery. The first step is therefore to reduce the decoding problem to a low-weight codeword problem.

Theorem 3.2 ([18, Theorem 4.1]). *Any instance of* $\mathsf{Dec}_q(n,k,t)$ *can be reduced, on average over the code, to at most* $\frac{q}{q-1}$ *instances of* $\mathsf{Dec}_q(n,k+1,t)$ *with target word* $\boldsymbol{y}=\mathbf{0}$.

In the DOOM setting, this reduction would require augmenting the code with the k noisy codewords, leading to a code of dimension $2k$. For code rate $\frac{1}{2}$, the resulting code is the whole ambient space $\mathbb{F}_q^n$, where every vector of weight t is

a solution. The reduction is therefore uninformative in this case. Consequently, when using the projective-space technique, we restrict ourselves to the standard, possibly regular, Decoding problem and do not consider the DOOM setting.

To solve the reduced low-weight codeword problem, the idea is to work in the projective space $\mathbb{F}_q^n/_\sim$ where $\sim$ denotes the collinearity equivalence relation. We write $[\boldsymbol{x}] \stackrel{\text{def}}{=} \left\{\boldsymbol{y} \in \mathbb{F}_q^n \ : \ \boldsymbol{y} \sim \boldsymbol{x}\right\}$ for the equivalence class of $\boldsymbol{x}$. Instead of enumerating all weight-p vectors as in the classical Stern algorithm, the algorithm enumerates one representative per equivalence class of weight-p vectors. To make collision search compatible with this projective space, distinguished representatives $\overline{\boldsymbol{u}} \in [\boldsymbol{u}]$ and $\overline{\boldsymbol{v}} \in [\boldsymbol{v}]$ are defined so that

$$[\mathbf{G}'_L \boldsymbol{u}] = [\mathbf{G}'_R \boldsymbol{v}] \quad \Longleftrightarrow \quad \mathbf{G}'_L \overline{\boldsymbol{u}} = \mathbf{G}'_R \overline{\boldsymbol{v}} \tag{3.10}$$

Thus, a collision between projective classes yields an actual collision between the chosen representatives. As a consequence, both the number of vectors to enumerate and the number of collisions to check are reduced by a factor $q-1$ compared with the classical Stern algorithm. The technique recovers a nonzero scalar multiple of the desired error vector, from which the original error vector is easily recovered.

3.2 The Reduce-and-Prange Algorithm.

Stern's algorithm improves upon Prange's algorithm by using a birthday-paradox strategy to relax the condition imposed on the error vector inside the information set. Another natural way to improve upon Prange is to keep the same bet, namely $\boldsymbol{e}_I = \mathbf{0}$, while reducing the cost of each iteration. Since a Prange iteration is dominated by Gaussian elimination, whose cost is only polynomial, such an improvement does not change the exponential complexity. However, in the regime considered here, where $t = \mathrm{o}(n)$, linear-algebra costs are no longer negligible and reducing them can lead to significant practical speedups. The main difficulty is then to amortize Gaussian elimination without introducing excessive dependencies between successive trials.

A method introduced by Canteaut and Chabaud [13], and later refined by Bernstein, Lange and Peters [7], proposed such an improvement based on local updates of the information set. Starting from an initial information set I, each iteration replaces c columns of I with c columns outside I, chosen uniformly at random, while preserving the information-set property. The Gaussian elimination is then updated from the previous one rather than recomputed from scratch.

This procedure induces a random walk on the set of information sets. Its main benefit is to substantially reduce the linear-algebra cost per iteration. Its drawback is that consecutive information sets differ only slightly and are therefore strongly correlated. As a consequence, the corresponding success events are no longer independent, and more iterations may be required than in the idealized independent setting. The overall complexity is thus governed by a trade-off between cheaper iterations and a potentially larger number of trials.

Another way to amortize Gaussian elimination is provided by the recently proposed Reduce-and-Prange algorithm [30]. As described in Algorithm 3.1, Reduce-and-Prange builds the information set progressively through a sequence of N partial reductions. When $N = 1$, Reduce-and-Prange coincides with Prange's algorithm. For $N > 1$, it performs a sequence of partial Prange bets: at recursive level i, it bets that the current error vector vanishes on a block of b_i positions, and then uses this bet to reduce the current decoding problem to a smaller instance. Each partial bet is repeated several times before proceeding to the next reduction level. The resulting dependencies are therefore organized as a tree rather than arising from a local random walk on information sets.

Algorithm 3.1: Reduce-and-Prange [30, Algorithm 3]

Input: An instance $(\mathbf{G}, \boldsymbol{y})$ of $\mathsf{Dec}_q(n, k, t)$.
Parameters: Block sizes $(b_1, \dots, b_N)$ such that $\sum_{i=1}^{N} b_i = k$, and repetition numbers $(r_1, \dots, r_N)$.
Output: Either $\boldsymbol{e} \in \mathbb{F}_q^n$ such that $|\boldsymbol{e}| = t$ and $(\boldsymbol{y} - \boldsymbol{e}) \in \{\mathbf{G}\boldsymbol{u} : \boldsymbol{u} \in \mathbb{F}_q^k\}$, or $\perp$.

return *RAP-PROCEDURE*$(1, \mathbf{G}, \boldsymbol{y})$;

Function *RAP-PROCEDURE*$(i, \mathbf{G}', \boldsymbol{y}')$**:**
  $(n', k') \leftarrow \left(n - \sum_{j=1}^{i-1} b_j,\ k - \sum_{j=1}^{i-1} b_j\right)$;
  repeat r_i **times**
    choose uniformly at random $I \subseteq [\![1, n']\!]$ of size b_i such that $\mathbf{G}'_{[I:]}$ has rank b_i;
    run a partial Gaussian elimination to compute $\mathbf{A} \in GL_{k'}(\mathbb{F}_q)$, $\mathbf{B} \in \mathbb{F}_q^{(n'-b_i)\times b_i}$, and $\mathbf{G}'' \in \mathbb{F}_q^{(n'-b_i)\times(k'-b_i)}$ such that $\mathbf{G}'_{[I:]}\mathbf{A} = [\mathbf{Id}_{b_i} \mid \mathbf{0}]$ and $\mathbf{G}'_{[\overline{I}:]}\mathbf{A} = [\mathbf{B} \mid \mathbf{G}'']$;
    $\boldsymbol{y}'' \leftarrow \boldsymbol{y}'_{\overline{I}} - \mathbf{B}\boldsymbol{y}'_I$;
    `// Stop condition`
    if $i = N$ **then**
      if $|\boldsymbol{y}''| = t$ **then**
        return $\boldsymbol{e}' \in \mathbb{F}_q^{n'}$ *such that* $\boldsymbol{e}'_I = \mathbf{0}$ *and* $\boldsymbol{e}'_{\overline{I}} = \boldsymbol{y}''$;
    else
      `// Recursive call`
      $\boldsymbol{e}'' \leftarrow$ RAP-PROCEDURE$(i + 1, \mathbf{G}'', \boldsymbol{y}'')$;
      if $\boldsymbol{e}'' \neq \perp$ **then**
        return $\boldsymbol{e}' \in \mathbb{F}_q^{n'}$ *such that* $\boldsymbol{e}'_I = \mathbf{0}$ *and* $\boldsymbol{e}'_{\overline{I}} = \boldsymbol{e}''$;
  return $\perp$;

The time complexity of one execution of Algorithm 3.1 is

$$T_{\mathrm{RaP}} \stackrel{\mathrm{def}}{=} \sum_{i=1}^{N} \left(\prod_{j=1}^{i} r_j \right) \left(T_{\mathrm{Gauss}}(i) + T_{\boldsymbol{y}''}(i) \right), \tag{3.11}$$

where, for all $i \in [\![1, N]\!]$ and with $B_i \stackrel{\mathrm{def}}{=} \sum_{j=1}^{i} b_j$,

$$T_{\mathrm{Gauss}}(i) \stackrel{\mathrm{def}}{=} \mathsf{GAUSS}_q \left(k - B_{i-1}, n - B_{i-1}, b_i \right), \tag{3.12}$$

$$T_{\boldsymbol{y}''}(i) \stackrel{\mathrm{def}}{=} \mathsf{MVProd}_q \left(n - B_i, b_i \right) + \mathsf{VAdd}_q \left(n - B_i \right) \text{ if } i < N, \tag{3.13}$$

$$T_{\boldsymbol{y}''}(N) \stackrel{\mathrm{def}}{=} \mathsf{MVProd}_q \left(b_N, \tfrac{q}{q-1}(t+1) \right) + \mathsf{VAdd}_q \left(\tfrac{q}{q-1}(t+1) \right) \tag{3.14}$$

The last expression uses Peters' early-abort trick: at the final step, in an unsuccessful trial, it is enough to compute on average $\frac{q}{q-1}(t+1)$ coordinates of $\boldsymbol{y}''$ before detecting that its weight is larger than t.

In [30], the number of repetitions for each level i is set to

$$r_i = \left\lceil \frac{1}{P_i} \right\rceil \stackrel{\mathrm{def}}{=} \left\lceil \frac{\binom{n-B_{i-1}}{t}}{\binom{n-B_i}{t}} \right\rceil. \tag{3.15}$$

Ignoring integer-rounding effects, the number of leaves in the complete reduction tree, or equivalently the number of complete information sets tested, is therefore

$$\prod_{i=1}^{N} r_i \approx \frac{\binom{n}{t}}{\binom{n-B_1}{t}} \cdot \frac{\binom{n-B_1}{t}}{\binom{n-B_2}{t}} \cdots \frac{\binom{n-B_{N-1}}{t}}{\binom{n-B_N}{t}} = \frac{\binom{n}{t}}{\binom{n-k}{t}} \tag{3.16}$$

Thus, the number of leaves in a complete Reduce-and-Prange tree is essentially the same as the expected number of iterations of Prange's algorithm. However, the average linear-algebra cost per leaf is much smaller, since the computations performed at the first reduction levels are shared by many leaves. This sharing constitutes the main source of the amortization gain.

It remains to determine whether this gain is offset by the success probability of one execution, namely the probability that the algorithm does not return $\perp$. The analysis of [30, Section 2.2] estimates this probability by $(1 - e^{-1})^N$. The underlying intuition is that a reduction that succeeds with probability P_i, repeated $r_i = \lceil \frac{1}{P_i} \rceil$ times, succeeds at least once with probability

$$1 - (1 - P_i)^{r_i} \geq 1 - e^{-r_i P_i} \geq 1 - e^{-1}. \tag{3.17}$$

Multiplying this estimate over the N reduction levels leads to $(1 - e^{-1})^N$. However, this level-by-level independence heuristic does not fully capture the structure of the algorithm. The events occurring at different levels are coupled: the number of successful reductions available at level i depends on the number of successful reductions obtained at level $i - 1$. These dependencies may in fact be

favorable to the attacker: several successful partial bets can coexist across the reduction tree, and a failure along one branch does not preclude success along another.

Thus, in Section 4, we refine the analysis of Reduce-and-Prange by using a branching-process model that capture naturally the dependency structure. We prove that the success probability is bounded below by a quantity of order $\frac{1}{N}$, rather than by an exponentially small quantity as it is suggested in [30]. Then, we relax the parameter choice $r_i = \left\lceil \frac{1}{P_i} \right\rceil$ and optimize the repetition numbers according to the actual cost of the different reduction levels.

4 Refining Reduce-and-Prange

4.1 Branching-Process Analysis of Reduce-and-Prange

In Algorithm 3.1, the number of successful partial bets at each level can be represented by a branching process in a varying environment $(Z_i)_{0\le i\le N}$ defined by

$$Z_0 = 1, \qquad \text{and} \qquad Z_{i+1} = \sum_{j=1}^{Z_i} X_j^{(i+1)}, \qquad 0 \le i < N \tag{4.1}$$

where, for each $i \in [\![1, N]\!]$, the random variables $(X_j^{(i)})_{j\ge 1}$ are independent copies of $X^{(i)} \sim \text{Binomial}(r_i,\, P_i)$.

Intuitively, each individual at generation i represents a partial choice of the information set. Its offspring correspond to the successful continuations at the next reduction level, namely to the choices for which the current error vector vanishes on the newly selected part of the information set.

This branching-process viewpoint gives the following recursive expression for the success probability of Algorithm 3.1.

Proposition 4.1. *For each $i \in [\![1, N]\!]$, let P_i denote the partial success probability of the i-th reduction level in Algorithm 3.1, and choose the corresponding repetition number as*

$$r_i \overset{\text{def}}{=} \left\lceil \frac{1}{P_i} \right\rceil. \tag{4.2}$$

Then the success probability of Algorithm 3.1 satisfies

$$\mathbb{P}_{\text{success}}\,(RaP) = Q_1 \ge \frac{1 - e^{-1}}{N}. \tag{4.3}$$

where we define recursively

$$Q_N = 1 - (1 - P_N)^{r_N}, \tag{4.4}$$

$$Q_i = 1 - (1 - P_i Q_{i+1})^{r_i}, \qquad 1 \le i < N. \tag{4.5}$$

Proof. By construction of the branching-process model, the algorithm succeeds if and only if the process survives up to generation N. Hence

$$\mathbb{P}_{\text{success}}(\text{RaP}) = \mathbb{P}(Z_N > 0).$$

For every $i \in [\![1, N]\!]$, the offspring distribution $X^{(i)} \sim \text{Binomial}(r_i, P_i)$ has generating function

$$f_i(s) = (1 - P_i + P_i s)^{r_i}.$$

Therefore, by Proposition 2.8,

$$\mathbb{P}_{\text{success}}(\text{RaP}) = 1 - f_1 \circ f_2 \circ \cdots \circ f_N(0).$$

We can show by induction that, for all $i \in [\![1, N-1]\!]$,

$$1 - Q_1 = f_1 \circ f_2 \circ \cdots \circ f_i(1 - Q_{i+1}),$$

and so in particular, $Q_1 = 1 - f_1 \circ f_2 \circ \cdots \circ f_{N-1}(1 - Q_N) = \mathbb{P}_{\text{success}}(\text{RaP})$, since $1 - Q_N = f_N(0)$.

It remains to show that $\frac{1}{Q_1}$ is at most linear in N. Since $r_i = \left\lceil \frac{1}{P_i} \right\rceil$, we have $r_i P_i \geq 1$ for every i. So, on the one hand, we have

$$Q_N = 1 - (1 - P_N)^{r_N} \geq 1 - e^{-r_N P_N} \geq 1 - e^{-1}.$$

And on the other hand, we have, for $1 \leq i < N$,

$$Q_i = 1 - (1 - P_i Q_{i+1})^{r_i} \geq 1 - e^{-r_i P_i Q_{i+1}} \geq 1 - e^{-Q_{i+1}} \geq \frac{Q_{i+1}}{1 + Q_{i+1}}.$$

Equivalently,

$$\frac{1}{Q_i} \leq 1 + \frac{1}{Q_{i+1}}.$$

Iterating this inequality from $i = 1$ to $i = N - 1$ gives

$$\frac{1}{Q_1} \leq N - 1 + \frac{1}{Q_N} \leq N - 1 + \frac{1}{1 - e^{-1}} \leq \frac{N}{1 - e^{-1}}.$$ □

This analysis shows that the success probability estimate $(1 - e^{-1})^N$ used in [30] is pessimistic. While $(1 - e^{-1})^N$ decreases exponentially with the number of reduction levels, the actual success probability decreases only inversely proportionally to N.

4.2 Optimizing the Repetition Numbers

We now relax the choice $r_i = \left\lceil \frac{1}{P_i} \right\rceil$ made in [30], and for each level i, define $\lambda_i \stackrel{\text{def}}{=} r_i P_i$. In the branching-process interpretation, λ_i is the expected number of successful children produced by one node at level $i - 1$.

With the original choice $r_i = \left\lceil \frac{1}{P_i} \right\rceil$, one has $1 \leq \lambda_i < 1 + P_i$. Thus, ignoring integer-rounding effects, the original Reduce-and-Prange algorithm corresponds to the critical choice $\lambda_i \approx 1$.

Allowing arbitrary positive integers r_i enlarges the parameter space of the algorithm. Since the original choice remains admissible, the optimum over this enlarged parameter space cannot be worse than the complexity of the original Reduce-and-Prange algorithm.

The following proposition generalizes Proposition 4.1.

Proposition 4.2. *For each $i \in [\![1, N]\!]$, let P_i denote the partial success probability of the i-th reduction level in Algorithm 3.1, let r_i be the corresponding repetition number, and define*

$$\lambda_i \stackrel{\text{def}}{=} r_i P_i. \tag{4.6}$$

Assume that $\lambda_i > 0$ for every i. Then, the success probability of Algorithm 3.1 satisfies

$$\mathbb{P}_{\text{success}}(RaP) = Q_1 \geq \frac{1}{1 + \sum_{i=1}^{N} \prod_{j=1}^{i} \frac{1}{\lambda_j}}. \tag{4.7}$$

where the sequence $\{Q_i\}$ is defined recursively as in Equations (4.4) and (4.5).

Proof. The recursive expression for Q_1 follows from the same branching-process argument as in Proposition 4.1. We only prove the lower bound.

First,

$$Q_N \geq 1 - e^{-\lambda_N} \geq \frac{\lambda_N}{1 + \lambda_N},$$

and similarly, for $1 \leq i < N$,

$$Q_i \geq 1 - e^{-\lambda_i Q_{i+1}} \geq \frac{\lambda_i Q_{i+1}}{1 + \lambda_i Q_{i+1}}.$$

Therefore

$$\frac{1}{Q_N} \leq 1 + \frac{1}{\lambda_N} \qquad \text{and} \qquad \frac{1}{Q_i} \leq 1 + \frac{1}{\lambda_i Q_{i+1}}.$$

Iterating from $i = 1$ to $i = N - 1$ gives

$$\frac{1}{Q_1} \leq 1 + \sum_{i=1}^{N} \prod_{j=1}^{i} \frac{1}{\lambda_j}.$$

□

Choosing $(\lambda_i)_{i \in [\![1,N]\!]} \neq \mathbf{1}$ changes the success probability of one execution compared with the original Reduce-and-Prange algorithm. At the same time, it also changes the number $\prod_{i=1}^{N} r_i$ of complete information sets tested in a full execution tree.

Let R_{RaP} denote the expected number of complete information sets tested when independent executions of the algorithm are repeated until success. Under

the conservative assumption that the entire reduction tree is explored at each execution, we have

$$R_{\mathrm{RaP}} = \frac{\prod_{i=1}^{N} r_i}{\mathbb{P}_{\mathrm{success}}\,(\mathrm{RaP})}. \tag{4.8}$$

To preserve the main advantage of Prange's algorithm in terms of the number of complete information sets tested, we would like R_{RaP} to remain as close as possible to the expected number of Prange iterations, that is

$$R_{\mathrm{Prange}} = \frac{\binom{n}{t}}{\binom{n-k}{t}} = \prod_{i=1}^{N} \frac{1}{P_i} \approx \prod_{i=1}^{N} \left\lceil \frac{1}{P_i} \right\rceil. \tag{4.9}$$

Ignoring integer-rounding effects, the corresponding normalized overhead satisfies

$$\begin{aligned} \frac{R_{\mathrm{RaP}}}{R_{\mathrm{Prange}}} &= \frac{\prod_{i=1}^{N} r_i}{\mathbb{P}_{\mathrm{success}}\,(\mathrm{RaP}) \cdot \prod_{j=1}^{N} \frac{1}{P_j}} \\ &= \frac{\prod_{i=1}^{N} \lambda_i}{Q_1} \\ &\leq \left(\prod_{i=1}^{N} \lambda_i \right) \cdot \left(1 + \sum_{i=1}^{N} \prod_{j=1}^{i} \frac{1}{\lambda_j} \right) \\ &\leq \sum_{i=0}^{N} \prod_{j=i+1}^{N} \lambda_j \end{aligned} \tag{4.10}$$

We would therefore like the sum of suffix products in (4.10) to remain as close as possible to 1.

The smallest possible value is obtained by taking $r_i = 1$ at every level, or equivalently $\lambda_i = P_i$. In this case,

$$\mathbb{P}_{\mathrm{success}}\,(\mathrm{RaP}) \approx \frac{\binom{n-k}{t}}{\binom{n}{t}} \qquad \text{and} \qquad \prod_{i=1}^{N} r_i = 1 \tag{4.11}$$

The normalized overhead is then exactly 1. However, the reduction tree contains only one leaf, so no Gaussian elimination is shared between different complete information sets. The resulting algorithm is therefore essentially equivalent to Prange's algorithm.

Conversely, choosing larger repetition numbers increases the size of the reduction tree and makes it possible to amortize the Gaussian eliminations performed at the first levels, but it also increases the upper bound $\sum_{i=0}^{N} \prod_{j=i+1}^{N} \lambda_j$ on the normalized number of complete information sets tested. The repetition numbers must therefore balance two competing objectives: keeping the suffix-product sum small, while maintaining a sufficiently large number $\prod_{i=1}^{N} r_i$ of leaves in each execution tree to obtain a meaningful Gaussian elimination amortization.

5 Adapting Reduce-and-Prange to Stern's Algorithm

In this section, we extend the Reduce-and-Prange framework to Stern's ISD algorithm and to the DOOM setting. The purpose of this extension is to amortize the cost of Gaussian elimination across several iterations. Consequently, the approach is beneficial only when Gaussian elimination constitutes a significant fraction of the cost of an iteration.

This condition is automatically satisfied for Prange's algorithm, since each iteration essentially consists of performing Gaussian elimination. Stern's algorithm, however, additionally involves a meet-in-the-middle step. In the linear-weight regime, where t grows linearly with n, this step dominates the cost of an iteration, leaving little room for improvement through the amortization of Gaussian elimination. In the sublinear-weight regime, by contrast, the meet-in-the-middle step need not be dominant. Our approach is therefore particularly relevant in this setting.

5.1 Our MAGE-Stern Algorithm

In Reduce-and-Prange (Algorithm 3.1), the reduction performed at iteration i relies on the assumption that the error vector $\boldsymbol{e}$ vanishes on a prescribed set of b_i coordinates. These coordinates can consequently be eliminated, producing a smaller decoding instance.

Stern's algorithm follows a different strategy: it searches for an error vector whose restriction to the information set has a prescribed nonzero weight. Consequently, the reduction mechanism of Reduce-and-Prange cannot be applied directly. Nevertheless, incremental guesses on the error support can still be combined with partial Gaussian eliminations. This yields a new variant of Stern's algorithm based on Multilevel Amortized Gaussian Elimination, which we call *MAGE-Stern*. Its pseudocode is given in Algorithm 5.1.

Remark 5.1. As discussed in the paragraph *"Adaptation to DOOM"* of Subsection 3.1, the optimal value of δ in Algorithm 5.1 is typically either 0 or 1: it is equal to 1 for small values of q and tends to 0 as q increases.

Implementation Optimizations. Several standard implementation techniques can be used to reduce the cost of the meet-in-the-middle phase performed between Lines 11 and 25. First, when constructing the lookup table at Lines 16, the vectors $-\mathbf{G}'_L\boldsymbol{u}$ (and subsequently the vectors $\boldsymbol{z}$) need not be used directly as table addresses. Instead, an addressing hash function @ may be applied to compress the address space and avoid an under-populated lookup table. The additional collisions introduced by this compression do not affect correctness, since every candidate obtained from the table is subsequently verified. Moreover, the cost of applying this addressing function is counted in the computational Model 2.1.

In addition, using Peters' revolving-door enumeration technique, the vectors $-\mathbf{G}'_L\boldsymbol{u}$ at Line 16 and $\boldsymbol{y}'_Z + \mathbf{G}'_R\boldsymbol{v}$ at Line 21 can be updated using a single column addition between two consecutive fixed-weight vectors.

Algorithm 5.1: MAGE-Stern

Input: An instance $(\mathbf{G}, (\boldsymbol{y}_j)_{j\in[\![1,k]\!]})$ of $\mathsf{DOOM}_q(n,k,t)$.
Parameters: a number $N \in \mathbb{N}^*$ of blocks, block sizes $(b_1, \cdots, b_N)$ such that $\sum_{i=1}^{N} b_i = k$, repetition numbers $(r_1, \ldots, r_N, r_{N+1})$, an integer $\ell \in [\![1, n-k]\!]$, and two integers δ and p such that $\delta \leq p \leq t$.
Output: Either $(\boldsymbol{e}, j) \in \mathbb{F}_q^n \times [\![1,k]\!]$ such that $|\boldsymbol{e}| = t$ and $(\boldsymbol{y}_j - \boldsymbol{e}) \in \{\mathbf{G}\boldsymbol{u} : \boldsymbol{u} \in \mathbb{F}_q^k\}$, or $\perp$.

return *MAGE-Stern-procedure*$(1, \mathbf{G}, \emptyset)$;

Function *MAGE-Stern-procedure*$(i, \mathbf{G}', I)$**:**
  $\mu \leftarrow \sum_{j=1}^{i-1} b_j$;
  $(J, \overline{J}) \leftarrow ([\![1, \mu]\!], [\![\mu+1, k]\!])$;
  repeat r_i **times**
    choose uniformly at random $I_i \subseteq \overline{I}$ such that $\operatorname{rank}\left(\mathbf{G}'_{[I_i:\overline{J}]}\right) = |I_i| = b_i$;
    $I \leftarrow I \cup I_i$;
    $\mathbf{G}' \leftarrow \mathbf{G}'\mathbf{A}$ where $\mathbf{A} \in GL_k(\mathbb{F}_q)$ is such that $\mathbf{A}_{[J:]} = [\mathbf{Id}_\mu | \mathbf{0}]$ and $\mathbf{G}'_{[I_i:]}\mathbf{A}_{[:J]} = \mathbf{0}$ and $\mathbf{G}'_{[I_i:]}\mathbf{A}_{[:\overline{J}]} = [\mathbf{Id}_{b_i} \mid \mathbf{0}]$;
    `// Stop condition`
    if $i = N$ **then**
      repeat r_{N+1} **times**
        choose uniformly at random $Z \in \overline{I}$ such that $|Z| = \ell$;
        choose uniformly at random a balanced bipartition $I_L \sqcup I_R$ of $[\![1,k]\!]$; `/* it can be different from` $[\![1, \frac{k}{2}]\!] \sqcup [\![\frac{k}{2}+1, k]\!]$ `*/`
        $(\mathbf{G}'_L, \mathbf{G}'_R) \leftarrow \left(\mathbf{G}'_{[Z:I_L]}, \mathbf{G}'_{[Z:I_R]}\right)$;
        initialize a lookup table $\mathcal{T}$;
        for $\boldsymbol{u} \in \mathbb{F}_q^{k/2}$ *such that* $|\boldsymbol{u}| = p$ **do**
          append $\boldsymbol{u}$ in $\mathcal{T}\left[-\mathbf{G}'_L\boldsymbol{u}\right]$;
        for $j = 1$ **to** k **do**
          $\boldsymbol{y} \leftarrow \boldsymbol{y}_j$;
          $\boldsymbol{y}'_Z \leftarrow \boldsymbol{y}_Z - \mathbf{G}'_{[Z:]}\boldsymbol{y}_I$
          for $\boldsymbol{v} \in \mathbb{F}_q^{k/2}$ *such that* $|\boldsymbol{v}| = p - \delta$ **do**
            $\boldsymbol{z} \leftarrow \boldsymbol{y}'_Z + \mathbf{G}'_R\boldsymbol{v}$;
            for $\boldsymbol{u} \in \mathcal{T}\left[\boldsymbol{z}\right]$ **do**
              $\boldsymbol{r} \leftarrow \boldsymbol{y}_{\overline{I}} - \mathbf{G}'_{[\overline{I}:]}\boldsymbol{y}_I + \mathbf{G}'_{[\overline{I}:I_L]}\boldsymbol{u} + \mathbf{G}'_{[\overline{I}:I_R]}\boldsymbol{v}$;
              if $|\boldsymbol{r}| = t - 2p + \delta$ **then**
                return $(\boldsymbol{e}, j) \in \mathbb{F}_q^n \times [\![1,k]\!]$ *such that* $(\boldsymbol{e}_I)_{I_L} = \boldsymbol{u}$ *and* $(\boldsymbol{e}_I)_{I_R} = \boldsymbol{v}$ *and* $\boldsymbol{e}_{\overline{I}} = \boldsymbol{r}$;
    else
      `// Recursive call`
      return *MAGE-Stern-procedure*$(i+1, \mathbf{G}', I)$;
  return $\perp$;

Finally, Peters' early-abort technique can be applied to the computation of $\boldsymbol{r}$ at Line 23. More precisely, it suffices to first compute $\frac{q}{q-1}(t-2p+\delta+1)$ coordinates of $\boldsymbol{r}$ and to compute the remaining coordinates only when the partial weight test succeeds. Moreover, for a fixed noisy codeword $\boldsymbol{y}_j$, the term $\boldsymbol{y}_{\overline{I}} - \mathbf{G}'_{[\overline{I}:]}\boldsymbol{y}_I$ at Line 23 may also be computed once and reused for all colliding pairs $(\boldsymbol{u}, \boldsymbol{v})$. The candidate-dependent part of the computation then essentially reduces to a product between $\left[\mathbf{G}'_{[:I_L]} \middle| \mathbf{G}'_{[:I_R]}\right]$ and the sparse vector $\begin{pmatrix}\boldsymbol{u}\\ \boldsymbol{v}\end{pmatrix}$. Since this precomputed term is needed only when a collision occurs, it may be evaluated lazily upon the first collision and then cached for any subsequent collisions associated with the same noisy codeword. However, since fewer than one collision per noisy codeword is expected on average, recomputing $\boldsymbol{y}_{\overline{I}} - \mathbf{G}'_{[\overline{I}:]}\boldsymbol{y}_I$ whenever a collision occurs is, on average, no more expensive than computing it upon the first collision and caching it for subsequent ones.

Limitations and Possible Further Amortization. Once the parameters have been optimized so that Gaussian elimination is no longer the dominant operation, further amortizing its cost provides little additional improvement. One might instead attempt to amortize the meet-in-the-middle computations across several iterations, in the same way as the Gaussian elimination steps. This is not straightforward, however, because the corresponding meet-in-the-middle instance depends on the complete information set. More precisely, once the information set I has been constructed, the meet-in-the-middle procedure seeks a vector $\boldsymbol{x} \in \mathbb{F}_q^k$ of weight $|\boldsymbol{x}| = 2p-\delta$ satisfying $\mathbf{G}_{[Z:]}\mathbf{G}_{[I:]}^{-1}\boldsymbol{x} = \mathbf{G}_{[Z:]}\mathbf{G}_{[I:]}^{-1}(\boldsymbol{y}_j)_I - (\boldsymbol{y}_j)_Z$ for some $j \in [\![1, k]\!]$. Both the reduced matrix and the target vector in this equation depend on the entire information set I. By contrast, the partial Gaussian elimination associated with a block I_i can be performed without knowing the subsequent blocks I_j, for $j > i$. Consequently, the meet-in-the-middle computation cannot be decomposed into independent operations associated with the successive blocks, and its intermediate computations cannot be directly reused across the corresponding branches.

5.2 Complexity of the MAGE-Stern Algorithm

The complexity of Algorithm 5.1, including the implementation optimizations described above, is given in Proposition 5.2.

Proposition 5.2. *The time complexity of one execution of Algorithm 5.1 is*

$$T_{\text{MAGE-Stern}} \stackrel{\text{def}}{=} \sum_{i=1}^{N} \left(\prod_{j=1}^{i} r_j\right) T_{\text{Gauss}}(i) + \left(\prod_{i=1}^{N+1} r_i\right)\left(T_{\text{list}} + k\left(T_{\boldsymbol{y}'_Z} + T_{\text{check}}\right)\right). \tag{5.1}$$

Its memory complexity is

$$M_{\text{MAGE-Stern}} = 2kn + M_{\mathcal{T}} \cdot p\left\lceil \log_q\left(\tfrac{k}{2}\right)\right\rceil \text{ symbols of } \mathbb{F}_q, \tag{5.2}$$

where

$$M_{\mathcal{T}} \stackrel{\text{def}}{=} \binom{k/2}{p}(q-1)^p,$$
$$M' \stackrel{\text{def}}{=} \binom{k/2}{p-\delta}(q-1)^{p-\delta},$$
$$T_{\text{Gauss}}(i) \stackrel{\text{def}}{=} \mathsf{GAUSS}_q\left(k, n-\sum_{j=1}^{i-1} b_j, b_i\right),$$

and T_{list}, $T_{\boldsymbol{y}'_Z}$ *and* T_{check} *are defined as in Proposition 3.1.*

Proof. The execution of Algorithm 5.1 can be represented by a rooted tree. For all $i \in [\![1, N+1]\!]$, every node at level $i-1$ gives rise to r_i nodes at level i. Consequently, the number of nodes at level i is $\prod_{j=1}^{i} r_j$

For each $i \in [\![1, N]\!]$, the algorithm performs a partial Gaussian elimination at every node of level i. At this stage, the first $\mu \stackrel{\text{def}}{=} \sum_{j=1}^{i-1} b_j$ rows have already been eliminated. Therefore, the algorithm performs a partial Gaussian elimination on a new block of b_i rows of the submatrix $\mathbf{G}'_{[\bar{I}:]} \in \mathbb{F}_q^{(n-\mu)\times k}$. Summing this cost over all nodes and all levels accounts for the term $\sum_{i=1}^{N}\left(\prod_{j=1}^{i} r_j\right) T_{\text{Gauss}}(i)$.

At level $N+1$, the leaves perform no further Gaussian elimination. Instead, each leaf executes the meet-in-the-middle procedure described between Lines 11 and 25. Since the number of leaves is $\prod_{i=1}^{N+1} r_i$, it remains to determine the cost incurred at each leaf. This cost is identical to that of the meet-in-the-middle step in Stern's algorithm. Hence, the per-leaf cost is the same term $T_{\text{list}} + k\left(T_{\boldsymbol{y}'_Z} + T_{\text{check}}\right)$ as in Proposition 3.1.

Finally, the memory complexity follows from the storage required for the matrix, the k noisy codewords, and the lookup table $\mathcal{T}$. There is no need to store a separate copy of $\mathbf{G}'$ at each recursion level. Indeed, to compute the matrix associated with recursion level i, one may start from the matrix obtained at any recursion level $j \geq i-1$, rather than necessarily from the matrix at level $i-1$. This is because the matrix at level j is obtained from that at level $i-1$ through column operations, which can be reversed or continued as needed. Consequently, only the current matrix $\mathbf{G}'$ needs to be stored; it can be viewed as a global variable updated at each partial Gaussian-elimination step. □

5.3 Success Probability of the MAGE-Stern Algorithm

As with Algorithm 3.1, the overall success probability of Algorithm 5.1 can be expressed as the survival probability of a branching process. However, since the distribution of the surviving branches depends on the weight of the error restricted to the blocks processed so far, the algorithm must be modeled as a multitype branching process in a varying environment.

Definition 5.3. *Consider the inputs, parameters, and notation of Algorithm 5.1. For all $j \in [\![1,k]\!]$, let $\boldsymbol{e}_j$ denote the unique error vector satisfying $\boldsymbol{y}_j - \boldsymbol{e}_j \in \{\mathbf{G}\boldsymbol{u} : \boldsymbol{u} \in \mathbb{F}_q^k\}$ and $|\boldsymbol{e}_j| = t$. Let $\mathcal{P} \stackrel{\text{def}}{=} [\![0, 2p-\delta]\!] \cup \{\perp\}$ denote the set of types, where $\perp$ represents all weights strictly greater than $2p-\delta$[7]. For all $i \in [\![1,N]\!]$, we define the random vector $\boldsymbol{Z}_i \stackrel{\text{def}}{=} \left(Z_i(\tau)\right)_{\tau\in\mathcal{P}}$ where $Z_i(\tau)$ counts the number of error vectors whose restriction to the blocks processed up to depth i has Hamming weight τ. More precisely, for all $\tau \in \mathcal{P}$,*

$$Z_i(\tau) \stackrel{\text{def}}{=} |\{j \in [\![1,k]\!] \ : \ |(\boldsymbol{e}_j)_{J_i}| = \tau\}| \tag{5.3}$$

Here, $J_i \stackrel{\text{def}}{=} \bigcup_{j=1}^{i} I_i$ is the random set of coordinates selected during the first i levels of the recursion. Its cardinality is

$$\mu_i \stackrel{\text{def}}{=} |J_i| = \sum_{j=1}^{i} b_j. \tag{5.4}$$

By convention, we set $J_0 \stackrel{\text{def}}{=} \emptyset$ and $\mu_0 \stackrel{\text{def}}{=} 0$.

Additionally, we define the random vector $\boldsymbol{Z}_{N+1} \stackrel{\text{def}}{=} \left(Z_{N+1}(\tau)\right)_{\tau\in\mathcal{P}}$ with

$$Z_{N+1}(2p-\delta) \stackrel{\text{def}}{=} |\{j \in [\![1,k]\!] \ : \ |(\boldsymbol{e}_j)_{I_L}| = p, \ |(\boldsymbol{e}_j)_{I_R}| = p-\delta, \ (\boldsymbol{e}_j)_Z = \mathbf{0}\}| \tag{5.5}$$

$$Z_{N+1}(\perp) \stackrel{\text{def}}{=} k - Z_{N+1}(2p-\delta), \tag{5.6}$$

$$Z_{N+1}(\tau) \stackrel{\text{def}}{=} 0, \qquad \tau \in \mathcal{P} \setminus \{2p-\delta, \perp\} \tag{5.7}$$

where (I_L, I_R) is a random bipartition of $I = J_N$ and Z is a random set of size ℓ included in $\overline{I}$.

Remark 5.4. At the final generation $N+1$, the types have a different interpretation from that used at the preceding generations. By convention, individuals of type $2p-\delta$ are precisely the target individuals, whereas all remaining individuals are assigned type $\perp$.

Proposition 5.5. *Let $\mathcal{P}$ and $\left(\boldsymbol{Z}_i\right)_{i\in[\![1,N+1]\!]}$ be defined as in Definition 5.3. Then $\left(\boldsymbol{Z}_i\right)_{i\in[\![1,N+1]\!]}$ is a multitype branching process in a varying environment with initial population $\boldsymbol{Z}_0 = (k, 0, \cdots, 0)$ and offspring distribution*

$$\boldsymbol{X}^{(i,\tau)} \stackrel{\text{def}}{=} \left(X^{(i,\tau)}(\sigma)\right)_{\sigma\in\mathcal{P}} \sim \text{Multinomial}(r_i; (P_i(\tau,\sigma))_{\sigma\in\mathcal{P}}) \tag{5.8}$$

where

$$P_i(\tau,\sigma) \stackrel{\text{def}}{=} \mathbb{P}\left(|\boldsymbol{e}_{J_i}| = \sigma \ \middle| \ \left|\boldsymbol{e}_{J_{i-1}}\right| = \tau\right), \qquad 1 \le i \le N \tag{5.9}$$

[7] By a slight abuse of notation, for any integer x, we write $x = \perp$ to mean that $x > 2p-\delta$. Moreover, we extend the usual order on $[\![0, 2p-\delta]\!]$ by declaring that $x < \perp$ is true for all $x \in [\![0, 2p-\delta]\!]$ and that $\perp < \perp$ is false.

are given by

$$P_i(\tau,\sigma) = \begin{cases} 1 & \text{if } \tau = \sigma = \perp \\ 0 & \text{if } \sigma < \tau \\ \dfrac{\binom{b_i}{\sigma-\tau}\binom{n-\mu_i}{t-\sigma}}{\binom{n-\mu_{i-1}}{t-\tau}} & \text{if } \tau \le \sigma < \perp \\ 1 - \displaystyle\sum_{j=\tau}^{2p-\delta} \dfrac{\binom{b_i}{j-\tau}\binom{n-\mu_i}{t-j}}{\binom{n-\mu_{i-1}}{t-\tau}} & \text{if } \tau < \sigma = \perp \end{cases} \tag{5.10}$$

and

$$P_{N+1}(\tau,\sigma) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \tau \neq 2p-\delta \text{ and } \sigma = \perp \\ 0 & \text{if } \tau \neq 2p-\delta \text{ and } \sigma \neq \perp \\ 0 & \text{if } \tau = 2p-\delta \text{ and } \sigma \notin \{2p-\delta, \perp\} \\ \dfrac{\binom{k/2}{p}\binom{k/2}{p-\delta}}{\binom{k}{2p-\delta}} \cdot \dfrac{\binom{n-k-2p+\delta}{\ell}}{\binom{n-k}{\ell}} & \text{if } \tau = \sigma = 2p-\delta \\ 1 - P_{N+1}(2p-\delta, 2p-\delta) & \text{if } \tau = 2p-\delta \text{ and } \sigma = \perp. \end{cases} \tag{5.11}$$

Proof. We identify an individual at generation $i \in [\![1, N]\!]$ with a pair consisting of a node of the recursion tree at depth i and one of the k error vectors. Each node at level i is associated with a set J_i of cardinality μ_i, obtained from the set J_{i-1} associated with its parent by adjoining a uniformly random block $I_i \stackrel{\text{def}}{=} J_i \setminus J_{i-1}$ of cardinality b_i, chosen among the coordinates not previously selected. The type of an individual is then the Hamming weight of the corresponding error vector restricted to the set J_i, with all weights greater than $2p-\delta$ identified with the type $\perp$.

At generation 0, the recursion tree consists of a single root and $|(\boldsymbol{e}_j)_{J_0}| = 0$ for every $j \in [\![1, k]\!]$. Consequently, $\boldsymbol{Z}_0 = (k, 0, \ldots, 0)$.

Fix $i \in [\![1, N]\!]$ and consider an individual of type $\tau \in \mathcal{P}$ at generation $i-1$. The corresponding node gives rise to r_i children at generation i. We first determine the type distribution of one such child.

Suppose first that $\tau = \perp$. Since $J_{i-1} \subseteq J_i$, the Hamming weight of the restriction cannot decrease. Hence every child is also of type $\perp$, and therefore

$$P_i(\perp, \perp) = 1, \qquad P_i(\perp, \sigma) = 0 \quad \text{for every } \sigma < \perp.$$

Now assume that $\tau < \perp$. Conditional on $\left|(\boldsymbol{e}_j)_{J_{i-1}}\right| = \tau$, there are exactly $t-\tau$ error positions among the $n-\mu_{i-1}$ coordinates outside J_{i-1}. The new block $I_i = J_i \setminus J_{i-1}$ has cardinality b_i. Thus, for $\tau \le \sigma < \perp$, the event $|(\boldsymbol{e}_j)_{J_i}| = \sigma$ occurs precisely when I_i contains $\sigma - \tau$ of the remaining $t-\tau$ error positions. Consequently,

$$P_i(\tau,\sigma) = \frac{\binom{b_i}{\sigma-\tau}\binom{n-\mu_i}{t-\sigma}}{\binom{n-\mu_{i-1}}{t-\tau}}.$$

Moreover, the weight cannot decrease, so $P_i(\tau, \sigma) = 0$ whenever $\sigma < \tau$.

Finally, a child has type $\perp$ exactly when the weight of the corresponding error vector on J_i is greater than $2p - \delta$. Hence

$$P_i(\tau, \perp) = 1 - \sum_{j=\tau}^{2p-\delta} \frac{\binom{b_i}{j-\tau}\binom{n-\mu_i}{t-j}}{\binom{n-\mu_{i-1}}{t-\tau}}.$$

The final generation $N+1$ is treated differently. Any individual whose type at generation N is different from $2p - \delta$ is assigned type $\perp$ at generation $N+1$, since such an individual cannot be accepted. Now consider an individual of type $2p - \delta$ at generation N. This means that the corresponding error vector has weight $2p - \delta$ on $I = J_N$. The individual is then accepted, and hence assigned type $2p - \delta$ at generation $N+1$, if the support of this error vector is suitably distributed with respect to the random bipartition $I = I_L \cup I_R$ and the random subset $Z \subseteq \overline{I}$ of cardinality ℓ. Otherwise, it is rejected and assigned type $\perp$. This yields the transition probabilities P_{N+1}.

Finally, let $T_1, \cdots, T_{r_i}$ denote the types of the r_i children of the considered individual. By construction, conditional on the parent having type τ, the random blocks used to generate these children are independent and identically distributed. Therefore,

$$\mathbb{P}\left(T_j = \sigma \mid T_{\text{parent}} = \tau\right) = P_i(\tau, \sigma)$$

for all $j \in [\![1, r_i]\!]$ and all $\sigma \in \mathcal{P}$. Therefore, the offspring vector is

$$X^{(i,\tau)}(\sigma) \stackrel{\text{def}}{=} \sum_{j=1}^{r_i} \mathbb{1}_{T_j = \sigma},$$

hence

$$\boldsymbol{X}^{(i,\tau)} \sim \text{Multinomial}\left(r_i; \left(P_i(\tau, \sigma)\right)_{\sigma \in \mathcal{P}}\right).$$

Since the offspring vectors associated with distinct individuals are independent, and since their distributions depend only on their types and on the generation i, the process $\left(\boldsymbol{Z}_i\right)_{i \in [\![0,N]\!]}$ is a multitype branching process in a varying environment. □

We are now ready to state our main theorem, which characterizes the success probability of Algorithm 5.1. In particular, Theorem 5.6 provides an explicit expression that can be evaluated numerically for any fixed set of parameters.

Theorem 5.6. *Algorithm 5.1 succeeds, that is, returns an output different from $\perp$, with probability*

$$\mathbb{P}_{\text{success}}\left(\textit{MAGE-Stern}\right) = 1 - q_0(0)^k. \tag{5.12}$$

The quantities $q_i(\tau)$ are defined recursively as follows:

$$q_N(\tau) = 1, \qquad \text{for all } \tau \neq 2p-\delta \tag{5.13}$$

$$q_N(2p-\delta) = \left(1 - \frac{\binom{k/2}{p}\binom{k/2}{p-\delta}}{\binom{k}{2p-\delta}} \cdot \frac{\binom{n-k-t+2p-\delta}{\ell}}{\binom{n-k}{\ell}}\right)^{r_{N+1}} \tag{5.14}$$

$$q_{i-1}(\tau) = \left(1 - \sum_{\sigma=\tau}^{2p-\delta} \frac{\binom{b_i}{\sigma-\tau}\binom{n-\mu_i}{t-\sigma}}{\binom{n-\mu_{i-1}}{t-\tau}}\big(1-q_i(\sigma)\big)\right)^{r_i}, \quad 1 \le i \le N \tag{5.15}$$

Proof. Algorithm 5.1 succeeds if and only if at least one individual of type $2p-\delta$ is present at generation $N+1$. Therefore,

$$\mathbb{P}_{\text{success}}\,(\text{MAGE-Stern}) = \mathbb{P}\left(Z_{N+1}(2p-\delta) > 0 \;\middle|\; \boldsymbol{Z}_0 = (k, 0, \cdots, 0)\right).$$

For all $i \in [\![0, N+1]\!]$ and all $\tau \in \mathcal{P}$, let $q_i(\tau)$ denote the probability that no individual of type $2p-\delta$ is present at generation $N+1$, conditional on the process starting at generation i with a single individual of type τ. At the terminal generation, we thus have

$$q_{N+1}(\tau) = 1 - \mathbb{1}_{\tau=2p-\delta}$$

By Proposition 5.5 and Proposition 2.11, the failure probability starting from a single individual of type 0 at generation 0 is $q_0(0)$. Since the process starts with k independent individuals of type 0, the branching property gives

$$\mathbb{P}\left(Z_{N+1}(2p-\delta) = 0 \;\middle|\; \boldsymbol{Z}_0 = (k, 0, \ldots, 0)\right) = q_0(0)^k.$$

Taking the complementary event yields Equation (7.3).

It remains to derive the recursion defining the q_i's. For an individual of type τ at generation $i-1$, by Proposition 2.11 and Equation (2.12),

$$q_{i-1}(\tau) = F_i^{(\tau)}\big((s_\sigma)_{\sigma\in\mathcal{P}}\big) = \left(\sum_{\sigma\in\mathcal{P}} P_i(\tau,\sigma)s_\sigma\right)^{r_i}. \tag{5.16}$$

We first consider the transition from generation N to generation $N+1$. If $\tau \neq 2p-\delta$, the individual is necessarily assigned type $\perp$ at generation $N+1$. Hence $q_N(\tau) = 1$. Now suppose that $\tau = 2p-\delta$. Let

$$\rho \stackrel{\text{def}}{=} P_{N+1}(2p-\delta, 2p-\delta) = \frac{\binom{k/2}{p}\binom{k/2}{p-\delta}}{\binom{k}{2p-\delta}} \cdot \frac{\binom{n-k-t+2p-\delta}{\ell}}{\binom{n-k}{\ell}}$$

denote the probability that one child is accepted at the final generation. Such a child has type $2p-\delta$ with probability ρ, and type $\perp$ otherwise. Since

$$q_{N+1}(2p-\delta) = 0 \qquad \text{and} \qquad q_{N+1}(\perp) = 1,$$

Equation (5.16) gives

$$q_N(2p-\delta) = \big(\rho q_{N+1}(2p-\delta) + (1-\rho)q_{N+1}(\perp)\big)^{r_{N+1}} = (1-\rho)^{r_{N+1}},$$

which proves Equation (5.14).

We now consider a generation $i \in [\![1, N]\!]$. An individual of type $\perp$ remains of type $\perp$ at every subsequent generation and can therefore never produce an accepted individual. Thus,

$$q_i(\perp) = 1, \qquad \text{for all } i \in [\![0, N]\!].$$

For $\tau \in [\![0, 2p - \delta]\!]$, Equation (5.16) therefore becomes

$$\begin{aligned} q_{i-1}(\tau) &= \left(P_i(\tau, \perp) + \textstyle\sum_{\sigma=0}^{2p-\delta} P_i(\tau, \sigma) q_i(\sigma)\right)^{r_i} \\ &= \left(1 - \textstyle\sum_{\sigma=0}^{2p-\delta} P_i(\tau, \sigma) + \sum_{\sigma=0}^{2p-\delta} P_i(\tau, \sigma) q_i(\sigma)\right)^{r_i} \\ &= \left(1 - \textstyle\sum_{\sigma=0}^{2p-\delta} P_i(\tau, \sigma)\big(1 - q_i(\sigma)\big)\right)^{r_i}. \end{aligned}$$

Since $P_i(\tau, \sigma) = 0$ whenever $\sigma < \tau$, this simplifies to

$$q_{i-1}(\tau) = \left(1 - \sum_{\sigma=\tau}^{2p-\delta} P_i(\tau, \sigma)\big(1 - q_i(\sigma)\big)\right)^{r_i}.$$

Substituting the expression of $P_i(\tau, \sigma)$ given in Proposition 5.5 yields Equation (5.15). □

6 Application to HQC

The security of HQC relies on the hardness of solving instances of the 2-DOOM and 3-DOOM problems, whose parameters are reported in Tables 6.1 and 6.2, respectively. HQC provides parameter sets targeting NIST security Categories I, III, and V, corresponding to security levels comparable to those of AES-128, AES-192, and AES-256, respectively.

2-DOOM	k	$n = 2k$	t
HQC-1 (Cat. I)	17669	35338	132
HQC-3 (Cat. III)	35851	71702	200
HQC-5 (Cat. V)	57637	115274	262

Table 6.1: Parameters of the 2-DOOM instances underlying the security of HQC.

Our results are summarized in Tables 6.3 and 6.4, corresponding to the 2-DOOM and 3-DOOM problems underlying the security of HQC, respectively. In these tables, the columns denoted by T report the base-2 logarithm of the time complexity, while the columns denoted by M report the base-2 logarithm

3-DOOM	k	$n = 3k$	t
HQC-1 (Cat. I)	17669	53007	225
HQC-3 (Cat. III)	35851	107553	342
HQC-5 (Cat. V)	57637	172911	447

Table 6.2: Parameters of the 3-DOOM instances underlying the security of HQC.

of the memory complexity. All complexities are evaluated in the logic-gate cost Model 2.1.

According to [22], Stern's algorithm is the most efficient known attack against the HQC parameter sets considered here, and therefore serves as our main baseline. Accordingly, the columns labeled *Stern* in our tables refer to the classical Stern algorithm, including the improvements described in Section 3.1. Its complexity is established in Proposition 3.1.

The columns labeled *Progressive Sieving-ISD* correspond to the attack introduced in [45], where it is presented as the state-of-the-art ISD attack against HQC. In particular, the authors report an improvement of approximately 8 bits over Stern's algorithm and other ISD techniques for the HQC Category I parameter set. However, this comparison relies on an overestimated complexity for Stern's algorithm. This overestimation stems from an incorrect treatment of the DOOM setting, together with an error in the implementation of the complexity computation that introduces an extraneous multiplicative factor. Moreover, the two algorithms are evaluated under inconsistent random-access cost assumptions: random-access queries are assigned unit cost for Progressive Sieving-ISD, whereas they are charged $\log_2(M)$ cost for Stern's algorithm, where M denotes the memory size. After correcting these discrepancies and evaluating both algorithms within the same computational Model 2.1 (see Appendix A), our results show that the actual improvement achieved by Progressive Sieving-ISD is substantially smaller than that originally reported in [45].

The columns labeled *MAGE-Stern* report the complexity of our approach. Its expected running time is obtained by dividing the cost of one execution of Algorithm 5.1, established in Proposition 5.2, by the success probability derived in Theorem 5.6. This column places our approach as the new state-of-the-art for attacking HQC, in all the parameter sets. They also indicate that the security of HQC falls short of the NIST security requirements by up to 3 bits for Category I and V. Another noteworthy observation is the significant reduction in memory complexity achieved by MAGE-Stern compared to Stern's algorithm, reaching up to 13 bits for Category V.

We also evaluate an idealized lower bound on the complexity of MAGE-Stern. This bound is obtained by considering Stern's algorithm with a quadratic-cost Gaussian elimination step, without crediting the algorithm with any increase in success probability. More precisely, it is given by the complexity in Proposition 3.1, with T_{Gauss} replaced by $\mathsf{GAUSS}_q(k, n, 1)$. It therefore represents the

largest improvement that could theoretically be obtained through our optimization of the Gaussian-elimination step alone. Comparing MAGE-Stern with this lower bound shows that the reduction in Gaussian elimination cost achieved by our algorithm is nearly optimal.

2-DOOM	**Required security level**	**Stern**		**Progressive Sieving-ISD**		**MAGE-Stern**		**Stern with quadratic GE** (lower bound)	
		T	M	T	M	T	M	T	M
HQC-1 (Cat. I)	143	142.84	42.13	142.52	42.46	**139.87**	**31.12**	139.44	31.12
HQC-3 (Cat. III)	207	210.00	45.29	210.17	45.60	**207.28**	**33.21**	206.80	33.21
HQC-5 (Cat. V)	272	271.50	47.35	272.27	48.35	**268.91**	**34.58**	268.39	34.58

Table 6.3: Time and memory complexities of attacks against the HQC 2-DOOM instances ($\log_2$ scale).

3-DOOM	**Required security level**	**Stern**		**Progressive Sieving-ISD**		**MAGE-Stern**		**Stern with quadratic GE** (lower bound)	
		T	M	T	M	T	M	T	M
HQC-1 (Cat. I)	143	143.49	42.13	143.21	42.46	**140.44**	**31.47**	139.67	31.47
HQC-3 (Cat. III)	207	211.10	45.29	211.29	45.58	**208.31**	**33.54**	207.48	33.54
HQC-5 (Cat. V)	272	272.04	47.35	272.83	48.34	**269.39**	**34.91**	268.50	34.91

Table 6.4: Time and memory complexities of attacks against the HQC 3-DOOM instances ($\log_2$ scale).

Numerical Optimization of MAGE-Stern. To obtain the results reported in our tables, we optimized the parameters of MAGE-Stern by performing an exhaustive search over δ, ℓ, p, and N. For each candidate triple, the block sizes

$(b_i)_{i\in[\![1,N]\!]}$ and branching factors $(r_i)_{i\in[\![1,N+1]\!]}$ were optimized using a local-descent procedure.[8] Although this procedure yields good parameter sets, it does not guarantee global optimality, since the descent may converge to a local minimum. To mitigate this issue, we ran the optimization from several distinct initial parameter sets. Despite this, the reported complexities should therefore be interpreted as achievable upper bounds rather than as provably optimal values.

Three variables can be removed from the optimization by setting δ, r_1 and r_{N+1} to 1. First, as discussed in the paragraph "Adapting to DOOM" in Section 3.1, the optimal choice for δ in the binary case is 1, as this ensures the balance condition in Equation (3.7).

Next, to see why r_1 can be fixed to 1, recall that one execution of Algorithm 5.1 succeeds with probability $\mathbb{P}_{\text{success}}$ (MAGE-Stern). Hence, the algorithm must be executed an expected $\frac{1}{\mathbb{P}_{\text{success}}(\text{MAGE-Stern})}$ times before it succeeds. Consequently, the average total number of trials performed at the first recursion level is $\frac{r_1}{\mathbb{P}_{\text{success}}(\text{MAGE-Stern})}$, rather than merely r_1. Increasing r_1 can therefore be absorbed into the number of independent executions of the algorithm. We may thus set $r_1 = 1$ and account for this choice through the success probability given in Theorem 5.6.

Finally, r_{N+1} can also be set to 1, or at least to a value very close to 1. Indeed, in the sublinear regime considered here, randomizing the collision window and the bipartition of the information set provides almost no additional benefit: the corresponding conditional success probability P_{N+1} is already very close to 1. Repeating this final step is therefore unnecessary.

The code used to compute the various complexities, together with the parameter sets yielding the results reported in Tables 6.3 and 6.4, is available at https://github.com/ValerianHatey/HQC_Calibration.

For instance, optimizing MAGE-Stern for the Category I parameters of 2-DOOM_2 yields $p = 2$, $\ell = 43$ and $N = 10$, with block sizes

$$(b_i)_{i\in[\![1,N]\!]} = [13978,\ 1247,\ 568,\ 399,\ 385,\ 298,\ 218,\ 203,\ 188,\ 184], \tag{6.1}$$

and branching factors

$$(r_i)_{i\in[\![1,N+1]\!]} = [1,\ 94,\ 7,\ 3,\ 3,\ 3,\ 2,\ 2,\ 2,\ 2,\ 1]. \tag{6.2}$$

By Theorem 5.6, the corresponding success probability is

$$\mathbb{P}_{\text{success}}\,(\text{MAGE-Stern}) = 2^{-83.57}. \tag{6.3}$$

Thus, the expected number of independent executions, and hence of first-level trials, is $2^{83.57}$. Hence, the optimized parameters indicate that extensive recursive factorization of the Gaussian elimination step is not beneficial in this instance. Most coordinates are processed in the first block, whereas the subsequent factorization is restricted to considerably smaller blocks. Beyond a certain recursion

[8] We adopted a similar strategy for Progressive Sieving-ISD, combining local descent with an exhaustive search over a neighborhood of the parameter values expected to be optimal.

depth, Gaussian elimination becomes cheaper than the meet-in-the-middle step, so further factorization would reduce the former while increasing the overall branching overhead without improving the total complexity.

Quantifying the Gains of MAGE-Stern over Stern. Table 6.5 compares the optimized parameters and the dominant per-information-set costs of Stern and MAGE-Stern for the HQC Category I parameter set. The last column reports the effective number of information sets explored before a solution is found. For Stern's algorithm, this quantity is simply the inverse of the success probability of one iteration. For MAGE-Stern, it is given by

$$\frac{\prod_{i=1}^{N+1} r_i}{\mathbb{P}_{\text{success}}\,(\text{MAGE-Stern})}, \tag{6.4}$$

where the success probability in the denominator is established in Theorem 5.6.

To compare the Gaussian-elimination costs on a per-information-set basis, the value reported for MAGE-Stern is the average Gaussian-elimination cost over all leaves of the branching process. More precisely,

$$T_{\text{Gauss}} = \frac{\sum_{i=1}^{N} \left(\prod_{j=1}^{i} r_j\right) T_{\text{Gauss}}(i)}{\prod_{i=1}^{N+1} r_i} = \sum_{i=1}^{N} \frac{T_{\text{Gauss}}(i)}{\prod_{j=i+1}^{N+1} r_j}. \tag{6.5}$$

Algorithm	**Parameters**			**Cost per information set**			**Number of information sets**
	p	ℓ	δ	T_{Gauss}	T_{list}	kT_{check}	
Stern	3	55	1	41.91	48.57	50.18	92.24
MAGE-Stern	2	43	1	35.84	36.04	37.15	101.69

Table 6.5: Comparison of the per-information-set costs of Stern and MAGE-Stern for the HQC Category I parameter set. All costs are reported on a $\log_2$ scale.

Table 6.5 explains the origin of both the time and memory improvements achieved by MAGE-Stern. For the parameters considered here, the Gaussian-elimination cost of Stern's algorithm amounts to $2^{41.91}$ operations. MAGE-Stern reduces this cost by approximately 6 bits. When p and ℓ are kept unchanged, this reduction has no effect on the overall complexity, since T_{Gauss} is not the dominant term. Its main benefit is instead to change the optimal parameter regime: $p = 2$ becomes optimal for MAGE-Stern, whereas Stern's algorithm is optimized with $p = 3$. For the resulting MAGE-Stern parameters, the Gaussian-elimination cost remains slightly below both the list-construction cost T_{list} and the verification cost kT_{check}. Overall, the cost per information set decreases by approximately

13 bits. This reduction comes at the expense of a lower success probability. As shown in the last column of Table 6.5, MAGE-Stern must explore approximately $2^{9.45}$ times as many information sets as Stern before finding a solution. After accounting for this additional branching overhead, the net improvement in time complexity is approximately 3.55 bits.

The memory improvement follows from the same phenomenon. In both algorithms, the memory complexity is given by

$$M = p\binom{k/2}{p}\log_2\left(\frac{k}{2}\right) + 2kn. \tag{6.6}$$

Since the optimal parameter changes from $p = 3$ for Stern's algorithm to $p = 2$ for MAGE-Stern, the memory complexity decreases from cubic to quadratic in k. Consequently, MAGE-Stern achieves an asymptotic reduction in memory complexity by a factor of order k.

6.1 Exploiting the Regular Structure

None of the three algorithms considered previously exploits the regular structure of the 2-DOOM and 3-DOOM problems. The reduction of [26] implies that any algorithm solving the regular problem can provide at most a multiplicative gain of

$$\frac{\binom{n}{t}}{\binom{n/d}{t/d}^d} \tag{6.7}$$

over the corresponding unstructured problem. That would give us the complexities given in Table 6.6 for HQC.

	2-DOOM		3-DOOM	
	$\log_2\left(\frac{\binom{n}{t}}{\binom{n/d}{t/d}^d}\right)$	**Regular MAGE-Stern** (lower bound)	$\log_2\left(\frac{\binom{n}{t}}{\binom{n/d}{t/d}^d}\right)$	**Regular MAGE-Stern** (lower bound)
HQC-1 (Cat. I)	3.85	136.02	8.09	132.35
HQC-3 (Cat. III)	4.15	203.13	8.69	199.62
HQC-5 (Cat. V)	4.35	264.56	9.08	260.21

Table 6.6: Potential acceleration for attacking HQC by exploiting the regular structure of 2-DOOM and 3-DOOM ($\log_2$ scale).

As shown in Table 6.6, for the HQC Category I parameters, achieving this upper bound would yield, for instance, a 4-bit speedup over the non-regular MAGE-Stern attack against 2-DOOM, which would be significant.

However, this upper bound appears to be overly optimistic. For the HQC parameter sets, we were unable to obtain any noticeable improvement by exploiting the regular structure within MAGE-Stern. To illustrate this phenomenon, consider an instance of 2-$\mathsf{Dec}_2(n,k,t)$ and the classical Stern algorithm. For simplicity, we omit the DOOM setting and consider Stern rather than MAGE-Stern.

Let $\boldsymbol{e} = (\boldsymbol{e}^{(1)}, \boldsymbol{e}^{(2)}) \in \mathbb{F}_2^{n/2} \times \mathbb{F}_2^{n/2}$ denote the target error vector, with $\left|\boldsymbol{e}^{(1)}\right| = \left|\boldsymbol{e}^{(2)}\right| = \frac{t}{2}$.

A natural approach is to adapt Stern's guessing strategy to the regular structure. For each $i \in \{1,2\}$, let

$$I^{(i)} \subseteq \left[\!\left[1, \frac{n}{2}\right]\!\right], \qquad \overline{I^{(i)}} \stackrel{\text{def}}{=} \left[\!\left[1, \frac{n}{2}\right]\!\right] \setminus I^{(i)} \qquad \text{and} \qquad Z^{(i)} \subseteq \overline{I^{(i)}} \tag{6.8}$$

where

$$\left|I^{(i)}\right| = \frac{k}{2} \qquad \text{and} \qquad \left|Z^{(i)}\right| = \frac{\ell}{2}. \tag{6.9}$$

Instead of using the usual Stern guess, we require, for each $i \in \{1,2\}$,

$$\left|\left(\boldsymbol{e}^{(i)}\right)_{I^{(i)}}\right| = p, \qquad \left|\left(\boldsymbol{e}^{(i)}\right)_{\overline{I^{(i)}}}\right| = \frac{t-2p}{2} \qquad \text{and} \qquad \left|\left(\boldsymbol{e}^{(i)}\right)_{Z^{(i)}}\right| = 0. \tag{6.10}$$

The success probability of this regularity-aware guess is therefore

$$\mathbb{P}_{\text{success}}(\text{Reg-Stern}) = \frac{\binom{\frac{k}{2}}{p}^2 \binom{\frac{n-k-\ell}{2}}{\frac{t-2p}{2}}^2}{\binom{\frac{n}{2}}{\frac{t}{2}}^2}. \tag{6.11}$$

This modified guess does not change the cost of one iteration of Stern's algorithm; it only affects its success probability. Hence,

$$\frac{T_{\text{Stern}}}{T_{\text{Reg-Stern}}} = \frac{\mathbb{P}_{\text{success}}(\text{Reg-Stern})}{\mathbb{P}_{\text{success}}(\text{Stern})} \tag{6.12}$$

$$= \frac{\binom{\frac{n-k-\ell}{2}}{\frac{t-2p}{2}}^2 \binom{n}{t}}{\binom{\frac{n}{2}}{\frac{t}{2}}^2 \binom{n-k-\ell}{t-2p}} \tag{6.13}$$

$$\approx \frac{\sqrt{2\pi\frac{t}{2}}^2 \sqrt{2\pi(t-2p)}}{\sqrt{2\pi\frac{t-2p}{2}}^2 \sqrt{2\pi t}} = \sqrt{\frac{t}{t-2p}}. \tag{6.14}$$

where the approximation in (6.14) follows from Stirling's formula.

For the HQC parameters, the optimal value is $p = 3$, and $p \ll t$. Consequently, $\frac{T_{\text{Stern}}}{T_{\text{Reg-Stern}}}$ is very close to 1. Adapting Stern's guessing strategy to the regular setting therefore provides only a negligible improvement for the HQC parameter sets. The same reasoning applies to Progressive Sieving-ISD and MAGE-Stern, whose guessing strategies have the same underlying combinatorial structure. We were unable to identify an approach that achieves a significant fraction of the theoretical upper bound established in [26].

7 Projective MAGE-Stern

The security of many Pseudorandom Correlation Generator (PCG) protocols relies on the hardness of either the classical decoding problem $\mathsf{Dec}_q(n,k,t)$ or the regular decoding problem $d\text{-}\mathsf{Dec}_q(n,k,t)$ over large finite fields. In this section, we introduce a new algorithm, called *Projective MAGE-Stern*, tailored to such parameter regimes. Its main idea is to combine the Gaussian-elimination factorization technique of [30] with the projective-space framework of [18]. As explained at the end of Section 3.1, the latter reduces the enumeration cost of the meet-in-the-middle step by a factor of approximately q by carrying out the search in projective space. For very large q, this technique makes Stern's algorithm competitive, whereas the non-projective approach is effectively limited to Prange's algorithm.

Let $(\mathbf{G}, \boldsymbol{y})$ be an instance of the decoding problem $\mathsf{Dec}_q(n,k,t)$. The projective-space framework can only be applied when the target noisy word $\boldsymbol{y}$ is itself a codeword, which we may take to be $\boldsymbol{y} = \mathbf{0}$ since the code is linear. We therefore first apply Theorem 3.2 to reduce the original problem $\mathsf{Dec}_q(n,k,t)$ to an instance of $\mathsf{Dec}_q(n,k+1,t)$ with target vector $\mathbf{0}$.

Algorithm 7.1 describes our Projective MAGE-Stern algorithm for solving $\mathsf{Dec}_q(n,k,t)$. Lines 1–7 implement the reduction of Theorem 3.2. The procedure *Proj-MAGE-Stern-Procedure* then solves the resulting instance of $\mathsf{Dec}_q(n,k,t)$ with target vector $\boldsymbol{y} = \mathbf{0}$[9]. The structure of the algorithm closely follows that of Algorithm 5.1, but is adapted to the setting in which the unique target vector is $\mathbf{0}$; in particular, the problem is no longer a DOOM instance. Moreover, the algorithm operates on representatives of the equivalence classes induced by nonzero scalar multiplication. To ensure that equivalent vectors are mapped to the same representative, and hence collide whenever expected, we repeatedly use the function $\mathsf{fnze}(\cdot)$, which returns the first nonzero entry of a nonzero vector:

Notation 7.1. *For any $n \in \mathbb{N}^*$ and any nonzero vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, we define*

$$\mathsf{fnze}(\boldsymbol{x}) \stackrel{\text{def}}{=} x_{\min \operatorname{supp}(\boldsymbol{x})}, \tag{7.1}$$

where

$$\operatorname{supp}(\boldsymbol{x}) \stackrel{\text{def}}{=} \{i \in [\![1,n]\!] \ : \ x_i \neq 0\} \tag{7.2}$$

denotes the support of $\boldsymbol{x}$.

Algorithm 7.1 deals with the classical decoding problem and no longer with DOOM. So there is only one word to decode here and we essentially choose $\delta = 0$ to ensure that our lists are balanced (that is why δ doesn't appear in Algorithm 7.1 contrary to Algorithm 5.1). But apart from that, the branching process is exactly the same. And so, we have the following success probability for our Projective MAGE-Stern algorithm:

[9] We do not provide further details on the rationale underlying the algorithm, since its correctness follows from the same principles as Projective Stern in [18]. We refer the reader to [18] for a detailed explanation of the projective-space framework and of the correctness of this approach.

Algorithm 7.1: Projective MAGE-Stern

Input: An instance $(\mathbf{G}, \boldsymbol{y})$ of $\mathsf{Dec}_q(n,k,t)$.

Parameters: a number $N \in \mathbb{N}^*$ of blocks, block sizes $(b_1, \cdots, b_N)$ such that $\sum_{i=1}^{N} b_i = k$, repetition numbers $(r_1, \ldots, r_N, r_{N+1})$, an integer $\ell \in [\![1, n-k-1]\!]$, and an integer p.

Output: Either $\boldsymbol{e} \in \mathbb{F}_q^n$ such that $|\boldsymbol{e}| = t$ and $(\boldsymbol{y} - \boldsymbol{e}) \in \{\mathbf{G}\boldsymbol{u} : \boldsymbol{u} \in \mathbb{F}_q^k\}$, or $\perp$.

compute $\boldsymbol{h} \in \mathbb{F}_q^n$ such that $\mathbf{G}^\top \boldsymbol{h} = \mathbf{0}$ and $\langle \boldsymbol{h}, \boldsymbol{y} \rangle \neq 0$;
$\mathbf{G}' \leftarrow [\mathbf{G}|\boldsymbol{y}]$;
$\boldsymbol{c} \leftarrow$ PROJ-MAGE-STERN-PROCEDURE$(1, \mathbf{G}', \emptyset)$;
if $\boldsymbol{c} = \perp$ *or* $\langle \boldsymbol{h}, \boldsymbol{c} \rangle = 0$ **then**
  return $\perp$
else
  return $\frac{\langle \boldsymbol{h}, \boldsymbol{y} \rangle}{\langle \boldsymbol{h}, \boldsymbol{c} \rangle} \boldsymbol{c}$

Function *PROJ-MAGE-STERN-PROCEDURE*$(i, \mathbf{G}', I)$**:**
  $\mu \leftarrow \sum_{j=1}^{i-1} b_j$;
  $(J, \overline{J}) \leftarrow ([\![1, \mu]\!], [\![\mu + 1, k+1]\!])$;
  repeat r_i **times**
    choose uniformly at random $I_i \subseteq \overline{I}$ such that rank $\left(\mathbf{G}'_{[I_i:\overline{J}]}\right) = |I_i| = b_i$;
    $I \leftarrow I \cup I_i$;
    $\mathbf{G}' \leftarrow \mathbf{G}'\mathbf{A}$ where $\mathbf{A} \in GL_{k+1}(\mathbb{F}_q)$ is such that $\mathbf{A}_{[J:]} = [\mathbf{Id}_\mu | \mathbf{0}]$ and $\mathbf{G}'_{[I_i:]}\mathbf{A}_{[:J]} = \mathbf{0}$ and $\mathbf{G}'_{[I_i:]}\mathbf{A}_{[:\overline{J}]} = [\mathbf{Id}_{b_i} \mid \mathbf{0}]$;
    `// Stop condition`
    if $i = N$ **then**
      repeat r_{N+1} **times**
        choose uniformly at random $Z \in \overline{I}$ such that $|Z| = \ell$;
        choose uniformly at random a balanced bipartition $I_L \sqcup I_R$ of $[\![1, k+1]\!]$; `/* it can be different from` $[\![1, \frac{k+1}{2}]\!] \sqcup [\![\frac{k+1}{2} + 1, k+1]\!]$ `*/`
        $(\mathbf{G}'_L, \mathbf{G}'_R) \leftarrow \left(\mathbf{G}'_{[Z:I_L]}, \mathbf{G}'_{[Z:I_R]}\right)$;
        initialize a lookup table $\mathcal{T}$;
        for $\boldsymbol{u} \in \mathbb{F}_q^{(k+1)/2}$ *such that* $\mathsf{fnze}(\boldsymbol{u}) = 1$ *and* $|\boldsymbol{u}| = p$ **do**
          $\alpha \leftarrow \mathsf{fnze}(-\mathbf{G}'_L \boldsymbol{u})$;
          append $\alpha^{-1}\boldsymbol{u}$ in $\mathcal{T}\left[-\alpha^{-1}\mathbf{G}'_L\boldsymbol{u}\right]$;
        for $\boldsymbol{v} \in \mathbb{F}_q^{(k+1)/2}$ *such that* $\mathsf{fnze}(\boldsymbol{v}) = 1$ *and* $|\boldsymbol{v}| = p$ **do**
          $\alpha \leftarrow \mathsf{fnze}(\mathbf{G}'_R \boldsymbol{v})$;
          for $\boldsymbol{u} \in \mathcal{T}\left[\alpha^{-1}\mathbf{G}'_R\boldsymbol{v}\right]$ **do**
            $\boldsymbol{r} \leftarrow \mathbf{G}'_{[\overline{I}:I_L]}\boldsymbol{u} + \alpha^{-1}\mathbf{G}'_{[\overline{I}:I_R]}\boldsymbol{v}$;
            if $|\boldsymbol{r}| = t - 2p$ **then**
              return $\boldsymbol{c} \in \mathbb{F}_q^n$ *such that* $(\boldsymbol{c}_I)_{I_L} = \boldsymbol{u}$ *and* $(\boldsymbol{c}_I)_{I_R} = \alpha^{-1}\boldsymbol{v}$ *and* $\boldsymbol{c}_{\overline{I}} = \boldsymbol{r}$;
    else
      `// Recursive call`
      return *PROJ-MAGE-STERN-PROCEDURE*$(i+1, \mathbf{G}', I)$;
  return $\perp$;

Theorem 7.1. *Algorithm 7.1 succeeds, that is, returns an output different from* $\perp$*, with probability*

$$\mathbb{P}_{\text{success}}\left(\textit{Proj-MAGE-Stern}\right) = \frac{q-1}{q}\left(1 - q_0(0)\right) \tag{7.3}$$

where $q_0(0)$ *is defined by the same recursion as in Theorem 5.6, specialized to* $\delta = 0$ *and with* k *replaced by* $k+1$*.*

Proof. The proof follows the same argument as that of Theorem 5.6, with three modifications. First, the factor $\frac{q-1}{q}$ and the replacement of k by $k+1$ in the definition of $q_0(0)$ arise from the reduction in Theorem 3.2. Second, the branching process starts from a single individual rather than from k individuals (that is $\boldsymbol{Z}_0 = (1, 0, \cdots, 0)$) because we are no longer in the DOOM setting; consequently, the exponent k applied to $q_0(0)$ in Theorem 5.6 disappears. Finally, we set $\delta = 0$, which is the optimal choice in the non-DOOM setting, as it balances the two lists in the meet-in-the-middle step.

We now adapt the cost of a single execution of Algorithm 7.1 to the projective-space framework.

Proposition 7.2. *The time complexity of one execution of Algorithm 7.1 is*

$$T_{\text{Proj-MAGE-Stern}} \stackrel{\text{def}}{=} \sum_{i=1}^{N} \left(\prod_{j=1}^{i} r_j \right) T_{\text{Gauss}}(i) + \left(\prod_{i=1}^{N+1} r_i \right) \left(T_{\text{list}} + T_{\text{check}} \right). \tag{7.4}$$

Its memory complexity is

$$M_{\text{Proj-MAGE-Stern}} = (k+1)n + M \cdot p \left\lceil \log_q \left(\tfrac{k+1}{2} \right) \right\rceil \ \textit{symbols of } \mathbb{F}_q, \tag{7.5}$$

where

$$M \stackrel{\text{def}}{=} \binom{(k+1)/2}{p} (q-1)^{p-1},$$

$$T_{\text{Gauss}}(i) \stackrel{\text{def}}{=} \mathsf{GAUSS}_q \left(k+1, n - \sum_{j=1}^{i-1} b_j, b_i \right),$$

$$\begin{aligned} T_{\text{list}} \stackrel{\text{def}}{=}\ & (k+1)\mathsf{VAdd}_q(\ell) + (M - 1 + p)\,\mathsf{VAdd}_q(\ell) \\ & + M \cdot \left(p + \mathsf{VProd}_q(\ell) + p \left\lceil \log_q \left(\tfrac{k+1}{2} \right) \right\rceil + \mathsf{SVRAM}_q \left(\tfrac{k+1}{2}, p, M \right) \right), \end{aligned}$$

$$\begin{aligned} T_{\text{check}} \stackrel{\text{def}}{=}\ & (M - 1 + p)\,\mathsf{VAdd}_q(\ell) \\ & + M \cdot \left(1 + \mathsf{VProd}_q(\ell) + \mathsf{SVRAM}_q \left(\tfrac{k+1}{2}, \tfrac{M}{q^\ell} p, q^\ell \right) \right) \\ & + \frac{M^2(q-1)}{q^\ell} \left(\mathsf{MSVProd}_q \left(\tfrac{q}{q-1}(t - 2p + 1), k+1, 2p \right) \right). \end{aligned}$$

Proof. The proof follows essentially the same argument as that of Proposition 5.2, with four modifications.

First, the factor k and the term $T_{\boldsymbol{y}'_Z}$ appearing in the meet-in-the-middle part of Equation (5.1) in Proposition 5.2 are absent here, since $\boldsymbol{y} = \mathbf{0}$ and there is only one target vector to decode rather than k.

Second, we replace k by $k+1$ because of the reduction of Theorem 3.2 and set $\delta = 0$ because we are no longer in the DOOM setting.

Third, working in projective space reduces the list sizes by a factor of $q-1$ and having $\delta = 0$ induces $M_{\mathcal{T}} = M' \stackrel{\text{def}}{=} M$.

Finally, the additional terms

$$p + \mathsf{VProd}_q(\ell) \qquad \text{and} \qquad 1 + \mathsf{VProd}_q(\ell)$$

arise from the normalization of the projective representatives. This normalization requires computing α^{-1}, at unit cost, multiplying a sparse vector of weight p by α^{-1}, at cost $p-1$ since its first nonzero entry is already equal to 1, and multiplying a vector of length ℓ by α^{-1}, at cost $\mathsf{VProd}_q(\ell)$. The total normalization cost is therefore $p + \mathsf{VProd}_q(\ell)$. The term $1 + \mathsf{VProd}_q(\ell)$ is obtained similarly when only the inversion and the multiplication of the length-ℓ vector are required. □

7.1 Regular Projective MAGE-Stern

In the regular decoding problem t-$\mathsf{Dec}_q(n,k,t)$, the error vector is partitioned into t regular blocks of size $\frac{n}{t}$, each containing exactly one nonzero coordinate. To solve this variant of the decoding problem, we use a regular guessing strategy similar to that of [25]. More precisely, we modify Algorithm 7.1 so that, at recursion level i, the set I_i is sampled uniformly as a disjoint union

$$I_i^{(1)} \sqcup \cdots \sqcup I_i^{(t)} = I_i \tag{7.6}$$

where, for all $j \in [\![1,t]\!]$,

$$\left|I_i^{(j)}\right| = \frac{b_i}{t}, \tag{7.7}$$

$$I_i^{(j)} \subseteq \left[\!\!\left[\tfrac{(j-1)n}{t} + 1, \tfrac{jn}{t}\right]\!\!\right] \setminus \left(I_i^{(1)} \cup \cdots \cup I_i^{(j-1)}\right). \tag{7.8}$$

This regular sampling strategy modifies the transition probabilities of the branching process.

Proposition 7.3. *Let $\mathcal{P}$, $(J_i)_{i\in[\![0,N]\!]}$, and $\left(\boldsymbol{Z}_i\right)_{i\in[\![1,N+1]\!]}$ be defined as in Definition 5.3, specialized to $\delta = 0$. Then $\left(\boldsymbol{Z}_i\right)_{i\in[\![1,N+1]\!]}$ is a multitype branching process in a varying environment with initial population $\boldsymbol{Z}_0 = (1,0,\cdots,0)$ and offspring distribution*

$$\boldsymbol{X}^{(i,\tau)} \stackrel{\text{def}}{=} \left(X^{(i,\tau)}(\sigma)\right)_{\sigma\in\mathcal{P}} \sim \text{Multinomial}(r_i; (P_i(\tau,\sigma))_{\sigma\in\mathcal{P}}) \tag{7.9}$$

where

$$P_i(\tau,\sigma) \stackrel{\text{def}}{=} \mathbb{P}\left(|\boldsymbol{e}_{J_i}| = \sigma \;\middle|\; \left|\boldsymbol{e}_{J_{i-1}}\right| = \tau\right), \qquad 1 \le i \le N \tag{7.10}$$

are given by

$$P_i(\tau,\sigma) = \begin{cases} 1 & \text{if } \tau = \sigma = \perp \\ 0 & \text{if } \sigma < \tau \\ \binom{t-\tau}{\sigma-\tau}\left(\frac{b_i}{n-\sum_{j=1}^{i-1} b_j}\right)^{\sigma-\tau}\left(1-\frac{b_i}{n-\sum_{j=1}^{i-1} b_j}\right)^{t-\sigma} & \text{if } \tau \le \sigma < \perp \\ 1-\sum_{j=\tau}^{2p} P_i(\tau,j) & \text{if } \tau < \sigma = \perp \end{cases}$$

and

$$P_{N+1}(\tau,\sigma) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \tau \neq 2p \text{ and } \sigma = \perp \\ 0 & \text{if } \tau \neq 2p \text{ and } \sigma \neq \perp \\ 0 & \text{if } \tau = 2p \text{ and } \sigma \notin \{2p, \perp\} \\ \frac{\binom{(k+1)/2}{p}^2}{\binom{k+1}{2p}} \cdot \frac{\binom{n-k-1-2p}{\ell}}{\binom{n-k-1}{\ell}} & \text{if } \tau = \sigma = 2p \\ 1 - P_{N+1}(2p,2p) & \text{if } \tau = 2p \text{ and } \sigma = \perp. \end{cases}$$

Proof. Let $i \in [\![1, N]\!]$ and $\tau, \sigma \in [\![0, 2p]\!]$, with $\tau \le \sigma$. Conditioned on $\left|\boldsymbol{e}_{J_{i-1}}\right| = \tau$, exactly $t - \tau$ regular blocks have not yet had their unique error coordinate selected. In each such block, the probability that this coordinate is selected at level i is

$$1 - \frac{\binom{\left(n-\sum_{j=1}^{i-1} b_j\right)/t-1}{b_i/t}}{\binom{\left(n-\sum_{j=1}^{i-1} b_j\right)/t}{b_i/t}} = \frac{b_i}{n-\sum_{j=1}^{i-1} b_j}. \tag{7.11}$$

Since the coordinate selections are independent across the regular blocks, the number of newly selected error coordinates at level i follows the distribution

$$\text{Binomial}\left(t-\tau\,;\ \frac{b_i}{n-\sum_{j=1}^{i-1} b_j}\right). \tag{7.12}$$

Consequently,

$$\begin{aligned} P_i(\tau,\sigma) &\stackrel{\text{def}}{=} \mathbb{P}\left(|\boldsymbol{e}_{J_i}| = \sigma \;\middle|\; \left|\boldsymbol{e}_{J_{i-1}}\right| = \tau\right), \\ &= \mathbb{P}\left(\text{Binomial}\left(t-\tau; \frac{b_i}{n-\sum_{j=1}^{i-1} b_j}\right) = \sigma-\tau\right), \\ &= \binom{t-\tau}{\sigma-\tau}\left(\frac{b_i}{n-\sum_{j=1}^{i-1} b_j}\right)^{\sigma-\tau}\left(1-\frac{b_i}{n-\sum_{j=1}^{i-1} b_j}\right)^{t-\sigma}. \end{aligned}$$

The remaining transition probabilities follow by grouping all weights greater than $2p$ into the type $\perp$. The treatment of the final level is the same as in Proposition 5.5, after replacing k by $k+1$ and setting $\delta = 0$. □

Remark 7.4. For simplicity, we assume that each b_i is divisible by t, so that the same number of coordinates is selected from every regular block. When this divisibility condition does not hold, one may distribute the b_i coordinates as evenly as possible among the blocks, assigning either $\lfloor b_i/t \rfloor$ or $\lceil b_i/t \rceil$ coordinates to each of them. The probability of selecting the error coordinate then depends slightly on the block, and the binomial distribution used above must be replaced by the corresponding Poisson-binomial distribution. The formulas above may nevertheless be used as an approximation obtained by replacing these block-dependent probabilities by their average value.

Finally, the success probability of Projective MAGE-Stern in the regular setting is obtained from Theorem 7.1, using the transition probabilities P_i established in Proposition 7.3. The cost of a single execution is unchanged from the non-regular setting and is therefore given by Proposition 7.2.

8 Application to Pseudorandom Correlation Generators

We now assess the security of PCG-based protocols by evaluating Projective MAGE-Stern and its regular variant on the reference parameter sets proposed in [10]. These protocols rely on the hardness of either the decoding problem $\mathsf{Dec}_q(n,k,t)$ or its regular counterpart $t\text{-}\mathsf{Dec}_q(n,k,t)$.

We compare our results with the best currently known attacks. In particular, we consider the estimator of [46], introduced in [31], which incorporates classical ISD algorithms [36,23,5], the dual attack of [16], and the algebraic and hybrid attacks targeting the regular setting [11,43]. We additionally include the Reduce-and-Prange algorithm of [30], which is not currently covered by this estimator.

8.1 Projective MAGE-Stern in the Large-Field PCG Regime

A distinctive feature of the PCG parameter sets proposed in [10] is their very large field size, namely $q = 2^{128}$. In classical ISD algorithms, enumerating vectors of weight p supported on $\frac{k}{2}$ coordinates requires $\binom{k/2}{p}(q-1)^p$ operations. Consequently, when $q = 2^{128}$, even the choice $p = 1$ introduces an additional factor of approximately 2^{128}, making the enumeration step prohibitively expensive. Practical ISD variants are therefore effectively restricted to $p = 0$, in which case they reduce to Prange's algorithm. This behavior is consistent with the asymptotic result of [14], according to which all ISD algorithms become asymptotically equivalent to Prange as $q \to \infty$.

Projective Stern [18] fundamentally changes this situation by enumerating equivalence classes under multiplication by nonzero scalars rather than individual vectors. The corresponding enumeration cost becomes $\binom{k/2}{p}(q-1)^{p-1}$,

yielding a gain of a factor $q-1$. In particular, choosing $p=1$ no longer incurs any field-size-dependent multiplicative factor and therefore becomes computationally feasible.

For $p=1$, the success probability of Projective Stern is

$$\mathbb{P}_{\text{success}}\,(\text{Proj-Stern}) = \frac{\binom{k/2}{1}^2\binom{n-k-\ell}{t-2}}{\binom{n}{t}}, \tag{8.1}$$

whereas that of Prange's algorithm is

$$\mathbb{P}_{\text{success}}\,(\text{Prange}) = \frac{\binom{n-k}{t}}{\binom{n}{t}}. \tag{8.2}$$

Projective Stern thus improves the success probability over Prange by the factor

$$G = \frac{\mathbb{P}_{\text{success}}\,(\text{Proj-Stern})}{\mathbb{P}_{\text{success}}\,(\text{Prange})} = \frac{k^2\binom{n-k-\ell}{t-2}}{4\binom{n-k}{t}}. \tag{8.3}$$

Since $p=1$, the enumeration cost remains essentially linear in k, and Gaussian elimination remains the dominant contribution to the cost of one iteration. Reducing this cost is therefore particularly beneficial, as already observed for Reduce-and-Prange. Our Projective MAGE-Stern algorithm, which combines Projective Stern with multilevel amortized Gaussian elimination, is therefore particularly well suited here. The resulting improvement relies on two complementary mechanisms: the projective representation increases the success probability by making a nonzero value of p affordable, while multilevel amortized Gaussian elimination reduces the cost of each iteration. Projective MAGE-Stern can therefore outperform Prange-based approaches over a broad range of large-field PCG parameters.

8.2 Our Results for PCG Protocols

The resulting security estimates are reported in Tables 8.1, 8.2, 8.3, and 8.4.

The results confirm the analysis above. Projective MAGE-Stern provides the best attack complexity for most of the parameter sets proposed in [10]. In particular, it improves upon Reduce-and-Prange by up to 10 bits and upon the best attacks selected by the estimator of [46] by up to 6 bits.

Remark 8.1. For the parameter sets reported in Table 8.3, the estimator of [46] does not support instances with $n \geq 2^{16}$. We were therefore unable to obtain the corresponding security estimates for these instances.

Although the factor G in Equation (8.3) suggests a potential gain of approximately $\log_2(G)$ bits over Reduce-and-Prange, the practical improvement of Projective MAGE-Stern is smaller. For example, for the first parameter set of Table 8.1, $\log_2(G) = 11.72$, whereas the observed gain is only about 6 bits. This discrepancy arises because Reduce-and-Prange can exploit the exact knowledge

n	k	t	**Reduce-and-Prange [30]**	**Estimator of [31]**	**Projective MAGE-Stern**
2^{10}	652	57	105	103.65	**98.79**
2^{12}	1589	98	94	92.73	**89.23**
2^{14}	3482	198	97	97.21	**92.64**
2^{16}	7391	389	99	101.97	**96.00**
2^{18}	15336	760	108	105.13	**99.22**
2^{20}	32771	1419	104	107.05	**102.67**
2^{22}	67440	2735	107	108.90	**105.67**

Table 8.1: Attack costs for the PCG decoding instances $\mathsf{Dec}_q(n,k,t)$ of [10] over $\mathbb{F}_{2^{128}}$ ($\log_2$ scale).

n	k	t	**Reduce-and-Prange [30]**	**Estimator of [31]**	**Projective MAGE-Stern**
2^{12}	$0.75 \cdot 2^{12}$	44	111	108.70	**105.11**
2^{14}	$0.75 \cdot 2^{14}$	39	107	103.29	**100.91**
2^{16}	$0.75 \cdot 2^{16}$	34	104	99.04	**97.78**
2^{18}	$0.75 \cdot 2^{18}$	32	106	100.92	**100.73**
2^{20}	$0.75 \cdot 2^{20}$	31	110	**104.81**	105.35
2^{22}	$0.75 \cdot 2^{22}$	30	114	**108.75**	109.56
2^{24}	$0.75 \cdot 2^{24}$	29	118	**112.71**	114.13

Table 8.2: Attack costs for the dual PCG decoding instances $\mathsf{Dec}_q(n,k,t)$ of [10] over $\mathbb{F}_{2^{128}}$ ($\log_2$ scale).

n	k	t	**Reduce-and-Prange [30]**	**Estimator of [31]**	**Projective MAGE-Stern**
2^{10}	652	57	105	97.61	**94.99**
2^{12}	1589	98	91	92.74	**88.26**
2^{14}	3482	198	94	95.62	**92.24**
2^{16}	7391	389	97	?	**95.77**
2^{18}	15336	760	100	?	99.17
2^{20}	32771	1419	102	?	102.83
2^{22}	67440	2735	105	?	105.67

Table 8.3: Attack costs for the PCG regular decoding instances t-$\mathsf{Dec}_q(n,k,t)$ of [10] over $\mathbb{F}_{2^{128}}$ ($\log_2$ scale).

n	k	t	**Reduce-and-Prange [30]**	**Estimator of [31]**	**Projective MAGE-Stern**
2^{12}	$0.75 \cdot 2^{12}$	44	107	108.70	**104.24**
2^{14}	$0.75 \cdot 2^{14}$	39	105	103.29	**100.72**
2^{16}	$0.75 \cdot 2^{16}$	34	101	99.05	**97.65**
2^{18}	$0.75 \cdot 2^{18}$	32	104	100.91	**100.75**
2^{20}	$0.75 \cdot 2^{20}$	31	110	**104.81**	105.36
2^{22}	$0.75 \cdot 2^{22}$	30	120	**108.75**	109.62
2^{24}	$0.75 \cdot 2^{24}$	29	124	**112.71**	114.13

Table 8.4: Attack costs for the dual PCG regular decoding instances $t\text{-}\mathsf{Dec}_q(n,k,t)$ of [10] over $\mathbb{F}_{2^{128}}$ ($\log_2$ scale).

that the error vanishes on the selected blocks to reduce both the rows and the columns of the matrix at each recursion level. On contrary, Projective MAGE-Stern can only reduce the rows of the matrix but not the columns. Consequently, its Gaussian-elimination factorization is less effective, and the overall gain falls short of that predicted solely from the increase in success probability.

Acknowledgments. This work was funded by the French Agence Nationale de la Recherche through ANR-22-CE39-0004-01 DECODE for KC, VH and LL and ANR-22-PETQ-0008 PQ-TLS for JPT.

We would also like to thank Alain Couvreur for insightful discussions on attacks against PCG protocols.

References

1. Agresti, A.: On the extinction times of varying and random environment branching processes. Journal of Applied Probability **12**(1), 39–46 (1975). https://doi.org/10.2307/3212405
2. Aguilar Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Bos, J., Deneuville, J.C., Dion, A., Gaborit, P., Lacan, J., Persichetti, E., Robert, J.M., Véron, P., Zémor, G., Bos, J.: HQC. Round 4 Submission to the NIST Post-Quantum Cryptography Call (Oct 2022), https://pqc-hqc.org/
3. Athreya, K.B., Ney, P.E.: Branching Processes. Springer Berlin, Heidelberg (1972). https://doi.org/https://doi.org/10.1007/978-3-642-65371-1
4. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'Cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 92–122. Springer International Publishing, Cham (2021)
5. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How 1+1 = 0 improves information set decoding. In: Advances in Cryptology - EUROCRYPT 2012. LNCS, Springer (2012)
6. Bernstein, D.J., Chou, T.: Cryptattacktester: high-assurance attack analysis. In: Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part VI. p. 141–182. Springer-Verlag, Berlin, Heidelberg (2024). https://doi.org/10.1007/978-3-031-68391-6_5, https://doi.org/10.1007/978-3-031-68391-6_5

7. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. In: Post-Quantum Cryptography 2008. LNCS, vol. 5299, pp. 31–46 (2008)
8. Both, L., May, A.: Optimizing BJMM with Nearest Neighbors: Full Decoding in $2^{2/21n}$ and McEliece Security. In: WCC Workshop on Coding and Cryptography (Sep 2017), http://wcc2017.suai.ru/Proceedings{_}WCC2017.zip
9. Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security. In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography 2018. LNCS, vol. 10786, pp. 25–46. Springer, Fort Lauderdale, FL, USA (Apr 2018), https://doi.org/10.1007/978-3-319-79063-3_2
10. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 896–912. CCS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243868, https://doi.org/10.1145/3243734.3243868
11. Briaud, P., Øygarden, M.: A new algebraic approach to the regular syndrome decoding problem and implications for PCG constructions. Cryptology ePrint Archive, Paper 2023/176 (2023), https://eprint.iacr.org/2023/176, https://eprint.iacr.org/2023/176
12. Bui, D., Couteau, G.: Improved private set intersection for sets with small entries. In: Public-Key Cryptography – PKC 2023. pp. 190–220. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-31371-4_7
13. Canteaut, A., Chabaud, F.: A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. IEEE Trans. Inform. Theory **44**(1), 367–378 (1998). https://doi.org/10.1109/18.651067, http://dx.doi.org/10.1109/18.651067
14. Canto Torres, R.: Asymptotic analysis of ISD algorithms for the $q-$ary case. In: Proceedings of the Tenth International Workshop on Coding and Cryptography WCC 2017 (Sep 2017)
15. Canto-Torres, R., Sendrier, N.: Analysis of information set decoding for a sub-linear error weight (2015), preprint
16. Carrier, K., Debris-Alazard, T., Meyer-Hilfiger, C., Tillich, J.: Statistical decoding 2.0: Reducing decoding to LPN. In: Advances in Cryptology - ASIACRYPT 2022. LNCS, Springer (2022), https://eprint.iacr.org/2022/1000
17. Carrier, K., Debris-Alazard, T., Meyer-Hilfiger, C., Tillich, J.P.: Reduction from sparse LPN to LPN, dual attack 3.0. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI. LNCS, vol. 14656, pp. 286–315. Springer (2024). https://doi.org/10.1007/978-3-031-58754-2_11, https://doi.org/10.1007/978-3-031-58754-2_11, artifact available at https://artifacts.iacr.org/eurocrypt/2024/a10/
18. Carrier, K., Hatey, V., Tillich, J.P.: Projective space Stern decoding and application to SDitH. In: Applied Cryptography and Network Security Workshops: ACNS 2024 Satellite Workshops, AIBlock, AIHWS, AIoTS, SCI, AAC, SiMLA, LLE, and CIMSS, Abu Dhabi, United Arab Emirates, March 5–8, 2024, Proceedings, Part II. p. 29–52. Springer-Verlag, Berlin, Heidelberg (2024). https://doi.org/10.1007/978-3-031-61489-7_3, https://doi.org/10.1007/978-3-031-61489-7_3
19. Cui, H., Wang, X., Yang, K., Yu, Y.: Actively secure half-gates with minimum overhead under duplex networks. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 35–67. Springer Nature Switzerland, Cham (2023)

20. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Authenticated garbling from simple correlations. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. pp. 57–87. Springer Nature Switzerland, Cham (2022)
21. Dumer, I.: On minimum distance decoding of linear codes. In: Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory. pp. 50–52. Moscow (1991)
22. Esser, A., Bellini, E.: Syndrome decoding estimator. In: IACR International Conference on Public-Key Cryptography. pp. 112–141. Springer (2022)
23. Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017. LNCS, vol. 10402, pp. 486–514. Springer, Santa Barbara, CA, USA (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_17, https://doi.org/10.1007/978-3-319-63715-0_17
24. Esser, A., May, A., Zweydinger, F.: McEliece needs a break–solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 433–457. Springer (2022)
25. Esser, A., Santini, P.: Not just regular decoding: Asymptotics and improvements of regular syndrome decoding attacks. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024. pp. 183–217. Springer Nature Switzerland, Cham (2024)
26. Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022. LNCS, vol. 13508, pp. 541–572. Springer (2022), https://eprint.iacr.org/2022/188
27. Franzese, N., Katz, J., Lu, S., Ostrovsky, R., Wang, X., Weng, C.: Constant-overhead zero-knowledge for RAM programs. p. 178–191. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484800, https://doi.org/10.1145/3460120.3484800
28. Guo, Q., Johansson, T., Nguyen, V.: A new sieving-style information-set decoding algorithm. IEEE Trans. Inform. Theory **70**(11), 8303–8319 (2024). https://doi.org/10.1109/TIT.2024.3457150, https://doi.org/10.1109/TIT.2024.3457150
29. Harris, T.E.: The Theory of Branching Processes. Springer Berlin, Heidelberg (1963)
30. Kim, J., Lee, C.: Reduce and prange: Revisiting prange's ISD for solving LPN/RSD over large fields. Cryptology ePrint Archive, Paper 2024/276 (2024), https://eprint.iacr.org/2024/276
31. Liu, H., Wang, X., Yang, K., Yu, Y.: The hardness of LPN over any integer ring and field for PCG applications. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024. pp. 149–179. Springer Nature Switzerland, Cham (2024)
32. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $O(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology - ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer (2011)
33. Orgoványi, V., Simon, K.: Multitype branching processes in random environments with not strictly positive expectation matrices. Bernoulli **32**(2), 1328 – 1355 (2026). https://doi.org/10.3150/25-BEJ1908, https://doi.org/10.3150/25-BEJ1908
34. Peters, C.: Information-set decoding for linear codes over $\mathbf{F}_q$. In: Post-Quantum Cryptography 2010. LNCS, vol. 6061, pp. 81–94. Springer (2010)
35. Peters, C.: Curves, Codes, and Cryptography. Ph.D. thesis, Technische Universiteit Eindhoven (2011)
36. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962). https://doi.org/10.1109/TIT.1962.1057777, http://dx.doi.org/10.1109/TIT.1962.1057777

37. Raghuraman, S., Rindal, P.: Blazing fast PSI from improved OKVS and subfield VOLE. p. 2505–2517. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560658, https://doi.org/10.1145/3548606.3560658
38. Rindal, P., Schoppmann, P.: Vole-psi: Fast OPRF and circuit-PSI from vector-OLE. In: Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part II. p. 901–930. Springer-Verlag, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-030-77886-6_31, https://doi.org/10.1007/978-3-030-77886-6_31
39. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: Improved constructions and implementation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 1055–1072. CCS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3363228, https://doi.org/10.1145/3319535.3363228
40. Sendrier, N.: Decoding one out of many. In: Post-Quantum Cryptography 2011. LNCS, vol. 7071, pp. 51–67 (2011)
41. Stern, J.: A method for finding codewords of small weight. In: Cohen, G.D., Wolfmann, J. (eds.) Coding Theory and Applications. LNCS, vol. 388, pp. 106–113. Springer (1988)
42. Wang, T., Wang, A., Yang, K., Liu, H., Yu, Y., Zhang, J., Wang, X.: A hybrid algorithm for the regular syndrome decoding problem. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025. pp. 466–497. Springer Nature Singapore, Singapore (2026)
43. Wang, T., Wang, A., Yang, K., Liu, H., Yu, Y., Zhang, J., Wang, X.: A hybrid algorithm for the regular syndrome decoding problem. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025. pp. 466–497. Springer Nature Singapore, Singapore (2026)
44. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1074–1091 (2021). https://doi.org/10.1109/SP40001.2021.00056
45. Yu, T., Jiang, H., Wang, H., Chen, R., Cheng, Q., Huang, X., Zhu, Y.: Progressive sieving-style information-set decoding algorithm. Cryptology ePrint Archive, Paper 2026/633 (2026), https://eprint.iacr.org/2026/633
46. Yu, Y., Yang, K., Wang, X., Wang, A., Wang, T., Liu, H., Hao, X., Li, J.: Estimator of LPN problems over any finite fields and power-of-two rings for PCG and MPC applications. https://lpnestimator.com (2025)

A Complexity of Progressive Sieving-ISD in the DOOM setting

In this appendix, we reformulate the complexity expressions for the Progressive-DOOM-Sieving algorithm and the ANNS-Predicate-DOOM (Asymmetric Nearest Neighbor Search with Predicate for DOOM) subroutine, which are used to solve $\mathsf{DOOM}_2(n, k, w)$ [45, Algorithms 4 and 5]. We do not reproduce the algorithms here and refer the reader to [45, Section 4.2 and Appendix G] for a

detailed description. We adopt the notation of [45], while adapting the complexity expressions to the logic-gate Model 2.1.

We first introduce the following two modifications to the original Progressive Sieving-ISD algorithm:

(i) At the first level of [45, Algorithm 4], we apply ANNS-Predicate-2-DOOM directly, rather than first applying ANNS-Predicate-1-DOOM, since the improvement provided by the latter is negligible.
(ii) In [45, Algorithm 5], we store only the list L'_0 in the hash tables, while the lists L'_i, for $1 \le i \le k$, are generated on the fly during the checking step. This eliminates the table-precomputation cost, corresponding to the term T^i_{table} in [45, Equation (4.4)], and also reduces the memory complexity.

We may then adapt [45, Theorem 4.2] to obtain the time and memory complexities of Progressive Sieving-ISD in the DOOM setting, under Model 2.1 and incorporating modifications (i) and (ii). They are given by

$$T_{\text{ISD}}^{\text{DOOM}} \stackrel{\text{def}}{=} \frac{T_{\text{gauss}}^{\text{DOOM}} + T_{\text{Sieve}}^{\text{DOOM}} + T_{\text{check}}^{\text{DOOM}}}{P_{\text{outer}}^{\text{DOOM}}} \tag{A.1}$$

and

$$M_{\text{ISD}}^{\text{DOOM}} \stackrel{\text{def}}{=} M_{\text{sieve}}^{\text{DOOM}} + M_{\text{gauss}}^{\text{DOOM}} \tag{A.2}$$

where

$$T_{\text{gauss}}^{\text{DOOM}} = \mathsf{GAUSS}_2\left(n-k, n+k, n-k-\ell\right), \tag{A.3}$$

$$T_{\text{Sieve}}^{\text{DOOM}} = \frac{1}{\prod_{i=1}^{\ell'} p_i^{\text{DOOM}}} \cdot \sum_{i=1}^{\ell'} T_i^{\text{DOOM}}, \tag{A.4}$$

$$T_{\text{check}} = k \cdot \frac{N_{\ell'}}{2k} \cdot \mathsf{MSVProd}_q\left(2(w-\widehat{w}+1), k+\ell, \widehat{w}\right), \tag{A.5}$$

$$P_{\text{outer}}^{\text{DOOM}} \stackrel{\text{def}}{=} k \cdot \frac{\binom{k+\ell}{\widehat{w}}\binom{n-k-\ell}{w-\widehat{w}}}{\binom{n}{w}} \cdot \min\left(1\,;\, \frac{N_{\ell'}}{2k} \cdot \frac{2^\ell}{\binom{k+\ell}{\widehat{w}}}\right), \tag{A.6}$$

$$M_{\text{gauss}}^{\text{DOOM}} = (n-k)\cdot(n+k), \tag{A.7}$$

$$M_{\text{sieve}}^{\text{DOOM}} = \max_{1\le i\le \ell'}\left(M_i^{\text{DOOM}}\right). \tag{A.8}$$

Here, p_i^{DOOM} is defined in [45, Equation 4.6]. It remains to specify T_i^{DOOM} and M_i^{DOOM} which denote the time and memory complexities of ANNS-Predicate-(2,3)-DOOM at level i.

Under modification (ii), for every $i \in [\![1, \ell']\!]$, the memory complexity is

$$M_i^{\text{DOOM}} = \left(\frac{N_{i-1}}{2}(2w_{i-1}-1) + \binom{w_{i-1}}{w_{i-1}-w_i/2}\cdot w_{i-1} + \frac{N_i}{2}(2w_i-1)\right)\lceil \log_2(k+\ell)\rceil\,. \tag{A.9}$$

Moreover, under modification (i), the time complexity at level i is

$$T_i = \begin{cases} T_{\mathrm{bucket},i} + T_{\mathrm{check},1,i} + T_{\mathrm{check},2,i} & \text{if } i < \ell', \\ T_{\mathrm{bucket},i} + T_{\mathrm{check},2,i} & \text{if } i = \ell'. \end{cases} \tag{A.10}$$

where

$$\begin{aligned}
A_i &\stackrel{\text{def}}{=} \binom{w_{i-1}}{w_{i-1} - w_i/2}, \\
B_i &\stackrel{\text{def}}{=} \binom{w_{i-1} - 1}{w_{i-1} - w_i/2}, \\
C_i &\stackrel{\text{def}}{=} \frac{N_{i-1}}{2^{t_i+1}} \cdot \frac{\binom{w_{i-1}}{w_{i-1}-w_i/2}}{\binom{k+\ell}{w_{i-1}-w_i/2}}, \\
T_{\mathrm{bucket},i} &\stackrel{\text{def}}{=} \frac{N_{i-1}}{2}\Big(A_i \cdot (\mathsf{SVRAM}_2\,(k+\ell, w_{i-1}, A_i) + p_{i-1}\,\lceil \log_2(k+\ell) \rceil) \\
&\qquad + \mathsf{MSVProd}_2\,(t_i, k+\ell, w_{i-1}) \Big), \\
T_{\mathrm{check},1,i} &\stackrel{\text{def}}{=} \frac{N_{i-1}}{2}\Big(\mathsf{MSVProd}_2\,(t_i, k+\ell, w_{i-1}) \\
&\qquad + A_i \cdot \mathsf{SVRAM}_2\,(k+\ell, C_i \cdot w_{i-1}, A_i) \\
&\qquad\qquad + A_i \cdot C_i \cdot \mathsf{SVAdd}_2\,(k+\ell, w_{i-1}) \Big), \\
T_{\mathrm{check},2,i} &\stackrel{\text{def}}{=} \frac{N_{i-1}}{2}\Big(\mathsf{MSVProd}_2\,(t_i, k+\ell, w_{i-1} - 1) \\
&\qquad + \mathsf{VAdd}_2\,(k+\ell) \\
&\qquad\quad + B_i \cdot \mathsf{SVRAM}_2\,(k+\ell, C_i \cdot w_{i-1}, A_i) \\
&\qquad\qquad + B_i \cdot C_i \cdot \mathsf{SVAdd}_2\,(k+\ell, w_{i-1}) \Big).
\end{aligned}$$

Finally, the parameters of the algorithm must be selected carefully during the optimization process so that the resulting list sizes satisfy the constraints given in [45, Equations (4.7), (4.8), (4.9) and (4.10)].