

Sampling fixed weight vectors via unranking in HQC

Gustavo Zamboni

Daniel Panario

Ricardo Custódio

Abstract—Sampling binary vectors of fixed Hamming weight is a core subroutine in code-based cryptography. Current methods are based on rejection sampling, shuffling or sorting networks, which consume a variable amount of randomness or are slightly biased, requiring tweaks to security arguments and careful constant-time implementations. To address this, we present two samplers based on combinatorial unranking, applied to HQC. The algorithms are deterministic, sample uniformly at random and require the optimal amount of random bits. We analyze their performance overhead on the standard KEM operations, and discuss constant-time implementation strategies. We also discuss an alternative decapsulation procedure via the complementary ranking function, which does not rely on seed expansion via SHAKE256.

Keywords. Randomness consumption · Enumerative coding · HQC

I Introduction. Let $n, k \in \mathbb{Z}_{>0}$ and $\mathbb{F}_2$ be the finite field of two elements. We denote by V_n the n -dimensional vector space over $\mathbb{F}_2$, and $V_{n,k} = \{\mathbf{v} \in V_n : |\text{supp}(\mathbf{v})| = k\}$. In computational terms, the elements of V_n are binary words, and those in $V_{n,k}$ have “fixed (Hamming) weight”. There exist several algorithms for sampling fixed-weight vectors uniformly at random, with extensive applications throughout cryptography [1], [2], [3], [4], [5]. The authors of [6] provide an overview of methods for sampling fixed-weight polynomials, which are equivalent to elements in $V_{n,k}$ if the underlying ring is $\mathbb{F}_2[X]/(X^n - 1)$. The authors of [7] give specialized algorithms for the sparse case, where $k = o(n)$. HQC [8], selected for standardization by NIST, is in this regime.

In HQC, secret vectors $\mathbf{x}, \mathbf{y} \in V_{n,\omega}$ are sampled during key generation [8, p. 23]. Ephemeral vectors $\mathbf{r}_1, \mathbf{r}_2 \in V_{n,\omega_r}$ and $\mathbf{e} \in V_{n,\omega_e}$ are generated during encapsulation [8, p. 24], and regenerated during decapsulation as part of the Fujisaki–Okamoto re-encryption check [8, p. 28]. Two fixed-weight vector samplers are employed [8, p. 13]: `SampleFixedWeightVects` is based on rejection sampling and used on key generation, while `SampleFixedWeightVect` is a slightly biased version based on [3] and used elsewhere, due to timing attacks described in [9]. This bias marginally impacts the security of the scheme [8, §6.2.3].

Combinatorial (un)ranking algorithms may be used to sample such structured mathematical objects uniformly at random. Let S be a finite set, $N = |S|$, and $s, t \in S$ such that $s \neq t$. We denote $[N] = \{0, \dots, N-1\}$. A *ranking function* is a bijection $\mathbf{R} : S \rightarrow [N]$ that defines a total order $\prec_{\text{ord}}$ on the elements of S , i.e. $s \prec_{\text{ord}} t$ if and only if $\mathbf{R}(s) < \mathbf{R}(t)$ [10, §2.1]. Given a ranking function, the *unranking function* is the inverse bijection $\mathbf{U} = \mathbf{R}^{-1}$, i.e., $\mathbf{U} : [N] \rightarrow S$ such that $\mathbf{R}(s) = i$ if and only if $\mathbf{U}(i) = s$ for all $i \in [N]$. To sample from S , simply generate a number x from $[N]$ uniformly at random and evaluate $\mathbf{U}(x)$.

II Model. We draw inspiration from the recent advances in [7], as well as connections of combinatorial ranking to several distinct cryptographic algorithms (cf. applications to PERK [11], format-preserving encryption [12], and hash-based signatures [13]). We apply our proposal to HQC, and sketch our rationale below.

It is well known that every integer $r \in [0, \binom{n}{k})$ can be decomposed as $r = \sum_{i=1}^k \binom{c_i}{i}$, where the sequence $(c_k, \dots, c_1)$ is strictly

decreasing. This combinatorial number system [14, p. 360] yields k -subsets of $\{0, \dots, n-1\}$ in lexicographic order. A greedy unranking algorithm $\mathbf{U}_{\text{lex}}$ is immediate for this order, where one must find the largest c_i that fits the partial rank [15].

To make $\mathbf{U}_{\text{lex}}$ constant-time, we rearrange Pascal’s rule as $\binom{n-1}{k} = \binom{n}{k} - \binom{n-1}{k-1}$. Throughout the algorithm, we maintain a sequence $\alpha = (\binom{n'}{j})_{j=0}^k$ where $n' \leq n$ is the remaining length to be processed. At every step of $\mathbf{U}_{\text{lex}}$, α is updated via the above subtractive identity, and c_i is selected via an oblivious linear scan. However, long integer arithmetic is required for most iterations, and the performance of this solution is not competitive.

Other orders give rise to (un)ranking algorithms with distinct characteristics. For instance, a “divide-and-conquer” strategy is obtained by making use of the identity $\binom{n}{k} = \sum_{j=0}^{\min(k, \lfloor \frac{n}{2} \rfloor)} \binom{\lfloor \frac{n}{2} \rfloor}{j} \cdot \binom{\lceil \frac{n}{2} \rceil}{k-j}$ (usually attributed to Vandermonde), which yields a “recursive block” order, with the associated algorithm hereafter denoted as $\mathbf{U}_{\text{rb}}$. In contrast to $\mathbf{U}_{\text{lex}}$, long integers are required only for the first levels of the recursion tree, greatly improving the performance of $\mathbf{U}_{\text{rb}}$. However, converting $\mathbf{U}_{\text{rb}}$ to constant-time is not as simple due to its unbalanced recursion, which depends on the rank.

We replace both `SampleFixedWeightVects` (PKE.Keygen, Lines 3–4 [8, p. 23]) and `SampleFixedWeightVect` (PKE.Encrypt, Lines 6–8 [8, p. 24] and PKE.Decrypt, Line 3 [8, p. 25]) by $\mathbf{U}_{\text{lex}}$ (constant-time) and $\mathbf{U}_{\text{rb}}$ (variable-time), and study the cryptographic engineering challenges of this approach. Put simply, we squeeze a number B of bytes from the XOF, reduce modulo $\binom{n}{k}$, and unrank. We ensure that $\binom{n}{k} \cdot 2^{-8B} < 2^{-2\lambda}$ where λ is the security level of the HQC parameter set.

Because unranking is a bijection, the complementary ranking function $\mathbf{R}$ enables an alternative decapsulation key format (hereafter called **dd**): store the rank of the secret vector $\mathbf{y}$ directly, recovering it without XOF invocation and improving randomness consumption very slightly. We propose the replacement of `dkPKE` (a 32-byte seed) in the specification by a rank $r_{\mathbf{y}}$ of $\lceil \lg \binom{n}{\omega} \rceil$ bits, derived in PKE.Keygen, Line 3 [8, p. 23] and recovered via $\mathbf{R}$ in PKE.Decrypt, Line 3 [8, p. 25].

Technically, $r_{\mathbf{y}}$ is not an independent random value, but a deterministic reformatting of the existing seed derivation. In other words, `ctxPKE.dk` produces $\lceil \frac{B}{8} \rceil$ bytes via `XOF.GetBytes`, which are reduced modulo $\binom{n}{\omega}$ to yield $r_{\mathbf{y}}$. Since $r_{\mathbf{y}}$ is deterministically derivable from `seedKEM`, the compressed format `dkKEM` = (`seedKEM`) and keypair checking via `seedKEM` [8, p. 22] are both unaffected.

This modification preserves the IND-CCA2 bound of [8, Theorem 6.3]: the $\text{SFO}^\perp$ proof treats PKE.Decrypt as a black box requiring only δ -correctness and IND-CPA security, both of which hold since the distribution of $\mathbf{y}$ over $V_{n,\omega}$ is unchanged. We note that, under this change, $\mathbf{x}$ becomes irrecoverable from $r_{\mathbf{y}}$ alone, but $\mathbf{x}$ is not used in PKE.Decrypt or KEM.Decaps.

Additionally, the security argument in [8, §6.2.3] can be simplified. The correction factors $\tau_{\min}^\omega, \tau_{\max}^\omega$ that modify the IND-CCA2 bound

TABLE I
HQC PERFORMANCE (IN KILOCYCLES, AVERAGE OF 100 MEASUREMENTS) AND RANDOMNESS CONSUMPTION (IN BYTES).

Instance	Operation	Performance (reference implementation)			Performance (AVX2 implementation)			XOF usage		
		Baseline	U_{rb}	U_{lex}	Baseline	U_{rb}	U_{lex}	Baseline	U	$U+dd$
HQC-128	Keygen	2 229	2 449	35 744	43	143	33 344	2 653	2 481	2 481
	Encaps	4 425	4 817	67 046	88	268	62 271	3 109	2 545	2 545
	Decaps	6 746	7 187	86 101	202	381	79 293	3 307	2 657	2 545
HQC-192	Keygen	6 667	7 127	164 702	105	295	156 782	5 141	4 874	4 874
	Encaps	13 330	14 184	311 620	205	585	296 246	5 850	4 986	4 986
	Decaps	20 306	21 172	394 644	399	785	374 594	6 153	5 154	4 986
HQC-256	Keygen	15 536	16 386	442 770	210	541	435 700	8 055	7 717	7 717
	Encaps	30 965	32 361	839 082	416	1 022	819 259	8 993	7 877	7 877
	Decaps	47 269	48 991	1 083 524	837	1 477	1 035 190	9 386	8 101	7 877

by $(\tau_{\max}^{\omega_r})^3$ become unnecessary. We propose to replace both samplers by a single uniform bijective mapping, so the bound of [8, Theorem 6.3] applies directly without τ correction factors. We argue this is valuable for simpler mechanized verification of the security proofs with an assistant such as EasyCrypt.

III Experiments. We modified the [official HQC implementation](#) to implement our changes on top of commit 161cd4fd. All benchmarks are run in an AMD Ryzen™ 7 9700X @ 3.8GHz in a GNU/Linux environment (kernel 6.19.6) and clang 22.1.1, otherwise using default compilation flags of the official implementation. We observed a consistent and significant (up to 50%) reduction in cycle counts by switching from gcc to clang. We discuss Table I below.

The reference implementation overhead for U_{rb} ranges from 5.5% to 9.9%, and decreases with the security level. For HQC-256, the sampling overhead is amortized into the measurement. On the AVX2 implementation, U_{rb} introduces a factor of 2.5–3.3× on key generation and encapsulation, but only 1.8–2.0× on decapsulation. The wider relative gap on AVX2 is structural: the baseline sampler benefits from vectorized sorting networks, while the multi-precision arithmetic at the top two recursion levels of U_{rb} remains scalar.

The constant-time variant U_{lex} incurs higher overhead: 16× on the HQC-128 reference implementation, growing to 28× at HQC-256. On AVX2, the slowdown exceeds two orders of magnitude relative to the baseline. We note that U_{lex} on AVX2 is only marginally faster than on the reference implementation, confirming that the bottleneck is the sequential long integer computation, which does not benefit from SIMD vectorization.

While U_{lex} is not competitive for deployment in its current form, it demonstrates that constant-time unranking is achievable in principle. Randomness consumption decreases under our samplers for all operations, most pronounced in encapsulation (−18% for HQC-128). The dd variant further reduces decapsulation consumption to match that of encapsulation, since the XOF invocation for regenerating y is eliminated entirely.

IV Next steps. This is work in progress. We are extending our study to other cryptosystems, which use similar sampling procedures, such as BIKE, NTRU and Classic McEliece. We plan to implement constant-time U_{rb} by converting each recursion level to oblivious iteration over all weight splits with masked selection, combined with constant-time long integer division at the top levels. We also expect to explore alternatives to constant-time arithmetic in the form of [cryptoint](#) and [s2n-bignum](#), or constant-time algorithm implementations via [Jasmin](#).

References.

- [1] N. Drucker and S. Gueron, “Generating a Random String with a Fixed Weight,” in *Cyber Security Cryptography and Machine Learning (CSCML 2019)* (S. Dolev, D. Hendler, S. Lodha, and M. Yung, eds.), vol. 11527 of *Lecture Notes in Computer Science*, pp. 141–155, June 2019.
- [2] A. Barengi and G. Pelosi, “Constant weight strings in constant time: a building block for code-based post-quantum cryptosystems,” in *CF ’20: Proceedings of the 17th ACM International Conference on Computing Frontiers*, pp. 132–141, May 2020.
- [3] N. Sendrier, “Secure Sampling of Constant-Weight Words - Application to BIKE,” *Cryptology ePrint Archive*, Paper 2021/1631, 2021.
- [4] D. L. G. Filho, T. S. R. Silva, and J. López, “Efficient isochronous fixed-weight sampling with applications to NTRU,” *IACR Communications in Cryptology*, vol. 1, July 2024.
- [5] M. López-García, D. G. Farouk-Marei, and E. Cantó-Navarro, “Converting Fixed-Length Binary Strings Into Constant Weight Words: Application on Post-Quantum Cryptography,” *IEEE Transactions on Dependable and Secure Computing*, vol. 22, pp. 3063–3074, Dec. 2024.
- [6] M. Krausz, G. Land, J. Richter-Brockmann, and T. Güneysu, “A Holistic Approach Towards Side-Channel Secure Fixed-Weight Polynomial Sampling,” in *PKC 2023* (A. Boldyreva and V. Kolesnikov, eds.), vol. 13941 of *Lecture Notes in Computer Science*, pp. 94–124, May 2023.
- [7] D. L. G. Filho, G. Adj, S. Bettaieb, A. Budroni, J. Chávez-Saab, and F. Rodríguez-Henríquez, “Sota Voce: Low-Noise Sampling of Sparse Fixed-Weight Vectors,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2026, pp. 353–375, Jan. 2026.
- [8] P. Gaborit, C. Aguilar-Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, E. Persichetti, G. Zémor, J. Bos, A. Dion, J. Lacan, J.-M. Robert, P. Véron, P. L. Barreto, S. Ghosh, S. Gueron, T. Güneysu, R. Misoczki, J. Richter-Brockmann, N. Sendrier, J.-P. Tillich, and V. Vasseur, “Hamming Quasi-Cyclic (HQC),” Aug. 2025.
- [9] Q. Guo, C. Hlaschek, T. Johansson, N. Lahr, A. Nilsson, and R. L. Schröder, “Don’t Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, pp. 223–263, June 2022.
- [10] D. L. Kreher and D. R. Stinson, *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, 1st ed., 1998.
- [11] S. Bettaieb, A. Budroni, M. Palumbi, and D. L. G. Filho, “Quasi-optimal Permutation Ranking and Applications to PERK,” in *Progress in Cryptology – AFRICACRYPT 2024* (S. Vaudenay and C. Petit, eds.), vol. 14861 of *Lecture Notes in Computer Science*, pp. 49–65, July 2024.
- [12] S. Miracle and S. Yilek, “New Algorithms and Analyses for Sum-Preserving Encryption,” in *ASIACRYPT 2022* (S. Agrawal and D. Lin, eds.), vol. 13793 of *Lecture Notes in Computer Science*, pp. 3–31, Dec. 2022.
- [13] Z. Qin and S. Sun, “Extending the SPHINCS+ Framework: Varying the Tree Heights and Chain Lengths,” *Cryptology ePrint Archive*, Paper 2025/2236, Dec. 2025.
- [14] D. E. Knuth, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 1st ed., 2011.
- [15] T. M. Cover, “Enumerative source encoding,” *IEEE Transactions on Information Theory*, vol. 19, pp. 73–77, Jan. 1973.