

Combinatorial ranking algorithms for the general linear group

Gustavo Zambonin¹[0000–0002–6962–9347], Tiago Siqueira¹,
Larissa Gremelmaier Rosa¹[0009–0000–1238–2489], Rodrigo Stefanés¹,
Ricardo Custódio¹[0000–0001–9611–5694], and Daniel Panario²[0000–0003–3551–4063]

¹ Universidade Federal de Santa Catarina, Florianópolis, SC, 88040-900, Brazil
{gustavo.zambonin,larissa.gremelmaier.rosa}@posgrad.ufsc.br,
{tiago.siqueira,rodrigo.stefanes}@grad.ufsc.br, ricardo.custodio@ufsc.br

² Carleton University, Ottawa, ON, K1S 5B6, Canada
daniel@math.carleton.ca

Abstract. The general linear group over a finite field is a fundamental object in mathematics and computer science, with applications ranging from computational algebra to quantum-safe cryptography. Generating its elements uniformly at random is a common requirement, addressed in practice by the standard rejection sampling approach. However, it may repeatedly fail on small finite fields, resulting in suboptimal random bit complexity. To overcome this limitation, we introduce novel combinatorial (un)ranking algorithms for the general linear group, based on a mixed-radix number system. We give a formal description of the resulting lexicographical order, related to the sequence of certain linear algebra operations. We prove the correctness of our algorithms and analyze their computational cost in terms of field operations. Finally, we provide efficient implementations of our algorithms, benchmarking them against rejection sampling and other sampling methods using parameters relevant to cryptographic applications. We show that the performance of (un)ranking is competitive, while guaranteeing optimal random bit complexity.

Keywords: Random generation · Invertible matrices · Finite fields · Combinatorial enumeration

1 Introduction

The general linear group is central to the study of vector spaces in linear algebra [1], and its properties are of interest in computational group theory [14] and combinatorics [22]. Generating elements from this group, particularly uniformly at random, is a practical necessity in many computational domains, and several solutions exist in the literature.

Sampling algorithms. A frequently employed technique is rejection sampling (“rej”), at the heart of standard computer algebra systems such as GAP [15]. However, it can be inefficient; the probability that a random $n \times n$ matrix over a

binary field is invertible converges to ≈ 0.2887 as $n \rightarrow \infty$ [21]. Hence, a simple rejection sampling approach would discard over 71% of the generated candidates. Other approaches include constructive algebraic methods, which avoid this issue.

The generic subgroup algorithm [6] (“**subgroup**”) is a recursive method relying on sampling from different sets of coset representatives organized in a “chain”. Randall’s method (“**nonsing**”) [24] is a specialization of **subgroup** that builds LUP factors directly. There also exist algorithms based on sampling complete flags over a finite field. The method in [23, Algorithm 5.3.2] (“**schubert**”) is based on choosing Schubert cells and then random elements inside them. On the other hand, the method in [23, Algorithm 5.4.3] (“**bruhat**”) is a straightforward implementation of a Bruhat decomposition, which generalizes the LUP decomposition.

Another strategy involves ranking, which maps a combinatorial object to a unique integer (its rank). The complementary unranking function performs the inverse operation [17]. The authors of [2] provide a generic framework for (un)ranking any word in a regular language, using a deterministic finite automaton (DFA). However, a DFA recognizing invertible matrices must maintain the linear span of the columns processed thus far as its state. Since the number of distinct subspaces grows exponentially with its dimension, such a DFA has exponentially many states, rendering the generic (un)ranking algorithms computationally intractable for this problem.

Cryptographic applications. Invertible matrices are used in cryptographic protocols based on the matrix code equivalence (MCE) hard problem [4], which informally asks to determine whether two matrix codes are equivalent under the action of $\text{GL}_m(q) \times \text{GL}_n(q)$ and the standard notion of linear equivalence. Further, they are also featured in cryptography based on multivariate quadratic (MQ) equations [27], which can be equivalently described as matrices. Hence, sampling from $\text{GL}_n(q)$ is a common primitive required by several quantum-safe cryptographic schemes (cf. [3,25,18] for other works).

Heuristic methods are proposed in the cryptography literature, but they do not sample uniformly from $\text{GL}_n(q)$. Additive repair [27, Algorithm 1] generates a random matrix M and, if singular, modifies it via a perturbation $M \leftarrow M + aS$, for all $a \in \mathbb{F}_q$ and S a fixed invertible matrix. This rejects the singular sample and selects values from a specific additive coset, skewing the distribution. Similarly, constructing $M = LU$ with uniformly chosen components [25, Section 6.2], where L is lower-triangular and U is upper-triangular, can only generate matrices with non-zero leading principal minors. Extending this to $M = PLU$, with P a uniform random permutation, does not correct the bias, as the size of the subset associated with P (the Bruhat cell) varies with its number of inversions.

In implementations of cryptographic algorithms, randomness is derived from a seed via primitives such as hash functions or ciphers. The cost of their secure evaluation is non-negligible [10], and fresh randomness requirements may constrain hardware designs [8,19]. Furthermore, rejection sampling exhibits variable bit consumption per execution, which may leak information through timing side-channels [13]. A deterministic algorithm that consumes a fixed amount of entropy facilitates constant-time implementation, mitigating this class of attacks.

Therefore, in addition to computational cost, we are interested in reducing the *random bit complexity* of a sampling algorithm, i.e., the total number of random bits consumed [16, Section 3.4.1.G].

Contributions. We bridge the gap between algebraic generation methods and combinatorial enumeration by developing ranking and unranking algorithms for the general linear group. Specifically, we (i) propose algorithms for (un)ranking invertible matrices over a finite field that are optimal with respect to random bit complexity; (ii) prove their correctness and present a cost analysis in terms of finite field arithmetic; (iii) provide fast implementations and performance measurements, comparing our methods against standard techniques in the context of cryptographic applications.

2 Preliminaries

Let $\mathbb{F}_{p^d}$ be the finite field with $q = p^d$ elements, where p is a prime and $d \in \mathbb{Z}_{>0}$. Whenever clear from the context, we denote the field simply by $\mathbb{F}_q$. Field elements are represented as formal polynomials in the variable x of degree $d - 1$. We let $\mathbb{F}_q^n$ be the standard vector space over $\mathbb{F}_q$ of dimension $n \in \mathbb{Z}_{>0}$. Our main object of interest is $\text{GL}_n(q)$, the group of $n \times n$ invertible matrices over $\mathbb{F}_q$ equipped with the usual matrix multiplication. We give the necessary background to understand our combinatorial algorithms in the following.

Notation. We denote the set of integers $\{0, \dots, d - 1\}$ by $[d]$, and the order of a group G by $\text{ord}(G)$. For a sequence s we denote by s_i its i -th element. For any matrix M , we denote by $M_{i,*}$ its i -th row, and by $M_{*,j}$ its j -th column. We denote by $\mathbf{e}_r$ the standard basis vector, where the r -th coordinate is 1 and all others are 0. The identity matrix of size n is $\mathbf{I}_n$, and the $m \times n$ zero matrix is $\mathbf{0}_{m \times n}$. We let Id be the identity function and adopt the convention that an empty product equals one.

We read $X \stackrel{f}{\leftarrow} S$ as “ X is sampled uniformly at random from S ”. In sections of pseudocode, variables before and after the semicolon are “input” and “output” arguments, respectively. Finally, we denote by $\prec_{\text{lex}}$ the standard lexicographic (resp. $\prec_{\text{rev}}$ for reverse lexicographic) order with respect to the underlying combinatorial object. We now recall some definitions used throughout our work.

Definition 1. Let V be a vector space over a field $\mathbb{F}_q$ and $B \subseteq V$. A linear combination of elements of B is a finite sum of the form $\sum_{i=0}^{k-1} \alpha_i v_i$, where $k \in \mathbb{N}$, $\alpha_i \in \mathbb{F}_q$, and $v_i \in B$. The set of all linear combinations of B is called the span of B , denoted by $\langle B \rangle$.

The set B is *linearly independent* (LI) if for every finite subset $\{v_0, \dots, v_{k-1}\} \subseteq B$, the only solution to $\sum_{i=0}^{k-1} \alpha_i v_i = 0$ is $\alpha_0 = \dots = \alpha_{k-1} = 0$. If $\langle B \rangle = V$ and B is linearly independent, then B is a *basis* of V . An *ordered basis* for V is a basis B equipped with a total order. We refer to the cardinality of any basis B as the *dimension* of V , denoted $\dim(V)$.

Theorem 1 ([7, p. 413]). *The order of $\text{GL}_n(q)$ is $\prod_{i=0}^{n-1} (q^n - q^i)$.*

Intuitively, any vector $v_1 \neq 0$ can be picked from $\mathbb{F}_q^n$ as the first choice, giving $(q^n - 1)$ possibilities. The second vector must be outside $\langle v_1 \rangle$, resulting in $(q^n - q)$ choices. More generally, when choosing the $(i + 1)$ -th vector, the previous i rows span an i -dimensional subspace containing q^i vectors, so the next vector must be selected outside this subspace, giving $(q^n - q^i)$ possibilities. The product of all the possible choices yields the result.

We recall that the quantity of Theorem 1 also counts the number of ordered bases for $\mathbb{F}_q^n$. Furthermore, any linearly independent set can be *extended* to a basis of the underlying vector space [1, Theorem 2.32], and all bases of a vector space have the same cardinality [1, Theorem 2.34]. These facts will prove useful in developing our results; we now provide some combinatorial definitions.

Definition 2 ([17, Section 2.1]). *Let S be a finite set, $N = |S|$ and consider any distinct $s, t \in S$. 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 $\mathbf{R}(s) < \mathbf{R}(t)$. 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 $i \in [N]$.*

We now discuss *mixed radix systems*, a method to map integer tuples to specific ranges of integers. The following result is standard in the discipline and is the foundation for several combinatorial (un)ranking frameworks.

Lemma 1. *Let $n \in \mathbb{Z}_{>0}$. For any $\mathbf{q} = (q_0, \dots, q_{n-1}) \in \mathbb{Z}_{\geq 2}^n$, the function*

$$\begin{aligned} \psi_{\mathbf{q}} : \prod_{i=0}^{n-1} [q_i] &\rightarrow \left[\prod_{i=0}^{n-1} q_i \right], \\ (k_0, \dots, k_{n-1}) &\mapsto \sum_{i=0}^{n-1} k_i \prod_{j=0}^{i-1} q_j \end{aligned}$$

is a bijection.

The sequence $\mathbf{q}$ is denoted as a *mixed radix basis*. The inverse $\psi_{\mathbf{q}}^{-1}(k) = (k_0, \dots, k_{n-1})$ can be computed iteratively: let $K^{(0)} = k$, and for $i \in \{0, \dots, n-1\}$, calculate

$$k_i = K^{(i)} \pmod{q_i} \quad \text{and} \quad K^{(i+1)} = \lfloor \frac{K^{(i)}}{q_i} \rfloor.$$

For the special case $q_0 = \dots = q_{n-1} = q$, we denote the map simply by ψ_q , and the representation simplifies to the standard base- q expansion

$$\psi_q(k_0, \dots, k_{n-1}) = k_0 + k_1 q + \dots + k_{n-1} q^{n-1}.$$

Remark 1. Mixed radix systems can be used to map the coefficient vector of a field element $f = \sum a_i x^i \in \mathbb{F}_{p^d}$ to the vector space $\mathbb{F}_p^d$. Ranking is simply $\mathbf{R}_p(f) = \psi_p(a_0, \dots, a_{d-1})$. The unranking $\mathbf{U}_p(k)$ is evaluated via the inverse $\psi_p^{-1}(k)$, whose output is the coefficient vector of a polynomial in the finite field.

The function ψ induces the $\prec_{\text{rex}}$ order on all coordinate tuples of $\mathbb{F}_q^n$, where k_0 is the least significant component. (Of course, if the input tuple is reversed, the induced order is the usual $\prec_{\text{lex}}$.) The following result formalizes how an ordered basis preserves this order on the vectors of the space itself. We then define ranking functions for elements of a vector space with respect to any ordered basis.

Lemma 2. *Let V be an n -dimensional vector space over $\mathbb{F}_q$ and any distinct $v, w \in V$. Given an ordered basis $B = \{b_0, \dots, b_{n-1}\}$ for V , the bijection*

$$\begin{aligned} \phi_B : V &\rightarrow \mathbb{F}_q^n \\ \sum_{i=0}^{n-1} v_i b_i &\mapsto (v_0, \dots, v_{n-1}) \end{aligned}$$

induces a total order $\prec_B$ on V defined as $v \prec_B w$ if and only if $\phi_B(v) \prec_{\text{rex}} \phi_B(w)$.

Definition 3. *Let V be an n -dimensional vector space over $\mathbb{F}_q$, and take any ordered basis B for V . Then, the ranking function of vectors in V to $\prec_B$ is defined as*

$$\begin{aligned} \mathbf{R}_V : V &\rightarrow [q^n] \\ v &\mapsto \psi_q(\phi_B(v)). \end{aligned}$$

The corresponding unranking function is defined as

$$\begin{aligned} \mathbf{U}_V : [q^n] &\rightarrow V \\ k &\mapsto \phi_B^{-1}(\psi_q^{-1}(k)). \end{aligned}$$

In other words, a vector $v = \sum_{i=0}^{n-1} v_i b_i \in V$ can be ranked by computing $\psi_q(\mathbf{R}_P(v_0), \dots, \mathbf{R}_P(v_{n-1}))$. Unranking is done by first decomposing the rank $k \in [q^n]$ into component ranks $(k_0, \dots, k_{n-1}) = \psi_q^{-1}(k)$, unranking each component to a field element $v_i = \mathbf{U}_P(k_i)$ and reconstructing $v = \sum v_i b_i$. Since the (un)ranking functions are defined as compositions of bijections by Lemmas 1 and 2, we have the following result.

Corollary 1. $\mathbf{U}_V \circ \mathbf{R}_V = \text{Id}$ and $\mathbf{R}_V \circ \mathbf{U}_V = \text{Id}$.

3 Novel (un)ranking algorithms

Our method for (un)ranking elements of $\text{GL}_n(q)$ is built in layers, following the combinatorial argument that yields $\text{ord}(\text{GL}_n(q))$. Constructing an invertible matrix (or an ordered basis) vector-by-vector corresponds to a sequence of dependent choices from successively smaller sets. This sequential structure can be captured by a mixed-radix number system. This suggests a twofold strategy:

- (i) methods to (un)rank a single vector from the set of available choices at each step, which is the complement of a subspace, $V \setminus U$; and
- (ii) methods to (un)rank an entire ordered basis by interpreting its global rank as a mixed-radix number, where each digit corresponds to a choice made at one of the layers.

3.1 Canonical basis extension

The ability to rank vectors relative to a subspace U requires a deterministic way to impose an order on all vectors in the ambient space V . Our approach relies on extending a basis of U to a full basis of V in a canonical way. This is achieved by taking an ordered basis for U and augmenting it with vectors from the standard basis of $\mathbb{F}_q^n$ in a fixed order, a standard procedure in linear algebra. The algorithm for this canonical basis completion is formalized in Algorithm 1. We assume the existence of a procedure **RREF** that computes the unique row-reduced echelon form of a matrix M .

C_B ($S; B$)	SELECTPIVOTS (M', i, S, B, l, s)
1. Set $i \leftarrow S$. 2. If $i \geq n$, set $B \leftarrow S$ and exit. 3. Set $M \leftarrow [S \mid \mathbf{I}_n]$. 4. Set $M' \leftarrow \text{RREF}(M)$. 5. Set $l \leftarrow 0$. 6. Set $s \leftarrow -1$. 7. Call SELECTPIVOTS(M', i, S, B, l, s).	1. For $r \in 0, \dots, n-1$: 1A. For $c \in s+1, \dots, i+n-1$: - A1. If $M'_{r,c} = 0$, go to Step 1A. - A2. If $c < i$, set $B_{*,l} \leftarrow S_{*,c}$. - A3. If $c \geq i$, set $B_{*,l} \leftarrow \mathbf{e}_{c-i}$. - A4. Set $l \leftarrow l+1$. - A5. Set $s \leftarrow c$. - A6. Go to Step 1.

Algorithm 1. Helper algorithm to complete a set of vectors to a unique basis.

We first show that Algorithm 1 induces a particular order on the vectors of U , ensuring that the procedure is deterministic.

Lemma 3. *Let U be a subspace of $V = \mathbb{F}_q^n$ and let $E(U)$ be the basis constructed by Algorithm 1. The first $q^{\dim(U)}$ vectors of V under the order $\prec_{E(U)}$ are precisely the vectors in U .*

Proof. The coordinate map $\phi_{E(U)}$ maps vectors to their coordinate tuples. By construction, the first $\dim(U)$ vectors of the basis $E(U)$ form a basis for U . Therefore, any vector $u \in U$ can be written as a linear combination that only uses these first basis vectors. Its coordinate tuple will have zeros in all positions corresponding to the basis vectors from $V \setminus U$. Conversely, any vector $v \notin U$ must have at least one non-zero coordinate corresponding to a basis vector not in the basis of U . In reverse lexicographic order, any tuple with non-zero entries in higher positions is larger than any tuple with zeros in those same positions. Thus, all vectors in U have smaller ranks than any vector in $V \setminus U$. $\square$

3.2 Vector space complements

With a canonical way to order all vectors, we can formally define functions to (un)rank elements from the complement of a subspace U . The core insight is that since all vectors in U come first in the ordering, we can rank vectors in $V \setminus U$ by taking their standard rank and subtracting an offset equal to the size of U .

Definition 4. Let V be an n -dimensional vector space over $\mathbb{F}_q$, let U be an i -dimensional subspace of V , and let $E(U)$ be the basis for V constructed by Algorithm 1. Then, the ranking function of vectors in $V \setminus U$ to $\prec_B$ is defined as

$$\begin{aligned} \mathbf{R}_C : V \setminus U &\rightarrow [q^n - q^i] \\ v &\mapsto \mathbf{R}_V(E(U), v) - q^i. \end{aligned}$$

The corresponding unranking function is

$$\begin{aligned} \mathbf{U}_C : [q^n - q^i] &\rightarrow V \setminus U \\ k &\mapsto \mathbf{U}_V(E(U), k + q^i). \end{aligned}$$

To make the computation explicit, we present the pseudocode for these functions in Algorithm 2 before proving their correctness.

$\mathbf{U}_C(r, S, \mathbb{F}_q^n; v)$	$\mathbf{R}_C(v, S, \mathbb{F}_q^n; r)$
1. Set $i \leftarrow S $.	1. Set $i \leftarrow S $.
2. Set $F \leftarrow \mathbf{C}_B(S)$.	2. Set $F \leftarrow \mathbf{C}_B(S)$.
3. Set $r' \leftarrow r + q^i$.	3. Solve $F \cdot c = v$ for c .
4. Set $c \leftarrow \psi_q^{-1}(r')$.	4. Set $r' \leftarrow \psi_q(c)$.
5. Set $v \leftarrow F \cdot c$.	5. Set $r \leftarrow r' - q^i$.

Algorithm 2. Unranking (left) and ranking (right) algorithms for vectors outside a subspace S to $\prec_{\text{rex}}$.

Remark 2. The choice of $\prec_{\text{rex}}$ simplifies the description of Algorithm 2. We recall that Algorithm 1 places the basis vectors of S in the first i columns, which correspond to the least significant digits under this order. Consequently, the vectors in the subspace S map to the contiguous interval $[q^i]$, allowing the complement to be (un)ranked by a simple offset. This is not the case if $\prec_{\text{lex}}$ is chosen, where the initial coordinates correspond to the most significant digits; the subspace elements are scattered across the desired integer range with stride q^{n-i} , turning the complement into a union of disjoint intervals.

Proposition 1. $\mathbf{U}_C \circ \mathbf{R}_C = \text{Id}$ and $\mathbf{R}_C \circ \mathbf{U}_C = \text{Id}$.

Proof. We mark variables used in ranking with (r) and in unranking with (u) .

$\mathbf{U}_C \circ \mathbf{R}_C = \text{Id}$. Take any vector $v^{(r)} \in V \setminus U$. Let its rank in the complement be $k^{(r)} = \mathbf{R}_C(U, v^{(r)})$. We unrank this value to get a new vector $v^{(u)}$:

$$\begin{aligned} v^{(u)} &= \mathbf{U}_C(U, k^{(r)}) \\ &= \mathbf{U}_C(U, \mathbf{R}_V(E(U), v^{(r)}) - q^i) \\ &= \mathbf{U}_V(E(U), (\mathbf{R}_V(E(U), v^{(r)}) - q^i) + q^i) \\ &= \mathbf{U}_V(E(U), \mathbf{R}_V(E(U), v^{(r)})) \\ &= v^{(r)}. \end{aligned} \tag{Corollary 1}$$

Hence, the claim holds.

$\mathbf{R}_C \circ \mathbf{U}_C = \text{Id}$. Take any integer $k^{(u)} \in [q^n - q^i]$. Let its corresponding vector be $v^{(u)} = \mathbf{U}_C(U, k^{(u)})$. We now rank this vector to get an integer $k^{(r)}$:

$$\begin{aligned}
 k^{(r)} &= \mathbf{R}_C(U, v^{(u)}) \\
 &= \mathbf{R}_C(U, \mathbf{U}_V(E(U), k^{(u)} + q^i)) \\
 &= \mathbf{R}_V(E(U), \mathbf{U}_V(E(U), k^{(u)} + q^i)) - q^i \\
 &= (k^{(u)} + q^i) - q^i \\
 &= k^{(u)}.
 \end{aligned}
 \tag{Corollary 1}$$

Hence, $\mathbf{R}_C$ and $\mathbf{U}_C$ are mutual inverses. $\square$

3.3 Ordered bases

Having established how to (un)rank vectors relative to a subspace, we now compose this operation to (un)rank entire ordered bases. This maps directly to a mixed-radix number system with radices

$$\mathbf{q} = (q^n - q^0, q^n - q^1, \dots, q^n - q^{n-1}).$$

A global rank $K \in [\text{ord}(\text{GL}_n(q))]$ can be uniquely decomposed into $(k_0, \dots, k_{n-1})$, where each k_i corresponds to the choice of the i -th basis vector.

Definition 5. Let V be an n -dimensional vector space over $\mathbb{F}_q$, and let $\mathcal{B}$ be the set of all ordered bases of V . Then, the ranking function of elements in $\text{GL}_n(q)$ to $\prec_{\text{rex}}$ is defined as

$$\begin{aligned}
 \mathbf{R}_{\text{GL}} : \mathcal{B} &\rightarrow [\text{ord}(\text{GL}_n(q))] \\
 (v_0, \dots, v_{n-1}) &\mapsto \psi_{\mathbf{q}}(k_0, \dots, k_{n-1}),
 \end{aligned}$$

where $k_i = \mathbf{R}_C(\langle v_0, \dots, v_{i-1} \rangle, v_i)$. The corresponding unranking function is

$$\begin{aligned}
 \mathbf{U}_{\text{GL}} : [\text{ord}(\text{GL}_n(q))] &\rightarrow \mathcal{B} \\
 K &\mapsto (v_0, \dots, v_{n-1}),
 \end{aligned}$$

where $(k_0, \dots, k_{n-1}) = \psi_{\mathbf{q}}^{-1}(K)$ and $v_i = \mathbf{U}_C(\langle v_0, \dots, v_{i-1} \rangle, k_i)$.

The formal definition shows that the computation of each vector v_i (and its corresponding digit k_i) depends on the previously constructed vectors. To make this sequential process explicit, we present the algorithms in Algorithm 3 before proving their correctness.

Proposition 2. $\mathbf{U}_{\text{GL}} \circ \mathbf{R}_{\text{GL}} = \text{Id}$ and $\mathbf{R}_{\text{GL}} \circ \mathbf{U}_{\text{GL}} = \text{Id}$.

Proof. We mark variables used in ranking with (r) and in unranking with (u) .

$\mathbf{U}_{\mathbf{GL}}(r, \mathbb{F}_q^n; M)$	$\mathbf{R}_{\mathbf{GL}}(M, \mathbb{F}_q^n; r)$
1. Set $r' \leftarrow r$.	1. Set $r \leftarrow 0$.
2. For $i \in (0, \dots, n-1)$:	2. Set $w \leftarrow 1$.
2A. Set $S_i \leftarrow (M_{*,0}, \dots, M_{*,i-1})$.	3. For $i \in (0, \dots, n-1)$:
2B. Set $N_i \leftarrow q^n - q^i$.	3A. Set $S_i \leftarrow (M_{*,0}, \dots, M_{*,i-1})$.
2C. Set $k_i \leftarrow r' \pmod{N_i}$.	3B. Set $k_i \leftarrow \mathbf{R}_{\mathbf{C}}(M_{*,i}, S_i, \mathbb{F}_q^n)$.
2D. Set $r' \leftarrow \lfloor \frac{r'}{N_i} \rfloor$.	3C. Set $r \leftarrow r + k_i \cdot w$.
2E. Set $M_{*,i} \leftarrow \mathbf{U}_{\mathbf{C}}(k_i, S_i, \mathbb{F}_q^n)$.	3D. Set $N_i \leftarrow q^n - q^i$.
	3E. Set $w \leftarrow w \cdot N_i$.

Algorithm 3. Unranking (left) and ranking (right) algorithms for elements of $\mathbf{GL}_n(q)$ to $\prec_{\text{rex}}$.

$\mathbf{U}_{\mathbf{GL}} \circ \mathbf{R}_{\mathbf{GL}} = \text{Id}$. Take any $B \in \mathcal{B}$. We let $\mathbf{R}_{\mathbf{GL}}(B) = \psi_{\mathbf{q}}(r_0^{(r)}, \dots, r_{n-1}^{(r)})$, called K for short, where $r_j^{(r)} = \mathbf{R}_{\mathbf{C}}(V_j^{(r)}, v_j^{(r)})$ and $V_j^{(r)} = \langle v_0^{(r)}, \dots, v_{j-1}^{(r)} \rangle$ with $j < n$. We let $\mathbf{U}_{\mathbf{GL}}(K) = (v_0^{(u)}, \dots, v_{n-1}^{(u)})$ be denoted by B' , which yields intermediate local ranks $(r_0^{(u)}, \dots, r_{n-1}^{(u)}) = \psi_{\mathbf{q}}^{-1}(K)$. First, we show that $r_j^{(u)} = r_j^{(r)}$:

$$\begin{aligned}
(r_0^{(u)}, \dots, r_{n-1}^{(u)}) &= \psi_{\mathbf{q}}^{-1}(K) \\
&= \psi_{\mathbf{q}}^{-1}(\psi_{\mathbf{q}}(r_0^{(r)}, \dots, r_{n-1}^{(r)})) \\
&= (r_0^{(r)}, \dots, r_{n-1}^{(r)}). \tag{Lemma 1}
\end{aligned}$$

Thus, $r_j^{(u)} = r_j^{(r)}$ for all $j \in [n]$. Now, we show that $v_j^{(u)} = v_j^{(r)}$ by induction. For the base case ($j = 0$), we have that $V_0^{(u)} = \langle \emptyset \rangle = \langle \emptyset \rangle = V_0^{(r)}$ by definition, and

$$\begin{aligned}
v_0^{(u)} &= \mathbf{U}_{\mathbf{C}}(V_0^{(u)}, r_0^{(u)}) \\
&= \mathbf{U}_{\mathbf{C}}(\langle \emptyset \rangle, r_0^{(r)}) \\
&= \mathbf{U}_{\mathbf{C}}(\langle \emptyset \rangle, \mathbf{R}_{\mathbf{C}}(V_0^{(r)}, v_0^{(r)})) \\
&= v_0^{(r)}. \tag{Proposition 1}
\end{aligned}$$

For the inductive step, we assume $v_i^{(u)} = v_i^{(r)}$ for $i < j$, which implies that $V_j^{(u)} = \langle v_0^{(u)}, \dots, v_i^{(u)} \rangle = \langle v_0^{(r)}, \dots, v_i^{(r)} \rangle = V_j^{(r)}$. Then, we have

$$\begin{aligned}
v_j^{(u)} &= \mathbf{U}_{\mathbf{C}}(V_j^{(u)}, r_j^{(u)}) \\
&= \mathbf{U}_{\mathbf{C}}(V_j^{(u)}, r_j^{(r)}) \\
&= \mathbf{U}_{\mathbf{C}}(V_j^{(r)}, \mathbf{R}_{\mathbf{C}}(V_j^{(r)}, v_j^{(r)})) \tag{inductive hypothesis} \\
&= v_j^{(r)}. \tag{Proposition 1}
\end{aligned}$$

Hence, $v_j^{(u)} = v_j^{(r)}$ for all $j \in [n]$. Consequently, $B' = (v_0^{(u)}, \dots, v_{n-1}^{(u)}) = (v_0^{(r)}, \dots, v_{n-1}^{(r)}) = B$, and the claim holds.

$\mathbf{R}_{\text{GL}} \circ \mathbf{U}_{\text{GL}} = \text{Id}$. Take any $K \in [\text{ord}(\text{GL}_n(q))]$. We let $\mathbf{U}_{\text{GL}}(K) = (v_0^{(u)}, \dots, v_{n-1}^{(u)})$, called B' for short, built via $(r_0^{(u)}, \dots, r_{n-1}^{(u)}) = \psi_{\mathbf{q}}^{-1}(K)$ and $v_j^{(u)} = \mathbf{U}_{\text{C}}(V_j^{(u)}, r_j^{(u)})$, where $V_j^{(u)} = \langle v_0^{(u)}, \dots, v_{j-1}^{(u)} \rangle$. We let $\mathbf{R}_{\text{GL}}(B') = \psi_{\mathbf{q}}(r_0^{(r)}, \dots, r_{n-1}^{(r)})$ be denoted by K' , computed as $r_j^{(r)} = \mathbf{R}_{\text{C}}(V_j^{(r)}, v_j^{(u)})$, where $V_j^{(r)} = \langle v_0^{(u)}, \dots, v_{j-1}^{(u)} \rangle$. We recall that $V_j^{(r)} = V_j^{(u)}$ and first show that $r_j^{(r)} = r_j^{(u)}$:

$$\begin{aligned}
r_j^{(r)} &= \mathbf{R}_{\text{C}}(V_j^{(r)}, v_j^{(u)}) \\
&= \mathbf{R}_{\text{C}}(V_j^{(u)}, v_j^{(u)}) \\
&= \mathbf{R}_{\text{C}}(V_j^{(u)}, \mathbf{U}_{\text{C}}(V_j^{(u)}, r_j^{(u)})) \\
&= r_j^{(u)}.
\end{aligned}
\tag{Proposition 1}$$

Thus, $(r_0^{(r)}, \dots, r_{n-1}^{(r)}) = (r_0^{(u)}, \dots, r_{n-1}^{(u)})$. Finally, we are able to show that

$$\begin{aligned}
K' &= \mathbf{R}_{\text{GL}}(B') \\
&= \psi_{\mathbf{q}}(r_0^{(r)}, \dots, r_{n-1}^{(r)}) \\
&= \psi_{\mathbf{q}}(r_0^{(u)}, \dots, r_{n-1}^{(u)}) \\
&= \psi_{\mathbf{q}}(\psi_{\mathbf{q}}^{-1}(K)) \\
&= K.
\end{aligned}
\tag{Lemma 1}$$

Hence, $\mathbf{R}_{\text{GL}}$ and $\mathbf{U}_{\text{GL}}$ are mutual inverses. $\square$

Figure 1 shows a visual demonstration of the induced $\prec_{\text{rex}}$ order on a small group.



Fig. 1. Enumeration of all elements of $\text{GL}_3(2)$; each column represents a matrix read left-to-right, top-to-bottom. Black squares are 1s and gray squares are 0s.

3.4 Cost analysis

We perform our analysis in the standard RAM model [5, Section 2.2], where the cost of field operations is not assumed to be constant but is taken in terms of the bit size of the field elements.

Cost of field operations. The cost depends on arithmetic in $\mathbb{F}_q$; we recall that elements are represented as polynomials. Addition requires $\mathcal{O}(d)$ operations in $\mathbb{F}_p$; multiplication requires $\mathcal{O}(d^2)$ operations, although FFT-based methods can reduce this cost [9]. Inversion of a nonzero element may be carried out using

the Extended Euclidean Algorithm for polynomials, which again takes $\mathcal{O}(d^2)$ operations. Since each arithmetic operation in $\mathbb{F}_p$ is polynomial in $\log p$, it follows that any arithmetic operation in $\mathbb{F}_q$ can be executed in time $\text{poly}(\log q)$ [22].

Proposition 3. *The cost of $\mathbf{R}_{\text{GL}}$ and $\mathbf{U}_{\text{GL}}$ is $\mathcal{O}(n^4 \log q)$ field operations.*

Proof. We show they run in time polynomial in the size of their input, which is dominated by n and $\log q$. The cost of both algorithms is determined by their main loop. Within each iteration $i < n$, the most computationally expensive operation is the call to $\mathbf{R}_{\text{C}}$ or $\mathbf{U}_{\text{C}}$.

The dominant steps in these procedures are: (i) the call to $\mathbf{C}_{\text{B}}$, which computes the RREF of an $n \times (n+i)$ augmented matrix, taking $\mathcal{O}(n^2(n+i))$ field operations; (ii) linear system solving for $\mathbf{R}_{\text{GL}}$, which requires solving an $n \times n$ system of linear equations, taking $\mathcal{O}(n^3)$ field operations; and (iii) matrix-vector multiplication for $\mathbf{U}_{\text{GL}}$, which takes $\mathcal{O}(n^2)$ field operations. Since any arithmetic field operation takes $\log q$ steps, and the loops run for n times, the total cost for both (un)ranking functions is $\mathcal{O}(n^4 \log q)$. $\square$

Remark 3. While the (un)ranking algorithms are in FP (functions that can be computed in polynomial time), the problem of counting invertible matrices is in $\#\text{P}$. However, for $\mathbf{GL}_n(q)$, this counting problem is easy (see Theorem 1). The cost of our algorithms comes from the sequential nature of the ranking process, which is analogous to P-complete problems, as discussed below.

A note on P-completeness. The decision version of the ranking problem for $\mathbf{GL}_n(q)$ (i.e., given M and K , is $\mathbf{R}_{\text{GL}}(M) < K$?) is P-complete. We conjecture that the associated ranking problem is in fact P-complete.

Conjecture 1. The ranking problem for $\mathbf{GL}_n(q)$ is P-complete.

The basis for this conjecture is the inherently sequential nature of the ranking algorithm. The computation of the i -th mixed-radix digit k_i depends on the subspace spanned by the first $i - 1$ columns. This sequential dependency is a hallmark of P-complete problems. The problem is strongly analogous to the canonical P-complete problem of finding the Lexicographically First Maximal Independent Set (LFMIS) in a graph [11]. In LFMIS, a greedy algorithm iterates through vertices in a fixed order, and the decision to include a vertex depends on all previous ones. Similarly, our ranking algorithm iterates through the columns of the matrix, and the rank of each column depends on the previous columns.

A possible reduction could be from the Monotone Circuit Value Problem (MCVP). Given a monotone boolean circuit, one could construct a set of vectors such that the linear independence of a specific vector depends on the output of a gate in the circuit. The lexicographical ordering of the ranking algorithm would correspond to a topological sort of the circuit, and the rank of the final matrix would reveal the output of the circuit. A formal proof of this conjecture would require a detailed construction of this reduction and is left for future work.

4 Practical considerations

To complement the theoretical analysis presented above, we compare our algorithms to several other proposals in the literature, with respect to two metrics: (i) performance, measured in CPU cycles, and (ii) random bit complexity, measured in number of bits. We implement all algorithms using the open source FLINT C library [26], widely used in the literature due to its correctness and performance. We justify our experiment setup, algorithm selection methodology and parameter choices in more detail below.

Implementation details. In arbitrary-precision arithmetic, a wide integer is constructed from multiple *limbs*, each stored as a machine word. We intercept the amount of times `n_randlimb` is called throughout FLINT internals, which we identify to be the lowest level procedure used to generate pseudorandom bits across all primitives provided by the library. A FLINT limb is defined to be 32 or 64 bits (cf. `FLINT_BITS` in the `flint.h` file) [26].

The *fast dice roller* (FDR) uniform sampler [20] is employed to reduce the amount of random bits discarded from samples not aligned to the limb size, via a shared buffer. We let $w \in \{1, \dots, 64\}$ model the minimum size of a random word, or equivalently, the granularity of access to random bits. After each uniform sample produced by FDR, any unused bits are discarded, simulating a PRNG that delivers randomness in fixed w -bit words.

Our implementation of `unrank` departs from the pseudocode in Algorithm 2 in one respect: it does not invoke Algorithm 1 for each column. Instead, we maintain a sequence of available pivot positions (the indices of standard basis vectors not yet consumed) and update it incrementally after each column is placed, by removing the row index of the first non-zero complement coordinate.

Test machine & source code. To reproduce the experiments discussed below, we provide source code and accompanying documentation [28]. All benchmarks are run in an AMD Ryzen™ 7 9700X @ 3.8GHz in a GNU/Linux environment (kernel 6.18.3), gcc 15.2.1 and FLINT 3.4.0. Executables are compiled with the `-O3 -march=native -mtune=native` set of optimization flags. We disable any simultaneous multithreading and dynamic frequency scaling technologies, and set the governor to `performance`. We use the `cpuset` kernel feature to isolate a CPU core to exclusively run the desired benchmarks, without intervention from the scheduler, and disable address layout space randomization (ASLR) to reduce attempts from the operating system to cache certain memory pages.

Algorithm selection & parameter sets. We compare our unranking algorithm (“`unrank`”) with other methods for generating random invertible matrices, as discussed in Section 1: (i) `bruhat`; (ii) `nonsing`; (iii) `rej`; (iv) `subgroup`; and (v) `schubert`. We note that this selection is not exhaustive, as we understand it to be outside the scope of this work. However, we argue that it comprises prominent and algorithmically distinct methods, chosen to be representative of the primary techniques in the domain.

We select n and q employed as parameters of several cryptographic protocols discussed in Section 1. We choose (i) $n \in \{2, \dots, 5\}$ and $q = 16$ [27, Section 1.8]; (ii) $n \in \{9, \dots, 11\}$ and $q \in \{65521, 131071, 524287\}$ [25, Table 5]; and (iii) $n \in \{14, 22, 30\}$ and $q \in \{2039, 4093\}$ [3, Table 2]. We remark that, as cryptanalysis efforts improve, the security of some (or all) selected protocols may be affected, resulting in parameter changes or a total break of the algorithm. Nonetheless, we perform our analysis taking into account different sizes of n and q still used in practice.

Asymptotic time complexity. Most selected algorithms exhibit an asymptotic time complexity of $\mathcal{O}(n^\omega)$ field operations, where ω is the matrix multiplication exponent, as their computational cost is dominated by this operation. We note that (i) **rej** requires determinant computation; (ii) **bruhat**, (iii) **schubert**, and (iv) **nonsing** require structured matrix multiplication. In contrast, **subgroup** relies on a recursive structure that performs a dense matrix multiplication at each of the n steps. This dependency prevents the use of block-recursive strategies required for fast matrix multiplication, resulting in $\mathcal{O}(n^{\omega+1})$ field operations. Our implementation of **unrank** also achieves $\mathcal{O}(n^\omega)$ by avoiding the repeated basis completion step.

Table 1. Amount of CPU cycles ($\times 10^3$) used by sampling algorithms from $\text{GL}_n(q)$. Measurements are the truncated average of 2^{10} executions with random seeds.

n	q	bruhat	nonsing	rej	schubert	subgroup	unrank
2	16	6.14	6.53	10.73	3.35	8.77	4.23
3	16	15.47	13.01	24.30	8.99	25.73	9.85
4	16	29.84	25.01	43.97	18.23	56.15	18.45
5	16	51.42	45.97	77.88	32.24	106.23	33.08
9	524287	232.17	211.30	199.67	190.53	421.53	236.25
10	131071	266.73	251.34	244.00	220.57	525.90	274.74
11	65521	312.52	306.24	291.01	257.70	658.65	331.91
14	4093	444.32	487.75	451.41	358.33	1162.84	476.67
22	4093	1093.83	1487.95	1247.33	916.51	3990.94	1752.49
30	2039	1952.18	3716.43	2526.18	1661.09	10396.00	4116.35

We show the practical efficiency of the evaluated algorithms in Table 1. The **nonsing** method is moderately efficient for small parameter sets, but this behavior quickly vanishes for larger ones, where the **bruhat** method is faster. The **schubert** method is consistently the fastest across all parameter sets, while **subgroup** is the slowest. We observe that **unrank** is the second fastest for small finite fields ($q = 16$), trailing **schubert** by 1.01 – $1.26\times$, and is more than twice as fast as **rej**. For larger fields, the overhead of the underlying linear algebra operations becomes more prominent. Nonetheless, **unrank** is only 1.06 – $1.63\times$ slower than **rej**, and up to $2.48\times$ slower than **schubert** in the worst case ($q = 2039$).

Random bit complexity. By inspection of the algorithms as described by their original pseudocode, and assuming that sampling uniformly from $[0, m)$ costs exactly $\lceil \lg(m) \rceil$ bits, we have that

- (i) **rej** consumes $n^2 \lceil \lg(q) \rceil$ bits per trial, requiring $\frac{1}{\prod_{i=1}^n (1-q^{-i})}$ trials on average;
- (ii) **bruhat** consumes $\sum_{i=2}^n \lceil \lg([i]_q) \rceil + n(n-1) \lceil \lg(q) \rceil + n \lceil \lg(q-1) \rceil$ bits, where $[i]_q := \frac{q^i-1}{q-1}$;
- (iii) **nonsing** consumes $\sum_{i=1}^n [\lceil \lg(q^i-1) \rceil + (i-1) \lceil \lg(q) \rceil]$ bits; and
- (iv) **subgroup** consumes the same as **nonsing** plus $n \lceil \lg(q-1) \rceil$ bits.

The **schubert** method produces a bijection to $\text{GL}_n(q)$. Therefore, this method and **unrank** are asymptotically optimal, consuming $\lceil \lg(\text{ord}(\text{GL}_n(q))) \rceil$ bits.

Table 2. Random bit complexity (in limbs) of sampling algorithms from $\text{GL}_n(q)$, for several word sizes w (in bits). Measurements are the average of 2^{10} executions with random seeds.

n	q	w	bruhat	nonsing	rej	schubert	subgroup	unrank
5	16	1	3.00	2.00	2.13	3.00	3.00	2.00
		4	4.92	3.00	4.18	4.00	3.02	2.00
		16	8.05	5.00	7.52	6.98	6.18	2.00
		64	29.00	15.00	26.86	23.75	19.00	2.00
11	65521	1	46.04	32.00	33.00	44.00	34.03	31.00
		4	52.00	35.00	38.00	49.00	38.00	31.00
		16	76.00	46.00	61.00	70.01	51.00	31.00
		64	139.00	76.00	121.00	128.00	86.00	31.00
30	2039	1	245.08	162.85	169.81	239.39	168.07	155.00
		4	285.29	183.12	208.94	278.31	190.02	155.00
		16	441.60	260.06	360.74	429.58	271.31	155.00
		64	989.21	530.25	900.00	959.23	559.22	155.00

We show practical counts of random bits used by the algorithms in Table 2. We select a subset of the aforementioned parameters to simplify the discussion. Larger values of w reflect implementations of cryptographic primitives where randomness is drawn in discrete chunks. For $w = 64$, most algorithms suffer significant overhead, as repeated sampling steps require different (possibly non-integral) amounts of random bits, and no buffering can be employed. For smaller values of w up to the optimal $w = 1$, the random bit complexity shrinks as expected, approaching the theoretical bound. Evidently, **unrank** is essentially independent of w , as it must sample exactly one integer from $[\text{ord}(\text{GL}_n(q))]$.

In summary, while **schubert** provides the best computational performance, **unrank** is a relevant alternative, providing a reasonable trade-off for a deterministic algorithm with optimal random bit complexity. It can be used as a replacement to **rej** while maintaining similar technical requirements, such as access to linear algebra operations.

5 Conclusion

We have presented the first combinatorial (un)ranking algorithms for the general linear group $\text{GL}_n(q)$, enabling deterministic and uniform random generation of invertible matrices. Along the way, we presented complementary algorithms for (un)ranking vectors outside of a pre-determined vector subspace. We demonstrated the correctness of our algorithms and provided a detailed cost analysis measured in field arithmetic operations. Due to the framework employed, our algorithms are optimal with respect to random bit complexity, at the cost of a small computational overhead compared to other sampling strategies. Nonetheless, we argue that our novel algorithms provide an important connection between algebraic and combinatorial approaches to structured matrix generation.

Future work. We expect that our implementations can be further improved with micro-optimizations, particularly by using faster matrix multiplication algorithms to solve the underlying linear algebra systems; the implementations can also be made resistant to side-channel attacks. Furthermore, we suggest extensions to this novel combinatorial (un)ranking framework for matrix groups:

- (i) the development of (un)ranking algorithms for the special linear group $\text{SL}_n(q)$, the orthogonal group $\text{O}_n(q)$ and other classical groups; and
- (ii) (un)ranking of $\text{GL}_n(q)$ with respect to other orders. In particular, we expect that the combinatorial enumeration algorithm in Gray order of [12] can be extended to novel (un)ranking procedures.

Furthermore, while we presented ranking algorithms for completeness, use cases that enable the compression of invertible matrices (particularly for transmission in cryptographic protocols) may be further explored.

Acknowledgments. We are grateful to the anonymous reviewers for their helpful comments and suggestions. G. Zamboni was partially supported by the Fundacao de Amparo a Pesquisa e Inovacao do Estado de Santa Catarina (FAPESC), Edital 62/2024, and by the National Operator of the Civil Registry of Natural Persons (ON-RCPN) of Brazil. L. G. Rosa was partially supported by the Fundacao de Amparo a Pesquisa e Inovacao do Estado de Santa Catarina (FAPESC), Edital 61/2024. D. Panario was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), reference number RGPIN-2024-05341.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Axler, S.: Linear Algebra Done Right. Springer, 4th edn. (2023)
2. Bellare, M., Ristenpart, T., Rogaway, P., Stegers, T.: Format-Preserving Encryption. In: Jacobson, M.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 5867, pp. 295–312 (Aug 2009). https://doi.org/10.1007/978-3-642-05445-7_19

3. Chou, T., Niederhagen, R., Persichetti, E., Ran, L., Randrianarisoa, T.H., Reijnders, K., Samardjiska, S., Trimoska, M.: Matrix Equivalence Digital Signature. Round 1 Submission, NIST Post-Quantum Cryptography: Additional Digital Signature Schemes Standardization Process (Jul 2023), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/MEDS-spec-web.pdf>
4. Chou, T., Niederhagen, R., Persichetti, E., Randrianarisoa, T.H., Reijnders, K., Samardjiska, S., Trimoska, M.: Take Your MEDS: Digital Signatures from Matrix Code Equivalence. In: Mrabet, N.E., Feo, L.D., Duquesne, S. (eds.) Progress in Cryptology – AFRICACRYPT 2023. Lecture Notes in Computer Science, vol. 14064, pp. 28–52 (Jul 2023). https://doi.org/10.1007/978-3-031-37679-5_2
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition. The MIT Press, 4th edn. (2022)
6. Diaconis, P., Shahshahani, M.: The Subgroup Algorithm for Generating Uniform Random Variables. Probability in the Engineering and Informational Sciences **1**(1), 15–32 (Jan 1987). <https://doi.org/10.1017/S0269964800000255>
7. Dummit, D.S., Foote, R.M.: Abstract Algebra. Wiley, 3rd edn. (2003)
8. Feldtckeller, J., Knichel, D., Sasdrich, P., Moradi, A., Güneysu, T.: Randomness Optimization for Gadget Compositions in Higher-Order Masking. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 188–227 (Aug 2022). <https://doi.org/10.46586/tches.v2022.i4.188-227>
9. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, 3rd edn. (2013)
10. Gigerl, B., Klug, F., Mangard, S., Mendel, F., Primas, R.: Smooth Passage with the Guards: Second-Order Hardware Masking of the AES with Low Randomness and Low Latency. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(1), 309–335 (Dec 2023). <https://doi.org/10.46586/tches.v2024.i1.309-335>
11. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: Limits to Parallel Computation: P-Completeness Theory. Oxford University Press, 1st edn. (1995)
12. Gregor, P., Hoang, H.P., Merino, A., Mička, O.: Generating All Invertible Matrices by Row Operations. In: Mestre, J., Wirth, A. (eds.) 35th International Symposium on Algorithms and Computation (ISAAC 2024). Leibniz International Proceedings in Informatics (LIPIcs), vol. 322, pp. 35:1–35:14 (Dec 2024). <https://doi.org/10.4230/LIPIcs.ISAAC.2024.35>
13. Guo, Q., Hlauschek, C., Johansson, T., Lahr, N., Nilsson, A., Schröder, R.L.: Don’t Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(3), 223–263 (Jun 2022). <https://doi.org/10.46586/tches.v2022.i3.223-263>
14. Holt, D.F., Eick, B., O’Brien, E.A.: Handbook of Computational Group Theory. Chapman and Hall/CRC, 1st edn. (2005)
15. Horn, M.: Speed up `RandomInvertibleMat` and `Random(GL(n,q))` (Sep 2025), <https://web.archive.org/web/20260209181435/https://github.com/gap-system/gap/pull/6088>, accessed on: 2026-02-09
16. Knuth, D.E.: The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley, 3rd edn. (1997)
17. Kreher, D.L., Stinson, D.R.: Combinatorial Algorithms: Generation, Enumeration, and Search. CRC Press, 1st edn. (1998)

18. Kuchta, V., LeGrow, J.T., Persichetti, E.: Post-Quantum Blind Signatures from Matrix Code Equivalence. Cryptology ePrint Archive, Paper 2025/274 (Mar 2025), <https://eprint.iacr.org/2025/274>
19. Land, G., Marotzke, A., Richter-Brockmann, J., Güneysu, T.: Gadget-based Masking of Streamlined NTRU Prime Decapsulation in Hardware. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(1), 1–26 (Dec 2023). <https://doi.org/10.46586/tches.v2024.i1.1-26>
20. Lumbroso, J.: Optimal Discrete Uniform Generation from Coin Flips, and Applications. arXiv:1304.1916v1 (Apr 2013). <https://doi.org/10.48550/arXiv.1304.1916>
21. Morrison, K.: Integer Sequences and Matrices Over Finite Fields. Journal of Integer Sequences **9** (Apr 2006), <https://cs.uwaterloo.ca/journals/JIS/VOL9/Morrison/morrison37.html>, article 06.2.1
22. Mullen, G.L., Panario, D.: Handbook of Finite Fields. Chapman and Hall/CRC, 1st edn. (2013)
23. Pak, I.: When and how n choose k . In: Pardalos, P.M., Rajasekaran, S., Rolim, J. (eds.) Randomization Methods in Algorithm Design. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 43, pp. 191–238 (Dec 1997). <https://doi.org/10.1090/dimacs/043/12>
24. Randall, D.: Efficient generation of random nonsingular matrices. Random Structures & Algorithms **4**(1), 111–118 (1993). <https://doi.org/10.1002/rsa.3240040108>
25. Tang, G., Duong, D.H., Joux, A., Plantard, T., Qiao, Y., Susilo, W.: Practical Post-Quantum Signature Schemes from Isomorphism Problems of Trilinear Forms. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022. Lecture Notes in Computer Science, vol. 13277, pp. 582–612 (May 2022). https://doi.org/10.1007/978-3-031-07082-2_21
26. The FLINT team: FLINT: Fast Library for Number Theory (Nov 2025), version 3.4.0, <https://flintlib.org>
27. Wang, L.C., Chou, C.Y., Ding, J., Kuan, Y.L., Leegwater, J.A., Li, M.S., Tseng, B.S., Tseng, P.E., Wang, C.C.: SNOVA: A New Family of UOV-like Signature Schemes. Round 2 Submission, NIST Post-Quantum Cryptography: Additional Digital Signature Schemes Standardization Process (Jan 2025), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/snova-spec-round2-web.pdf>
28. Zambonin, G.: glg. Version 0.2.0 (Feb 2026). <https://doi.org/10.5281/zenodo.18636578>