New Algorithms and Analyses for Sum-Preserving Encryption

Sarah Miracle and Scott Yilek

University of St. Thomas, St. Paul, USA
{sarah.miracle, syilek}@stthomas.edu

Abstract. We continue the study of sum-preserving encryption schemes, in which the plaintext and ciphertext are both integer vectors with the same sum. Such encryption schemes were recently constructed and analyzed by Tajik, Gunasekaran, Dutta, Ellia, Bobba, Rosulek, Wright, and Feng (NDSS 2019) in the context of image encryption. Our first main result is to prove a mixing-time bound for the construction given by Tajik et al. using path coupling. We then provide new sum-preserving encryption schemes by describing two practical ways to rank and unrank the values involved in sum-preserving encryption, which can then be combined with the rank-encipher-unrank technique from format-preserving encryption. Finally, we compare the efficiency of the Tajik et al. construction and our new ranking constructions based on performance tests we conducted on prototype implementations.

Keywords: Sum-preserving encryption · Image encryption · Format-preserving encryption

1 Introduction

A sum-preserving encryption scheme, recently studied by Tajik, Gunasekaran, Dutta, Ellis, Bobba, Rosulek, Wright, and Feng [18] in the context of image encryption, is a symmetric encryption scheme with an encryption algorithm that takes as its plaintext input a vector of integers, and outputs as a ciphertext another vector of integers with the same sum as the plaintext vector. For applications, the vector components of both the plaintext and ciphertext will typically be integers from 0 up to d, where d is called the component bound.

Tajik et al. introduced definitions and provided a practical construction of sum-preserving encryption in order to build a separate primitive called thumbnail-preserving encryption [12, 20], which is a type of image encryption in which a much smaller version, called a thumbnail, of an encrypted image matches the thumbnail of the unencrypted image. The key idea is that, since sum-preserving encryption turns the plaintext vector of integers into a ciphertext vector with the same *length* and the same *sum*, then the mean will also be preserved. Creating a thumbnail of an image involves replacing a $b \times b$ block of pixels with the average pixel value in the block, so it follows that applying sum-preserving encryption

to each block will result in a ciphertext image with the same thumbnail as the original image.

Sum-preserving encryption can be viewed as a special type of format-preserving encryption (FPE). Format-preserving encryption schemes were originally studied by Brightwell and Smith [4] and were eventually formally defined and analyzed by Bellare, Ristenpart, Rogaway, and Stegers [2]. They have since been widely studied, have found numerous applications, and have even been standardized [6, 10]. Looking forward, one of the main techniques for constructing FPE schemes, the rank-encipher-unrank construction analyzed in [2], will be an important tool for constructing sum-preserving encryption schemes.

While Tajik et al. focused on the use of sum-preserving encryption in the context of images, the fact that the primitive allows one to encrypt a vector of data while maintaining a common statistical measure like mean opens up the possibility of numerous applications for more general dataset encryption. For example, suppose an instructor would like to encrypt her final exam scores each semester before archiving them (which might be especially important given student privacy laws like FERPA in the United States), yet she would still like to be able to go back and compute exam averages to compare different course sections or compare across semesters. Sum-preserving encryption seems to fit the instructor's requirements perfectly.

At the same time, as we will discuss later when introducing security notions, there is the possibility that the sum or mean themselves leak important information about the plaintexts. To see this, consider the extreme example in which the same instructor gave a particularly easy final exam one semester and every student got a score of 100 out of 100. Encrypting a vector of all 100s in a sum-preserving way will just result in another vector of all 100s, so every student's exam score would be revealed by the ciphertext! Nevertheless, while such extreme examples are concerning and are somewhat reminiscent of some of the security issues that arise with other property-preserving encryption schemes (e.g., order-preserving encryption [3]), we emphasize that sum-preserving encryption has already found practical application in the context of image encryption, and can likely be used safely in a number of other application settings, some of which we discuss later in the paper.

1.1 Previous construction

In order to build thumbnail encryption schemes, Tajik et al. set their sights on building a sum-preserving encryption scheme on vectors with values 0-255. Suppose $(m_1, \ldots, m_n)$ with sum S is such a vector. With the obvious connection to format-preserving encryption, they first explore using rank-encipher-unrank, one of the common techniques for building FPE schemes, for sum-preserving encryption. With rank-encipher-unrank, plaintexts are first ranked, meaning mapped to the set of integers $\{0, \ldots, N-1\}$, where N is the number of possible plaintexts. Then a cipher with integer domain is applied to get another integer in this same range, before finally an unrank algorithm maps that integer back into the original plaintext domain. Tajik et al. observe that it should be technically

possible to apply rank-encipher-unrank to sum-preserving encryption by first representing the plaintext vector using the stars-and-bars representation from combinatorics to get a vector $1^{m_1}01^{m_2}0\ldots01^{m_n}$, where each component of the plaintext vector is given in unary with 0s as separators. Such binary strings, with S total 1s (since S is the sum we wish to preserve), are a regular language, and thus known techniques for ranking deterministic finite automata (DFAs) could be applied [2]. Unfortunately, Tajik et al. point out such a method would have very high time complexity and would be impractical for applications.

Tajik et al. then show that, while ranking vectors of length n with a particular sum in general seems hard, it is actually possible to rank vectors of length 2 with a particular sum simply and efficiently. Given this ability to rank (and thus rank-encipher-unrank) vectors of length 2, they go on to give a method to encrypt longer vectors. The idea is to proceed in rounds and, in each round, match up adjacent vector components and apply rank-encipher-unrank to each pair. Since each pair is enciphered in a sum-preserving way, the entire vector will also maintain its sum. Then the entire vector is shuffled before the next round, effectively randomizing the points that will be matched up in the next round. Because each round matches up points and then applies rank-encipher-unrank to each matched pair, we refer to this algorithm as the matching-based construction for the rest of the paper.

Tajik et al. observe their algorithm can be modeled as a Markov chain and the number of necessary rounds to achieve security is then tied to the mixing time of this chain. They discuss how the mixing time would relate to the eigenvalues of the Markov chain transition matrix but, because such matrices are so large in practice, were unable to explicitly compute these values. They go on to give some heuristic arguments for what secure round choices might be, and ultimately test performance with 1000, 3000, and 5000 rounds, but their paper does not provide a mixing time proof for the matching-based construction.

1.2 Our Results

We continue the study of sum-preserving encryption and produce two main results. First, we provide the first mixing-time proof of the matching-based construction of Tajik et al., using a path-coupling technique due to Dyer and Greenhill [7]. Second, we show it actually is possible (and practical) to use rank-encipher-unrank to build sum-preserving encryption for vectors of length n by giving new algorithms for directly ranking and unranking such vectors. Further, we create prototype implementations of both the matching-based construction of Tajik et al. and our new ranking constructions, and show that our ranking constructions have significant performance benefits in a number of applications, including the thumbnail encryption application that was the original motivation for sum-preserving encryption. We now discuss each of these contributions in more detail.

Mixing-time proof of the Tajik et al. construction. Our first contribution is to formally analyze the matching-based construction given by Tajik et

al. [18] and give a bound on the mixing time. We begin by framing their algorithm as a Markov chain $\mathcal{M}_S$ with state space the set of all vectors of length n with component bound d and sum S. Next we prove that the mixing time $\tau_{\mathcal{M}_S}$ of their Markov chain $\mathcal{M}_S$ satisfies $\tau_{\mathcal{M}_S}(\epsilon) \leq n \ln(\min(dn, 2S)\epsilon^{-1})$.

Our proof uses a path coupling technique due to Dyer and Greenhill [7]. To apply path coupling, we carefully select a custom distance metric and design a coupling. Path coupling allows us to only consider a subset of pairs of configurations namely those that differ on only two points and show that using our coupling the expected distance between two configurations will decrease after a single step of $\mathcal{M}_S$. If the shuffling selected by $\mathcal{M}_S$ pairs the two points that differ together the distance decreases to zero. However if these points are not paired together the situation is much more complex and requires a detailed coupling and careful analysis. Much of the complexity comes because the two chains will often have a different number of possible valid next configurations (since the pairs of points have different sums) and thus each individual configuration is selected with a different probability in each chain. The complete proof is given in Section 3.

New algorithms for ranking/unranking sum-preserving vectors. Second, we give algorithms for directly ranking and unranking sum-preserving vectors based on two different total orders. The first is the standard lexicographical order. Stein [17] previously described an algorithm for unranking such vectors using lexicographical order, for use in random sampling. Their algorithm relies on pre-computing a table C_d where position (n, S) stores the number of vectors of length n with sum S and component bound d. We improve on this algorithm by using a cumulative sum table where each position (i, j) stores $\sum_{k=i}^{n} \mathsf{C}_d(k, j)$. Additionally we give a dynamic programming based justification for the computations required to fill the table (Section 4.3). Finally, in Section 4.1 we give a ranking algorithm which also uses the cumulative sum table.

In order to handle applications with larger parameters d and n, we develop a second set of rank and unrank algorithms based on a new total ordering we call recursive block order. While recursive block order uses ideas that are reminiscent to those used in orderings of monomials, specifically block order (see e.g., [8]) and graded order (see e.g. [5]), these are combined differently and applied recursively unlike in monomial orderings. At a high-level, in recursive block order configurations are first ordered based on the sum of the first $n/2$ points. Configurations with smaller "first-half" sums have lower rank. Configurations with the same "first-half" sum are then ordered recursively according to the first $n/2$ points or, if these are identical, the last $n/2$ points. We formally define this order, give both rank and unrank algorithms, and go over an example in Sections 4.2 and 4.5.

One of the main advantages of recursive block order is that it requires fewer rows of the C table to be computed and stored. Specifically it only requires at most $2 \log n$ rows (and only $\log n$ if n is a power of 2). However, the dynamic programming approach to filling the C table does not allow us to take advantage of this and requires all n rows be computed. To address this, we present an

alternative way to compute only the values needed from the C table using well-known formulas derived from generating functions (see e.g. [1], [15]).

Performance comparison. We created prototype implementations of both the Tajik et al. matching-based construction and our two ranking-base constructions and ran a number of performance tests for a variety of applications with a wide range of parameter choices for the vector length n and component bound d. In short, we found that for a number of applications, including the thumbnail encryption application that originally motivated sum-preserving encryption, our new ranking-based constructions are significantly more efficient, even when the matching-based construction is used with round numbers well below the bounds we prove in Section 3. The matching-based construction appears to be the superior choice when n is small but d is large (e.g., $n = 30$ and $d = 100000$). When n is very large, in the thousands or higher, then neither the matching-based construction nor our ranking constructions perform well, and new approaches are likely needed. We discuss these and other details of our performance analysis in Section 5.

2 Background on Sum-Preserving Encryption

In this section we define what a sum-preserving encryption scheme is, discuss security goals, describe some example applications, and give details on previous constructions. We note that much of this section closely follows the work of Tajik et al. [18].

2.1 Syntax

We now give a formal definition of a sum-preserving encryption scheme. Let $n \geq 1$ be an integer called the *vector length*, and $d > 0$ be an integer called the *component bound*. A d-bounded vector of length n is a vector of integers $(x_1, \ldots, x_n)$ with $0 \leq x_i \leq d$. We denote by $(\mathbb{Z}_{d+1})^n$ the set of all d-bounded vectors of length n. A sum-preserving encryption scheme for $(\mathbb{Z}_{d+1})^n$ is a pair of algorithms $(\mathsf{Enc}, \mathsf{Dec})$ with the following properties.

- The (deterministic) encryption algorithm $\mathsf{Enc} : \mathcal{K} \times \{0,1\}^* \times (\mathbb{Z}_{d+1})^n \to (\mathbb{Z}_{d+1})^n$ takes as input a key $K \in \mathcal{K}$, a nonce $T \in \{0,1\}^*$, and message $M \in (\mathbb{Z}_{d+1})^n$, and outputs a ciphertext $C \in (\mathbb{Z}_{d+1})^n$ which is also a d-bounded vector of length n. Importantly, the encryption algorithm must be *sum-preserving*, which means for all keys $K \in \mathcal{K}$, all nonces $T \in \{0,1\}^*$, and all messages $M \in (\mathbb{Z}_{d+1})^n$, it is true that $\sum M = \sum \mathsf{Enc}(K, T, M)$, meaning the sum of the message vector components $\sum M = \sum_{i=1}^{n} m_i$ must be equal to the sum of the ciphertext components $\sum \mathsf{Enc}(K, T, M) = \sum_{i=1}^{n} c_i$, where $\mathsf{Enc}(K, T, M) = (c_1, \ldots, c_n)$.
- The decryption algorithm $\mathsf{Dec} : \mathcal{K} \times \{0,1\}^* \times (\mathbb{Z}_{d+1})^n \to (\mathbb{Z}_{d+1})^n$ takes as input a key $K \in \mathcal{K}$, a nonce $T \in \{0,1\}^*$, and ciphertext $C \in (\mathbb{Z}_{d+1})^n$ and outputs a message vector $M \in (\mathbb{Z}_{d+1})^n$.

For correctness we require that for all $K \in \mathcal{K}$, all $T \in \{0,1\}^*$, and all $M \in (\mathbb{Z}_{d+1})^n$, it must be the case that $\mathsf{Dec}(K, T, \mathsf{Enc}(K, T, M)) = M$.

Connection to bounded integer compositions. We note that d-bounded vectors of length n with the sum of vector components equal to an integer S are also called d-bounded (or restricted) n-part compositions of S or d-bounded n-compositions of S. There has been much previous work studying restricted compositions. Much of the previous work has surrounded the problems of enumerating all compositions and counting the number of such compositions (see e.g., [1], [13], [14], [15], [19]). We will use the terminology bounded n-composition throughout the paper.

2.2 Examples

To help better understand sum-preserving encryption, we now discuss some example applications where it has either already been used or could potentially be used. We will also revisit these same examples in Section 5 when we evaluate our prototype implementations.

Example 1: Thumbnail-preserving image encryption. Tajik et al. previously used sum-preserving encryption to build a type of image encryption called Thumbnail-preserving encryption, in which the thumbnail of an encrypted image exactly matches the thumbnail of the unencrypted image. We can view image data as a matrix of pixel values 0-255 with dimensions $h \times w \times 3$, where each of the three $h \times w$ matrices represents an RGB channel. Tajik et al. showed one can do thumbnail-preserving encryption by taking $m \times m$ blocks and encrypting them in a sum-preserving way. Then, when forming a thumbnail by replacing each $m \times m$ block with a single pixel that is its average, the sum-preserving property of encryption ensures the encrypted blocks have the same average as the unencrypted image blocks. Thus, to use this construction we need sum-preserving encryption for vectors of length $n = m \cdot m$, the total number of pixels in a block, and component bound $d = 255$, the maximum value of a pixel.

Example 2: Exam scores. Suppose an instructor has a class with 300 students, and vector of final exam scores which are each integers 0-100. The instructor may want to encrypt this vector of scores in a way that the exam average can be calculated from the encrypted vector alone. In this case we can use sum-preserving encryption with vector length $n = 300$ and component bound $d = 100$.

Example 3: Employee Salaries. A small company with 30 employees wants to encrypt a vector of employee salaries, which are integers between 30,000 and 100,000, in a way such that the average salary can be computed from the encrypted salary vector. In this case we potentially have a few options for using sum-preserving encryption. We could set $n = 30$, and $d = 100000$; in this case the encrypted salaries will range from 0 to 100000, so we could get encrypted salaries below the lowest actual salary of 30,000, and the top salary of 100,000

will also potentially be revealed by the vector, since no value will be above that. Due to this latter issue, the encryptor might instead choose to use a larger choice of d. But, as we will discuss later in the implementation section, choosing a larger d in this example could have significant performance consequences.

Example 4: Rating dataset. Suppose a website has 5,000 user ratings, each integers that are 0 to 4 stars. The website might want to encrypt this dataset in a way that allows the average rating to still be computed. Sum-preserving encryption with $n = 5000$ and $d = 4$ could be used.

We note that for some example applications, there may be possible solutions other than using sum-preserving encryption. For example, one could encrypt the data using a standard symmetric encryption scheme, and then simply append the sum or mean of the original data to the ciphertext. However, sum-preserving encryption has the benefit of also maintaining the format of the original message, so it may be the superior solution when it is necessary, or even just more convenient, for ciphertexts to still be d-bounded vectors.

2.3 Security notions

Like previous work on format-preserving encryption and sum-preserving encryption, we aim to build sum-preserving encryption schemes that are indistinguishable from randomly chosen sum-preserving permutations on the same domains. Formally, let $\mathsf{Enc} : \mathcal{K} \times \{0,1\}^* \times (\mathbb{Z}_{d+1})^n \to (\mathbb{Z}_{d+1})^n$ be a sum-preserving encryption algorithm on d-bounded vectors of length n and let $\mathsf{Dec} : \mathcal{K} \times \{0,1\}^* \times (\mathbb{Z}_{d+1})^n \to (\mathbb{Z}_{d+1})^n$ be the corresponding decryption algorithm. To define PRP security, we say the prp-advantage of an adversary A is

$$\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{Enc}}(A) = \Pr\left[A^{\mathsf{Enc}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[A^{\pi(\cdot,\cdot)} \Rightarrow 1\right].$$

The adversary is given access to either an encryption oracle that takes as input a nonce and a message vector, or a randomly chosen family of permutations $\pi : \{0,1\}^* \times (\mathbb{Z}_{d+1})^n \to (\mathbb{Z}_{d+1})^n$. We can also target a stronger security notion, strong PRP security, if we additionally give the adversary either a decryption oracle or an inverse permutation family π^{-1}. The previous work of Tajik et al. on sum-preserving encryption further restricted the definition above to only consider adversaries that never repeat a nonce input to their oracles. They called such adversaries *nonce respecting* (NR), and argued that this security notion is meaningful for applications. Following Tajik et al., we will primarily focus on security against NR adversaries, but will also discuss how to achieve the stronger notions and the corresponding effects on performance.

Discussion. It is important to note there are inherent limitations in the security we can achieve with sum-preserving encryption, even when achieving the security notions just described. To see this, consider the exam score application described above. If the exam is particularly easy and all 300 students achieve a score of

100/100, then the resulting message vector with 100 repeated 300 times will encrypt to exactly the same vector and end up revealing every student's score. In other applications, this may not be as problematic. For example, in the thumbnail encryption example, a large block of white pixels will all have the maximum pixel value of 255 and thus encrypt to another block of all white pixels. Yet if our goal is to use such image encryption to hide details of the image like facial features, then a large block of pixels staying the same color does not seem so damaging.

2.4 Previous Constructions

After defining sum-preserving encryption, Tajik et al. observe that one option for constructing such schemes would be to use the rank-encipher-unrank construction of [2], which is one of the common ways of building format-preserving encryption schemes. We say that a set $\mathcal{X}$ has an efficient ranking if there is an efficient algorithm $\mathsf{rank} : \mathcal{X} \to \{0, \ldots, |\mathcal{X}| - 1\}$ mapping elements of the set to integers, and then an efficient inverse function unrank mapping integers back into the set $\mathcal{X}$. The idea behind rank-encipher-unrank is, given a point $x \in \mathcal{X}$ to encipher, one can first rank x to get an integer n_x in the range $\{0, \ldots, |\mathcal{X}| - 1\}$. Then one applies a cipher that works on that integer domain to get another integer n_y in the same domain. Finally, applying unrank to n_y yields another point $y \in \mathcal{X}$, which acts as the ciphertext.

Algorithms for ranking based on the DFA representation of a language are known [2], so Tajik et al. observe it is technically possible to use this paradigm for constructing sum-preserving encryption. The key idea is to use the stars-and-bars representation of the message space: the individual elements of the vector to encrypt are represented in unary and the symbol 0 can be used as a separator between these unary sequences, so the vector $(3, 1, 2)$ would be 11101011. It is easy to come up with a regular expression for such binary strings, and from there a DFA to be used in ranking. Unfortunately, Tajik et al. argue this would be far too computationally expensive to be useful.

Tajik et al. then observe that, while ranking vectors of length n with a particular sum in general would be too computationally expensive, it is actually straightforward to rank vectors of length 2. Let $(a, b) \in \{0, \ldots, d\}^2$ have sum $S = a + b$. Then $\mathsf{rank}((a, b)) = a$ if $S \leq d$ and $d + 1 - a$ otherwise, and $\mathsf{unrank}(t) = (t, S - t)$ if $S \leq d$ and $(d + 1 - r, S - d - 1 + r)$ otherwise.

Given a way to rank and unrank vectors of size 2, Tajik et al. then give an efficient construction for encrypting longer vectors while preserving their sum. Their algorithm proceeds in rounds. In each round, match up all of the points in the vector with their neighbor and, for each pair of neighbors, apply rank-encipher-unrank using the ranking and unranking formulas for length-2 vectors just described. As a result, each pair of neighboring points in the vector will be replaced by a new pair with the same sum, and the overall sum of the vector will be maintained. The points of the vector are then randomly shuffled between rounds (e.g., with the Knuth shuffle) so that the next round finds new neighboring points matched up.

3 Analyzing the Matching-based approach of Tajik et al.

In this section we formally analyze the matching-based construction given by Tajik et al. [18] and described in Section 2.4. We begin by framing the algorithm formally as a Markov chain $\mathcal{M}_S$ (Section 3.1) and then use Markov chain analysis techniques to bound the mixing time (Sections 3.2 and 3.3). Finally in Section 3.4 we apply our bound to each example application given in Section 2.2.

We begin by formally defining the mixing time. The time a Markov chain $\mathcal{M}$ takes to converge to its stationary distribution μ is measured in terms of the distance between μ and $\mathcal{P}^t$, the distribution at time t. Let $\mathcal{P}^t(x, y)$ be the t-step transition probability and Ω be the state space. The *mixing time* of $\mathcal{M}$ is

$$\tau_{\mathcal{M}}(\epsilon) = \min\{t : ||\mathcal{P}^{t'} - \mu|| \leq \epsilon, \forall t' \geq t\},$$

where $||\mathcal{P}^t - \mu|| = \max_{x \in \Omega} \frac{1}{2} \sum_{y \in \Omega} |\mathcal{P}^t(x, y) - \mu(y)|$ is the *total variation distance* at time t.

3.1 The Sum-Preserving Markov chain $\mathcal{M}_S$

Let the state space $\Omega_{S,d,n} = \Omega$ be the set of all d-bounded n-compositions of S. Specifically, given a configuration x, each point $x(i)$ for $1 \leq i \leq n$ satisfies $0 \leq x(i) \leq d$ and the sum of the points satisfies $\sum_{i=1}^{n} x(i) = S$. We analyze the following Markov chain which is equivalent to the construction given by Tajik et al. [18].

The Sum-Preserving Shuffle Markov chain $\mathcal{M}_S$

`Starting at any valid configuration` $x_0 \in \Omega$`, iterate the following:`

- At time t, choose a random shuffling R on all points uniformly at random (u.a.r.).
- Pair adjacent points in R to create a perfect matching M.
- Independently, for each matched pair of points $(p_i, p_j) \in M$ select values for $x_{t+1}(p_i)$ and $x_{t+1}(p_j)$ u.a.r. from all valid choices that preserve the sum. Namely, all choices that satisfy $x_t(p_i) + x_t(p_j) = x_{t+1}(p_i) + x_{t+1}(p_j)$, $x_{t+1}(i) \leq d$, and $x_{t+1}(j) \leq d$.

Next we show that this Markov chain is irreducible (i.e. the state space Ω is connected) and thus has a unique stationary distribution (see e.g., [11]) and that the stationary distribution is uniform.

Lemma 1. *The Markov chain $\mathcal{M}_S$ is irreducible and has the uniform distribution on $\Omega_{S,d,n}$ as its unique stationary distribution.*

Proof. We will prove that $\mathcal{M}_S$ is irreducible by defining a distance metric ϕ and showing that there is always a move of $\mathcal{M}_S$ that decreases the distance between any two configurations. By repeatedly decreasing the distance it is thus possible

to create a path between any two valid configurations. Define the distance ϕ between two configurations x and y as follows.

$$\phi(x, y) = \sum_{i=1}^{n} |x(i) - y(i)|. \tag{1}$$

We claim there is always a valid transition of $\mathcal{M}_S$ that will decrease the distance between x and y. Select a point p^+ which is larger in x than in y (i.e. $x(p^+) > y(p^+)$) and a point p^- that is smaller in x than in y. This is always possible since $x \neq y$ and the sum of the points in x is the same as the sum of the points in y. Next, decrease p^+ by 1, increase p^- by 1, and leave other points the same in x. This creates a valid configuration x' such that $\phi(x', y) < \phi(x, y)$ and (x, x') is a valid transition of $\mathcal{M}_S$. To see that this is a valid transition in $\mathcal{M}_S$ select a shuffling where these points are adjacent, it is clear there is a valid selection for each pair of matched points that gives the desired transition.

Next, we will show that for all $x, y \in \Omega, P(x, y) = P(y, x)$ and thus by detailed balance the uniform distribution must be the stationary distribution (see e.g., [11]). Consider any $x, y \in \Omega$. Let $\mathcal{T}_{x,y}$ be the set of all shufflings that allow a transition from x to y (i.e. starting from x if one of these shufflings is selected there is a way to select the values in the second step of the chain to match y.) Note that if $P(x, y) = 0$ then $\mathcal{T}_{x,y} = \emptyset$. It is clear that $\mathcal{T}_{y,x} = \mathcal{T}_{x,y}$ and for each shuffling $t \in \mathcal{T}_{x,y}$ each matched pair has the same sum in x as in y implying that both chains have the same valid choices in the last step of $\mathcal{M}_S$. Thus the probability of moving from x to y if t is selected is the same in both configurations and therefore $P(x, y) = P(y, x)$. □

3.2 Bounding the Mixing Time of $\mathcal{M}_S$

In order to bound the mixing time of $\mathcal{M}_S$ we will use the path coupling method due to Dyer and Greenhill [7] which is an extension of the well-known coupling method (see e.g., [9]). A coupling of Markov chains with transition matrix P is a stochastic process $(X_t, Y_t)_{t=0}^{\infty}$ on $\Omega \times \Omega$ such that X_t and Y_t are both Markov chains with stationary distribution P and if $X_t = Y_t$, then $X_{t+1} = Y_{t+1}$. In other words when viewed in isolation each of the chains X and Y simulate the original chain and once they agree, they will always agree. Informally, the coupling time is the time until the two chains agree. A common technique is to select an appropriate coupling and then use the coupling time to bound the mixing of time of the Markov chain. This is often done by defining a distance metric and showing that in expectation the distance between any two arbitrary pairs of configurations is decreasing. The path coupling technique, which common in the Markov chain community, only requires considering pairs of states that are close according to the selected distance metric. In our case we will again use the Manhattan distance metric and will only need to consider pairs of states that differ on exactly 2 points.

More formally, we will use the following path coupling theorem due to Dyer and Greenhill [7].

Theorem 1. *Let ϕ be an integer valued metric defined on $\Omega \times \Omega$ which takes values in $\{0, ..., B\}$. Let U be a subset of $\Omega \times \Omega$ such that for all $(x_t, y_t) \in \Omega \times \Omega$ there exists a path $x_t = z_0, z_1, ..., z_r = y_t$ between x_t and y_t such that $(z_i, z_{i+1}) \in U$ for $0 \leq i < r$ and $\sum_{i=0}^{r-1} \phi(z_i, z_{i+1}) = \phi(x_t, y_t)$. Let $\mathcal{M}$ be a Markov chain on Ω with transition matrix P. Consider any random function $f : \Omega \to \Omega$ such that $\Pr[f(x) = y] = P(x, y)$ for all $x, y \in \Omega$, and define a coupling of the Markov chain by $(x_t, y_t) \to (x_{t+1}, y_{t+1}) = (f(x_t), f(y_t))$. If there exists $\beta < 1$ such that $E[\phi(x_{t+1}, y_{t+1})] \leq \beta\phi(x_t, y_t)$, for all $(x_t, y_t) \in U$, then the mixing time of $\mathcal{M}$ satisfies*

$$\tau(\epsilon) \leq \frac{\ln(B\epsilon^{-1})}{1-\beta}.$$

To apply the theorem, we will present a coupling and show that the expected distance between any pair of configurations that differ by exactly two points decreases by at least β after each step of the Markov chain (for an appropriately choosen β). Note that here B is the maximum distance between two configurations using the selected distance metric.

Next, we prove the following theorem giving an upper bound on the mixing time of $\mathcal{M}_S$. Note that the mixing time (defined at the beginning of Section 3) bounds the number of steps of $\mathcal{M}_S$ needed and does not include the time to implement each step. We believe our bound can be improved.

Theorem 2. *Let n be the vector length, d the component bound, and S be the fixed sum. Given these definition, the mixing time $\tau_{\mathcal{M}_S}(\epsilon)$ of the sum-preserving Markov chain $\mathcal{M}_S$ on state space $\Omega_{n,d,S}$ satisfies*

$$\tau_{\mathcal{M}_S}(\epsilon) \leq n \ln(\min(dn, 2S)\epsilon^{-1}).$$

Proof. In order to apply Theorem 1 we begin by formally defining ϕ, U, and then bound B and β. As before, we define the distance $\phi(x, y)$ as in Equation 1 as the L_1 norm (the Manhattan distance). Let U be the set of configurations x and y which differ on exactly 2 points. Next, we will show that for all $(x, y) \in \Omega \times \Omega$ there exists a path $x = z_0, z_1, ..., z_r = y$ between x and y such that $(z_i, z_{i+1}) \in U$ for $0 \leq i < r$ and $\sum_{i=0}^{r-1} \phi(z_i, z_{i+1}) = \phi(x, y)$.

Consider the path between any true arbitrary configurations given above in the proof that $\mathcal{M}_S$ is irreducible (Lemma 1). We claim this path satisfies the conditions. Given a configuration z_i to determine the next step in the path z_{i+1} two points p^+ and p^- are chosen where $z_i(p^+) > y(p^+)$ and $z_i(p^-) < y(p^-)$. A shuffling is selected where these points are paired together and these are the only two points that are modified. Namely, $z_{i+1}(p^+) = z_i(p^+)-1$, $z_{i+1}(p^-) = z_i(p^-)-1$, and for all other points p, $z_i(p) = z_{i+1}(p)$. It is thus clear that $(z_i, z_{i+1}) \in U$. It is easily seen that for $0 \leq i < r$, $\phi(z_i, z_{i+1}) = 2$ and $\phi(z_{i-1}, y) = \phi(z_i, y) - 2$ and thus the second distance requirement is satisfied.

To bound the maximum distance B recall that S is the sum over all points $S = \sum_{i=1}^{n} x(i)$. Since each point can contribute at most d to ϕ and there are n points, B satisfies

$$B \leq \min(dn, 2S).$$

In our coupling we will use the same shuffling in both x and y. If the two points that differ get paired together in the shuffling then the valid choices for all paired points are identical in both x and y, our coupling will choose the same configurations in both chains, and the distance will decrease to 0. This happens with probability $1/(n-1) > 1/n$. Otherwise, both of the two points that differ between x and y will get paired with points that are the same in both x and y. For all other pairs that do not include these points, our coupling will make the same choice in both x and y. We will prove in Lemma 2 that no matter what values the 2 points that differ have, there is a way to couple them so that the distance never increases. Thus we have

$$E[\phi(x_{t+1}, y_{t+1})] \leq \left(\frac{1}{n}\right) \cdot 0 + \left(\frac{n-1}{n}\right) \cdot \phi(x_t, y_t) = \left(\frac{n-1}{n}\right) \cdot \phi(x_t, y_t).$$

Letting $\beta = \frac{n-1}{n}$, $B \leq \min(dn, 2S)$, and applying Theorem 1 gives

$$\tau_{\mathcal{M}_S}(\epsilon) \leq n \ln(\min(dn, 2S)\epsilon^{-1}).$$

□

3.3 Proof of Lemma 2 Coupling Two Points

It remains to show we can construct a coupling of any two points with different values where the expected change in distance is zero. We will prove the following lemma.

Lemma 2. *Given two arbitrary points i and j with $x_t(i) = y_t(i)$ and $x_t(j) \neq y_t(j)$, there exists a coupling such that*

$$|x_t(j) - y_t(j)| \geq |x_{t+1}(i) - y_{t+1}(i)| + |x_{t+1}(j) - y_{t+1}(j)|.$$

Proof. Recall that for two points i and j with $x_t(i) + x_t(j) = S_x$, the values $x_{t+1}(i), x_{t+1}(j)$ are chosen uniformly from all possible choices with $x_{t+1}(i) + x_{t+1}(j) = S_x$, $x_{t+1}(i) \leq d$, and $x_{t+1}(j) \leq d$. For example, if $S_x = 4$ and $d = 3$ the options are $(1,3), (2,2), (3,1)$ and each is selected with probability $1/3$. Since $x_t(j) \neq y_t(j)$, the two chains will often have a different number of possible choices (see Fig. 1) and thus each individual configuration is selected with a different probability in each chain.

We begin by creating an ordering of the possible choices for $(x_{t+1}(i), x_{t+1}(j))$ (and similarly for $(y_{t+1}(i), y_{t+1}(j))$) and then show how we will carefully pair the choices to ensure that the distance never increases. This is especially difficult because when the two chains have a different number of possible choices, one configuration in x will needed to be paired with multiple configurations in y (or vice versa). We can view the coupling as creating a weighted bipartite graph (as shown in Fig. 1) where one partition is the possible choices in x and the other is the choices in y. A valid coupling is a set of edges between the partitions where each edges is assigned a probability such that the sum of the edges adjacent to each configuration is equal to the probability of that configuration.

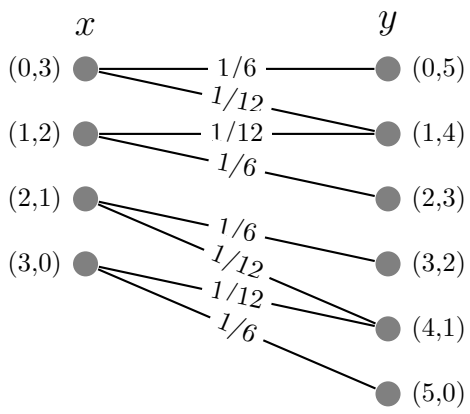

Fig. 1. The weighted bipartite graph visualizing the coupling for $S_x = 3$, $S_y = 5$, and $d = 5$.

We will order the choices by increasing value of $x_{t+1}(i)$ (or $y_{t+1}(i)$) starting from the lowest possible value (i.e. lexicographical order). For example, if $x_t(i)+x_t(j) = 3$ there are 4 choices ordered $(0,3),(1,2),(2,1),(3,0)$ and each has probability $1/4$. With configurations in this order, we will always couple a configuration in x to the lowest possible configuration in y while maintaining the correct probabilities. More specifically, let $S_x = x_t(i)+x_t(j)$ and $S_y = y_t(i)+y_t(j)$. We will begin with the case that $x_t(i) + x_t(j) <= d$, $y_t(i) + y_t(j) <= d$, and $S_x < S_y = \delta + S_x$. We will handle the general case later. In this case, x has $S_x + 1$ possible configurations ordered $(0, S_x), (1, S_x - 1), \ldots, (S_x, 0)$. Similarly, y has $S_y + 1$ possible configurations ordered $(0, S_y), (1, S_y - 1), \ldots, (S_y, 0)$. We start by pairing (adding an edge between) the lowest configurations (i.e.$(0, S_x)$ and $(0, S_y)$) with probability $1/(S_y + 1)$. At this point, configuration $(0, S_x)$ still has probability $1/(S_x + 1) - 1/(S_y + 1)$ remaining. We will then pair it to the next lowest configuration in y (i.e. $(1, S_y - 1)$) with the remaining probability or $1/(S_y + 1)$ whichever is smaller. We will continue pairing $(0, S_x + 1)$ with the lowest configuration in y that has unused probability (i.e. it's adjacent edges do not add to $1/(S_y + 1)$) until the edges assigned add to the correct probability $1/(S_x + 1)$. We then iterate through the remaining configurations in x (in the ordering above) using the same algorithm. Specifically, pairing each with the lowest configuration(s) in y that have remaining probability. For example, the case $x_t(i) = y_t(i) = 2, x_t(j) = 1$, and $y_t(j) = 4$ is shown in Fig. 1.

It remains to show that using the coupling described above the distance will never increase. Specifically we will show for each coupled pair (i.e. configurations connected by an edge), the distance between those configurations is at most $\delta = |x_t(j) - y_t(j)|$. Let $(i, S_x - i)$ for $0 \leq i \leq S_x$ be any valid configuration for x. We want to show that given our coupling the edges leaving this configuration only go to configurations in y at distance at most δ. Specifically we need to show that all edges are to configurations in the range $\{(i, S_y - i), \ldots, (i + \delta, S_y - i - \delta)\}$

(using the ordering defined above). Let $\mathbb{P}_x(i,j)$ be the probability of all x points in the range $\{(0,S_x),\ldots,(i,j)\}$ (and similarly define $\mathbb{P}_y(i,j)$). Recall that our coupling will match a point in x to the lowest point(s) in y that have remaining probability. Thus to show that $(i, S_x - i)$ only gets mapped to points in the appropriate range (i.e. the points in y at distance δ) we need to show that the point in y just before that range $(i-1, S_y - i - 1)$ will have no remaining probability and that $(i, S_x - i)$ will not get matched to points after $(i+\delta, S_y - i - \delta)$. Specifically it suffices to prove the following.

Proposition 1.

1. *For* $0 < i \leq S_x$, $\mathbb{P}_x(i-1, S_x - i - 1) \geq \mathbb{P}_y(i-1, S_y - i - 1)$.
2. *For* $0 \leq i < S_x$, $\mathbb{P}_x(i, S_x - i) \leq \mathbb{P}_y(i+\delta, S_y - i - \delta)$.

Proof. Since there are $S_x + 1$ configurations for x and $S_y + 1 = S_x + \delta + 1$ configurations for y we have the following

$$\mathbb{P}_x(i,j) = \frac{i+1}{S_x+1}, \quad \mathbb{P}_y(i,j) = \frac{i+1}{S_x+\delta+1}.$$

Given these definitions, it is straightforward to prove the proposition using basic algebra. □

It remains to consider the more general case where either $x_t(i) + x_t(j) > d$ or $y_t(i) + y_t(j) > d$ (or both). We will use the same coupling described above where a configuration x is paired with the lowest configuration(s) in y with remaining probability. Similar to what we did in the first case, for each valid configuration for x we will map the configuration to two configurations in y. We will prove that these two configurations and all configurations between them are at distance at most δ. Then we will show that the x configuration will never get coupled to a y configuration outside of this range.

Let n_x be the number of valid x configurations and n_y be the number of valid y configurations. Without loss of generality, we will assume that $n_x \leq n_y$. Let (x_1, x_2) be the lowest configuration for x and (y_1, y_2) be the lowest configuration for y. As before, if $S_x \leq d$ then $x_1 = 0, x_2 = S_x$ and there are $x_2 - x_1 + 1 = S_x + 1$ configurations. However if $S_x > d$ then $x_1 = d - n_x, x_2 = d$ and there are $x_2 - x_1 + 1 = 2d - S_x + 1$ configurations $(S_x - d, d), (d - S_x + 1, d - 1), \ldots, (d, S_x - d)$. We begin by proving that these initial configurations are at distance $\delta = |S_x - S_y|$.

Lemma 3. *Let* $\delta = |S_x - S_y|$ *be the initial distance between* x *and* y*. Assuming* $n_x \leq n_y$ *and the possible configurations are ordered as described above we have.*

1. *The lowest configurations* (x_1, x_2) *and* (y_1, y_2) *as defined above satisfy.*

$$\phi((x_1, x_2), (y_1, y_2)) \leq \delta.$$

2. *The highest configurations* (x_2, x_1) *and* (y_2, y_1) *as defined above satisfy.*

$$\phi((x_2, x_1), (y_2, y_1)) \leq \delta.$$

3. *For* $0 < c \leq x_2 - x_1$, $\phi((x_1 + c, x_2 - c), (y_1 + c, y_2 - c)) \leq \delta$.
4. *For* $0 < c \leq x_2 - x_1$, $\phi((x_2 - c, x_1 + c), (y_2 - c, y_1 + c)) \leq \delta$.

Proof. Here we will consider four cases based on how S_x and S_y compare to d. If both $S_x, S_y \leq d$ then $(x_1, x_2) = (0, S_x)$ and $(y_1, y_2) = (0, S_y)$. Here we have

$$|x_1 - y_1| + |x_2 - y_2| = |0 - 0| + |S_x - S_y| = \delta.$$

Next, we will consider the case where exactly one of S_x, S_y is greater than d. If $S_x > d$ and $S_y \leq d$. In this case $(x_1, x_2) = (d - S_x, d)$ and $(y_1, y_2) = (0, S_y)$. Here we have

$$|x_1 - y_1| + |x_2 - y_2| = |S_x - d| + |d - S_y| = S_x - S_y = \delta.$$

If instead $S_y > d$ and $S_x \leq d$ the argument is identical. Finally if both $S_x, S_y > d$ we have $(x_1, x_2) = (d - S_x, d)$ and $(y_1, y_2) = (d - S_y, d)$. Here we have

$$|x_1 - y_1| + |x_2 - y_2| = |(S_x - d) - (S_y - d)| + |d - d| = |S_x - S_y| = \delta.$$

It immediately follows that the final configurations (x_2, x_1) and (y_2, y_1) are also at distance at most δ. Similarly it can easily be shown by the definition of the L_1 distance metric that third and fourth statements are true. □

To begin, if $n_x = n_y$ then we will match each point $(x_1 + c, x_2 - c)$ with exactly one point $(y_1 + c, y_2 - c)$ with weight $1/n_x = 1/n_y$. By Lemma 3 these points are distance at most δ and we are done.

Next, assume $n_x < n_y$ and consider any general configuration for x, $(x_1 + c, x_2 - c)$. We will show that this configuration will only be matched with configurations in between (and including) $(y_1 + c, y_2 - c)$ and $(y_2 - (x_2 - x_1 - c), y_1 + (x_2 - x_1 - c))$. By Lemma 3 parts 3 and 4, these configurations are both at distance at most δ from $(x_1 + c, x_2 - c)$. It also immediate follows that any points between these are at distance at most d from $(x_1 + c, x_2 - c)$.

Next we show that $(x_1 + c, x_2 - c)$ will only be coupled to points in y between and including $(y_1 + c, y_2 - c)$ and $(y_2 - (x_2 - x_1 - c), y_1 + (x_2 - x_1 - c))$. Again as in the first case to do this we need to show that $(x_1 + c, x_2 - c)$ will never be matched with anything below $(y_1 + c, y_2 - c)$ or anything above $(y_2 - (x_2 - x_1 - c), y_1 + (x_2 - x_1 - c))$. Specifically we prove the following.

1. For $0 < c \leq x_2 - x_1$, $\mathbb{P}_x(x_1 + c - 1, x_2 - c + 1) > \mathbb{P}_y(y_1 + c - 1, y_2 - c + 1)$.
2. For $0 \leq c < x_2 - x_1$, $\mathbb{P}_x(x_1 + c, x_2 - c) \leq \mathbb{P}_y(y_2 - (x_2 - x_1 - c), y_1 + (x_2 - x_1 - c))$.

Recall that $\mathbb{P}_x(x_1 + c - 1, x_2 - c + 1)$ is the probability of all configurations for x up to and including $(x_1 + c - 1, x_2 - c + 1)$. This includes c configurations each with probability $1/n_x$ and thus $\mathbb{P}_x(x_1 + c - 1, x_2 - c + 1) = c/n_x$. Similarly $\mathbb{P}_y(y_1 + c - 1, y_2 - c + 1) = c/n_y$. Thus our first statement follows directly from the fact that $n_x < n_y$.

Using the fact that $n_x = x_2 - x_1 + 1$ and $n_y = y_2 - y_1 + 1$ we can prove the second statement as follows.

$$\begin{aligned}
c &\leq x_2 - x_1 \\
c+1 &\leq x_2 - x_1 + 1 \\
c+1 &\leq n_x \\
(n_y - n_x)(c+1) &\leq (n_y - n_x)n_x \\
n_y(c+1) &\leq (n_y - n_x)n_x + n_x(c+1) \\
(c+1)/n_x &\leq (n_y - n_x + c + 1)/n_y \\
(c+1)/n_x &\leq (y_2 - y_1 - x_2 + x_1 + c + 1)/n_y \\
\mathbb{P}_x(x_1 + c, x_2 - c) &\leq \mathbb{P}_y(y_2 - (x_2 - x_1 - c), y_1 + (x_2 - x_1 - c)).
\end{aligned}$$

□

3.4 Applying the Mixing Bound to Examples

Finally, we apply our upper bound on the mixing time (Theorem 2) to each of the example applications given in Section 2.2. The results are given below in Table 1. While our theorem gives the first formal proof bounding the mixing time of $\mathcal{M}_S$ that we are aware of, we expect that it is not a tight bound and further improvements are possible. Thus, the number of rounds given in the table while provably sufficient are likely more than needed.

Note that in our bound we have the term $\min(dn, 2S)$. Since we have not specified a specific sum in any of the examples we used dn for the bounds in the table. If the desired sum S satisfies $S < dn/2$ then our theorem could be used to obtain a smaller upper bound.

Table 1. Our upper bound for the rounds needed in the matching-based algorithm of Tajik et al.

Application	$\epsilon = 10^{-10}$	$\epsilon = 2^{-80}$
10x10 image block ($n = 100, d = 255$)	3,318	6,560
16x16 image block ($n = 256, d = 255$)	8,733	17,034
32x32 image block ($n = 1024, d = 255$)	36,351	69,555
Exam scores ($n = 300, d = 100$)	10,001	19,729
Salaries ($n = 30, d = 100000$)	1,139	2,111
Ratings ($n = 5000, d = 4$)	164,647	326,777

4 Approaches based on Ranking

In this section we describe algorithms for ranking and unranking d-bounded n-compositions of S based on two different total orderings. The first is the standard lexicographical order (Section 4.1) and the second is a new ordering we call recursive block ordering (Section 4.2). Both orderings rely heavily on precomputed information which we describe in Section 4.3. Finally in Sections 4.4 and 4.5 we describe the unrank algorithms for both orderings.

4.1 Lexicographical Ranking

In this section, we will use the lexicographical ordering on d-bounded n-compositions of S to generate a ranking. We will use the notation $<_L$ and $>_L$ to refer to lexicographical order. Specifically let $x, y \in \Omega_{S,d,n}$ be two arbitrary configurations such that $x \neq y$ and let i be the smallest integer such that $x(i) \neq y(i)$. If $x(i) < y(i)$ then $x <_L y$ otherwise $y >_L x$. For an example, see Fig. 2.

Configuration	Rank	Configuration	Rank
(0,3,3)	0	(2,3,1)	5
(1,2,3)	1	(3,0,3)	6
(1,3,2)	2	(3,1,2)	7
(2,1,3)	3	(3,2,1)	8
(2,2,2)	4	(3,3,0)	9

Fig. 2. The lexicographical ranking of configurations in $\Omega_{S,d,n} = \Omega_{6,3,3}$.

We begin by describing our ranking algorithm using the running example $S = 6, d = 3$, and $n = 3$ shown in Fig. 2. To rank a configuration x according to our ordering we start by determining how many configurations start with a number strictly less than $x(1)$. For example if $x = (2, 3, 1)$ there are three such configurations: $(0, 3, 3), (1, 2, 3)$, and $(1, 3, 2)$ (see Fig. 2). Next we determine the position of $(2, 3, 1)$ among configurations in $\Omega_{6,3,3}$ that start with 2. Since these configurations all start with 2 the remaining sum must add to 6-2 =4 and there are 3-1 = 2 remaining points. Thus this is equivalent to determining the rank of $(3, 1)$ in $\Omega_{6-2,3,3-1} = \Omega_{4,3,2}$ which is two. We then add the numbers together to get 5, the rank of $(2, 3, 1)$. More generally, let $C_d(n, S)$ be the number of n-compositions with sum S and component bound d and $\mathsf{l\text{-}rank}_{n,S}(x)$ be the lexicographical rank of x in $\Omega_{S,d,n}$. Given these definitions, we can define the rank recursively as follows.

```
1: procedure L-RANK((x_1, ..., x_n), S, CUM)
2:     rankSoFar ← 0
3:     startN ← n
4:     for i ← 1 to startN do
5:         if x_i ≠ 0 then
6:             rankSoFar ← rankSoFar + CUM[n − 1, S − x_i + 1]
7:             if S < CUM.length then
8:                 rankSoFar ← rankSoFar − CUM[n − 1, S + 1]
9:             end if
10:        end if
11:        S ← S − x_i
12:        n ← n - 1
13:    end for
14:    return rankSoFar
15: end procedure
```

Fig. 3. Lexicographic ranking algorithm with the cumulative sum table CUM.

$$\mathsf{l\text{-}rank}_{n,S}(x_1,\ldots,x_n) = \begin{cases} 0 & \text{if } n = 1 \\ \sum_{0\le i<x_1} C_d(n-1,S-i) + \mathsf{l\text{-}rank}_{n-1,S-x_1}(x_2,\ldots,x_n) & \text{if } n > 1 \end{cases}$$

It is straightforward to implement an algorithm for ranking given the definition above and a pre-computed C table storing the needed values.

In order to improve the efficiency of our ranking algorithm we will actually store the cumulative sum so position (i, j) in our table will store $\sum_{k=i}^{n} C_d(k, j)$. Using this cumulative sum table CUM, we give a more efficient ranking algorithm in Fig. 3. Note that the efficiency gain comes from not having to compute a sum in each iteration of the for loop.

4.2 Recursive Block Ranking

Next, we give an alternative ranking algorithm based on a new total ordering we call *recursive block order*. While recursive block order uses ideas that are reminiscent to those used in orderings of monomials, specifically block order (see e.g., [8]) and graded order (see e.g. [5]), these are combined differently and applied recursively unlike in monomial orderings. We will use the notation $<_B$ and $>_B$ to refer to recursive block order. Throughout this section we will assume n is a power of two. We can fairly easily generalize our ordering and algorithms to work when n is not a power of two and briefly describe the needed alterations at the end of the section. Let $x, y \in \Omega_{S,d,n}$ be two arbitrary configurations such that $x \neq y$. To compare x and y using recursive block order, we let x_L be the

first $n/2$ points in x, and x_R be the remaining points. Similarly define y_L as the first $n/2$ points in y and y_L to be the remaining points. We begin by considering the sum S_{x_L} of the points in x_L and similarly S_{y_L} (the sum of the points in y_L). If $S_{x_L} < S_{y_L}$ then $x <_B y$. Similarly, if $S_{x_L} > S_{y_L}$ then $x >_B y$. If the sums are equal then we will apply our ordering recursively. Specifically, if $x_L \neq y_L$ then $x <_B y$ if $x_L <_B y_L$ and $x >_B y$ if $x_L >_B y_L$. Finally if x_L is identical to y_L then $x <_B y$ if $x_R <_B y_R$ and $x >_B y$ if $x_R >_B y_R$. As a base case, if n is 1 then there is only one configuration with rank 0. See Figure 4 for an example of recursive block order. We summarize these conditions in the table below.

$$S_{x_L} < S_{y_L} \implies x <_B y \tag{2}$$

$$S_{x_L} > S_{y_L} \implies x >_B y \tag{3}$$

$$S_{x_L} = S_{y_L} \text{ and } x_L <_B y_L \implies x <_B y \tag{4}$$

$$S_{x_L} = S_{y_L} \text{ and } x_L >_B y_L \implies x >_B y \tag{5}$$

$$S_{x_L} = S_{y_L} \text{ and } x_L = y_L \text{ and } x_R <_B x_L \implies x <_B y \tag{6}$$

$$S_{x_L} = S_{y_L} \text{ and } x_L = y_L \text{ and } x_R >_B x_L \implies x >_B y \tag{7}$$

Configuration	Rank	Configuration	Rank
(0,2,3,3)	0	(2,1,2,3)	7
(1,1,3,3)	1	(2,1,3,2)	8
(2,0,3,3)	2	(3,0,2,3)	9
(0,3,2,3)	3	(3,0,3,2)	10
(0,3,3,2)	4	(1,3,1,3)	11
(1,2,2,3)	5	(1,3,2,2)	12
(1,2,3,2)	6	(1,3,3,1)	13

Fig. 4. The recursive block ranking of some configurations in $\Omega_{S,d,n} = \Omega_{8,3,4}$.

Based on the above definition of recursive block order we will give a recursive ranking algorithm. Again our algorithm will rely on pre-computed values stored in the C table where $C_d(n, S)$ is the number of n-compositions with sum S and component bound d. Let $\mathsf{b\text{-}rank}_n(x)$ be the recursive block rank of x in $\Omega_{S_x,d,n}$. In order to compute the rank of a configuration x we need to determine the number of configurations y such that $x >_B y$. Using the above definition of recursive block order such configurations fit into three cases given by Equations 3, 5, and 7 in the above table. Our algorithm will compute the number of configuration for each case and add the three to determine the rank of x. Consider first the case given by Equation 3. Here, we need to compute the number of configurations y that satisfy $S_{x_L} > S_{y_L}$. For each possible smaller sum S_{y_L} (i.e. any sum less than

```
1: procedure B-RANK((x_1, ..., x_n), S, C)
2:     if n == 1 then
3:         return 0
4:     end if
5:     X_L ← (x_1, ..., x_{n/2})
6:     X_R ← (x_{n/2+1}, ..., x_n)
7:     leftSum ← SUM(x_1, ..., x_{n/2})
8:     rightSum ← SUM(x_{n/2+1}, ..., x_n)
9:     case3 ← 0
10:    for s ← 0 to leftSum -1 do
11:        case3 ← case3 + C[n/2, s] · C[n/2, S − s]
12:    end for
13:    case5 ← C[n/2, rightSum]·B-RANK(X_L, leftSum, C)
14:    case7 ← B-RANK(X_R, rightSum, C)
15:    return case3 + case5 + case7
16: end procedure
```

Fig. 5. Recursive block ranking algorithm.

S_{x_L}) we compute the number of configurations with this left sum. We do this by using the C table to determine the number of choices for y_L and multiplying by the number of choices for y_R. This is exactly $\sum_{s=0}^{S_{x_L}-1} C(n/2, s) \cdot C(n/2, S-s)$. The second case is given by Equation 5, configurations with the same left sum but $y_L <_B x_R$. The number of such configurations y_L is given by $\mathsf{b\text{-}rank}_{n/2}(x_L)$. For each of these configurations there are $C(n/2, S_{x_R})$ choices for y_R. Thus the total number of such configurations is $\mathsf{b\text{-}rank}_{n/2}(x_L) \cdot C(n/2, S_{x_R})$. Finally the last case corresponding to Equation 7 is configurations with $y_L = x_L$ and $y_R <_B x_R$. There are $\mathsf{b\text{-}rank}_{n/2}(x_R)$ such configurations. Combining these gives the following recurrence for $\mathsf{b\text{-}rank}_n(x)$:

$$\sum_{s=0}^{S_{x_L}-1} C(n/2, s) \cdot C(n/2, S-s) + \mathsf{b\text{-}rank}_{n/2}(x_L) \cdot C(n/2, S_{x_R}) + \mathsf{b\text{-}rank}_{n/2}(x_R) \quad (8)$$

It is straightforward to design a recursive algorithm based on the above recurrence. Note that as a base case, if $n = 1$ then $\mathsf{b\text{-}rank}_1(x) = 0$ since there is only one configuration. Based on this equation we give a recursive algorithm for determining the recursive block rank in Figure 5. A key advantage of this algorithm is that it does not requirement the entire C table. As seen from Equation 8 in order to compute the rank of a configuration with n points we look at row $n/2$ of the C table and make two recursive calls both with $n/2$ points. Thus we only need to pre-compute the C table for rows that are powers of 2 resulting in only $\log n$ rows. Note that if n is not a power of 2 it is straightforward to generalize the rank algorithm and Theorem 4 below shows that we only need to at most double the number of rows that need to be pre-computed.

Lemma 4. *The recursive calls in each level of the recursion tree for the recursive block ranking algorithm will either all be the same size or have two unique sizes that differ by one.*

Proof. We will prove this using induction. As a base case the first call has a single size. We will assume inductively that the conditions are satisfied at a level i and show that they will continue to be satisfied at the next level. Recall that at each step of the recursion the size will be split in half. We will consider 2 different cases. If there is only one size x at level i then if x is even there will continue to be one size $x/2$ at the next level. If x is odd then there will be exactly two sizes that differ by one $(x-1)/2$ and $(x+1)/2$ at the next level. The second case we will consider is if there are two sizes x and $x+1$ that differ by one at level i. If x is even then at the next step the calls will all have sizes $x/2$ or $x/2+1$. If x is odd then at the next step the calls will all have sizes $(x-1)/2$ or $(x+1)/2$. In all cases the conditions of the theorem are satisfied.

4.3 Pre-Computing the C table

To support the rank and unrank algorithms for lexicographical and recursive block orderings we will need access to additional information stored in a table. We consider two approaches for pre-computing the information needed based on previous work in the area of counting restricted compositions. The first is based on dynamic programming techniques. While this algorithm is faster per table entry, it requires all values in the table to be computed. The second method is based on generating functions and while slower, does not rely on previously computed entries. This allows us to only compute the tables entries needed and is thus useful for the recursive block ranking algorithm. Again let $C_d(n, S)$ be the number of n-compositions with sum S and maximum value d.

Filling the Table with Dynamic Programming First we describe a dynamic programming based approach to filling the C table. We begin by developing and justifying a recurrence for $C_d(n, S)$. Note that this same recurrence is given previously by Abramson [1] although with a different more combinatorial explanation. For each n-composition with sum $S > 0, n > 1$ the first point is either 0 or some number greater than 0. The number of such compositions that start with 0 is equal to $C_d(n-1, S)$. For those that start with a number greater than 0, consider the quantity $C_d(n, S-1)$. For each of the compositions counted by $C_d(n, S-1)$ we can add one to the first point and obtain a n-composition with sum S and first point greater than 0. However, if $S > d$ some of these compositions will have the first point set to $d+1$ which is not a valid composition. Thus, in this case, $C_d(n, S-1)$ is over counting the number of n-compositions that start with a point greater than 0. If a n-composition starts with $d+1$, the remaining points are a $(n-1)$-compositions of the remaining sum $S-(d+1)$. Thus the excess number of compositions is exactly $C_d(n-1, S-(d+1))$. Combining these ideas gives the following recurrence for the C table.

$$C_d(n,S) = \begin{cases} 1 & \text{if } n = 1, S \leq d \\ 0 & \text{if } n = 1, S > d \\ 1 & \text{if } n = 0 \\ C_d(n{-}1,S) + C_d(n,S{-}1) & \text{if } n > 0, S \leq d \\ C_d(n{-}1,S) + C_d(n,S{-}1) - C_d(n{-}1,S{-}d{-}1) & \text{otherwise} \end{cases}$$

It is straightforward to see how using a nested loop and the above recurrence we can use fill the C table with dimensions $n \times (n*d)$ in time $\Theta(n^2 d)$. The value $n*d$ is chosen since this is the maximum possible sum. Note that Stein [17] uses a very similar approach to fill the table but does not use the dynamic programming framework or provide a justification for the recurrence. As mentioned above, in order to improve the efficiency of our lexicographical ranking and unranking algorithms we will actually store the cumulative sum so position (i,j) in the CUM table will store $\sum_{k=i}^{n} C_d(k,j)$.

Filling the Table with Generating Functions As an alternative approach to fill in the table, we can use formulas derived from a generating function view of the problem. This approach to counting restricted compositions is well established (see e.g. [1], [15]) and we briefly provide the details here for completeness. We will need the following well-known polynomial expansions:

$$\frac{1-x^{n+1}}{1-x} = 1 + x + x^2 + \ldots + x^n \tag{9}$$

$$(1+x)^n = 1 + \binom{n}{1}x + \binom{n}{2}x^2 + \ldots + \binom{n}{n}x^n \tag{10}$$

$$\frac{1}{(1-x)^n} = 1 + \binom{1+n-1}{1}x + \binom{2+n-1}{2}x^2 + \ldots \tag{11}$$

If $f(x)$ is a polynomial, we will use the notation $[x^k]f(x)$ to mean the coefficient of x^k in $f(x)$. So for example, $[x^3](1+x)^{15}$ would mean the coefficient of x^3 in the expansion of $(1+x)^{15}$, which from the identities above we can see is $\binom{15}{3}$.

We can use generating functions to compute the value $C_d(n,S)$ by noting that $C_d(n,S)$ will actually be the coefficient of the x^S term in the polynomial

$$(1 + x + x^2 + \ldots + x^d)^n \ .$$

Given this, we can then use the above identities to derive a formula for $C_d(n,S)$.

Lemma 5. *Consider vectors of length n with component bound d that sum to S. Then the number of such vectors is given by*

$$C_d(n,S) = [x^S](1 + x + x^2 + \ldots + x^d)^n = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \binom{n + S - (d+1)k - 1}{n-1}$$

Proof. From Equation (9) we can see that

$$[x^S](1+x+x^2+\ldots+x^d)^n = [x^S]\left(\frac{1-x^{d+1}}{1-x}\right)^n$$
$$= [x^S](1-x^{d+1})^n \cdot \frac{1}{(1-x)^n}$$

We can apply Equations (10) and (11) to the two parts of this last polynomial and see that

$$(1-x^{d+1})^n = 1 - \binom{n}{1}x^{d+1} + \ldots + (-1)^k\binom{n}{k}x^{k(d+1)} + \ldots + (-1)^n\binom{n}{n}x^{n(d+1)}$$

and

$$\frac{1}{(1-x)^n} = 1 + \binom{1+n-1}{1}x + \binom{2+n-1}{2}x^2 + \ldots$$

Given these two equations, we are interested in ways to get x^S. We need to account for possibly each term in the first equation combining with a term in the second equation. Specifically, if we have $x^{k(d+1)}$ from the first equation, then to get x^S we need the term $x^{S-k(d+1)}$ from the second. Summing up over all such possibilities and using the fact that $\binom{n}{k} = \binom{n}{n-k}$ we get

$$C_d(n,S) = \sum_{k=0}^{n}(-1)^k\binom{n}{k}\binom{S-k(d+1)+n-1}{S-k(d+1)}$$
$$= \sum_{k=0}^{n}(-1)^k\binom{n}{k}\binom{n+S-k(d+1)-1}{n-1}$$

□

In the next sections we describe the unrank algorithms for both lexicographical order and recursive block order. Both of these algorithms rely on the same pre-computed information as the associated rank algorithms.

4.4 Lexicographical Unrank

Next we describe the unranking algorithm for lexicographical order l-unrank. Note this is similar to the algorithm given by Stein [17] with the exception of using the cumulative sum table. We begin by describing our unrank algorithm again using the running example $S = 6, d = 3$, and $n = 3$ shown in Fig. 2. To unrank an integer r according to lexicographical order we start by determining x_1. We can do this again using our C table. For example we know that $x_1 = 0$ if the rank r is less than the number of configurations that start with 0 or $r < C_d(n-1,S)$. Similarly $x_1 = 1$ if $C_d(n-1,S) <= r < C_d(n-1,S-1)+C_d(n-1,S)$. In our example, x_1 is 1 if $1 \leq r < 3$. More generally,

$$x_1 = \max j : \sum_{i=S-j}^{S} C_d(n-1,i) < r.$$

```
1: procedure L-UNRANK(rank, S, n, CUM)
2:     (x_1, ..., x_n) ← (0, ..., 0)
3:     startN ← n
4:     for i ← 1 to startN -1 do
5:         if S < CUM.length then
6:             offset ← CUM[n − 1, S + 1]
7:         else
8:             offset ← 0
9:         end if
10:        while CUM[n − 1, S − x_i]−offset ≤ rank do
11:            x_i ← x_i + 1
12:        end while
13:        n ← n − 1
14:    end for
15:    x_startN ← S
16:    return x
17: end procedure
```

Fig. 6. Lexicographic unranking algorithm with the cumulative sum table CUM.

We can use this idea to define the unrank algorithm recursively. At each step i we will first determine x_i using the formula above and then recursively call the unrank algorithm to determine the remaining points $\mathsf{l\text{-}unrank}_{n-1,S-x_i}(r - \sum_{i=S-j+1}^{S} C_d(n-1,i))$. Based on this idea we give an l-unrank algorithm in Fig. 6 that again uses the cumulative sum table. Note that since we are using the cumulative sum table, the algorithm could be improved by replacing the while loop with a variation on binary search. More specifically, look at the value in the first position, the second, the fourth and so forth until a value greater than rank is found and then perform binary search.

4.5 Recursive Block Unrank

Finally we describe the unranking algorithm for the recursive block ordering b-unrank. At a high-level our algorithm begins by determining the sum of the left $n/2$ points. Next we determine the rank of x_L (i.e. $\mathsf{b\text{-}rank}_{n/2}(x_L)$) and the rank of x_R. Finally we apply our unrank algorithm recursively to each of these ranks to determine x_L and x_R. Throughout this section we will use the running example $S = 8, d = 3$, and $n = 4$ shown in Fig. 4.

We begin by showing how to determine the left sum S_{x_L}. In our example we know that $S_{x_L} = 2$ if the rank r is less than the total number of configurations with left sum 0,1, or 2. Since there are no configurations with left sum 0 or 1 the left sum is 2 if $r < C_d(n/2, 2) \cdot C_d(n/2, S-2) = 3 \cdot 1 = 3$. More generally,

$$S_{x_L} = \min s : r < \sum_{i=0}^{s} C_d(n/2, i) \cdot C_d(n/2, S-i).$$

```
1: procedure B-UNRANK(rank, S, n, C)
2:     if n== 1 then
3:         return (rank)
4:     end if
5:     leftPointsSmallerSum ← 0
6:     leftSum ← 0
7:     while leftPointsSmallerSum <= rank do        ▷ Determine the left sum
8:         leftPointsSmallerSum += C[n/2, leftSum] · C[n/2, n − leftSum]
9:         leftSum ← leftSum + 1
10:    end while                          ▷ The loop went one past the correct sum
11:    leftPointsSmallerSum -= C[n/2, leftSum] · C[n/2, n − leftSum]
12:    leftSum ← leftSum - 1
13:    rightPoints ← C[n/2, S − leftSum]
14:    leftRank ← (rank - leftPointsSmallerSum) / rightPoints
15:    rightRank ← (rank - leftPointsSmallerSum) % rightPoints
16:    left ← B-UNRANK(leftRank, leftSum, n/2, C)
17:    right ← B-UNRANK(rightRank, S - leftSum, n/2, C)
18:    return (left, right)
19: end procedure
```

Fig. 7. Recursive block unranking algorithm.

Next we will determine the rank of x_L. Note that each possible configuration of the left n points will occur in the ranking one time for each possible right configuration or $C_d(n/2, S - S_{x_L})$ times. For example, in our running example when $S_{x_L} = 3$ then there are 2 possible configurations for x_R namely $(2, 3)$ and $(3, 2)$ so each possible left configuration shows up in the ordering consecutively 2 times. Thus to determine the rank of the left configuration we first subtract the number of configurations with smaller sum from the rank and then divide by the number of right configurations $C_d(n/2, S - S_{x_L})$. To determine the right rank we again subtract the number of configurations with smaller sum and then determine the remainder when divided by $C_d(n/2, S - S_{x_L})$. Finally once we have the left and right ranks we can apply our b-unrank algorithm recursively to each rank. We give our b-unrank algorithm in Fig. 7.

5 Implementation and Performance Comparison

To compare the matching-based algorithm of Tajik et al. to the ranking algorithms we proposed in the previous section, we created prototype implementations of both and ran a number of performance tests. All implementations were done in Python 3.9.7 and performance tests were conducted on a machine with an Intel Core i5-8265U CPU @ 1.60GHz (1 socket, 4 cores per socket, and 2 threads per core) and 8 GB of RAM, running 64-bit Ubuntu Linux 18.04. Before looking at performance results, we discuss more details of our implementations.

5.1 Implementation details

Matching-based algorithm. We implemented the matching-based algorithm of Tajik et al. as we described it earlier. In each round, adjacent points are matched and rank-encipher-unrank is applied to each pair. We shuffled the points between rounds using the Knuth shuffle, which was suggested by Tajik et al. Like the previous work, we are considering NR security, so we implemented each rank-encipher-unrank using addition mod N, where N depends on the points being paired up and their sum. For our performance tests, we considered 50, 500, and 1000 rounds.

Ranking-based algorithms. We implemented both the lexicographic and recursive block ranking algorithms from the previous section to use with rank-encipher-encrypt. For lexicographic, we implemented the rank and unrank algorithms that use the cumulative sum table (Fig. 3 and Fig. 6). We also considered NR security, so the encryption portion was implemented using addition mod N, where N is the maximum rank given a vector length, component bound, and sum.

For the lexicographic ranking algorithm, we used a 2d numpy array with datatype object for the $C_d(n, S)$ table so that we could store unlimited precision Python integers in the arrays. This is necessary since the numbers in the table get very large; for the 16x16 image block example, the maximum value in the table is over 2000 bits.

For the recursive block ranking algorithm, since we only need a much smaller number of rows, we stored the needed rows of the $C_d(n, S)$ table in a Python dictionary indexed by n. We generated each entry using the generating function formula in Lemma 5. We also used the Python decorator `functools.cache` to memoize the choose function and speed up any repeated n choose k calculations that take place while generating the table.

5.2 Performance tests and results

To compare the performance of the matching-based and ranking-based solutions, we considered the applications mentioned earlier in Section 2.2 which have various parameter choices for n and d. For each application, we generated five vectors with n uniformly random elements chosen from 0 to d and measured the average time to encrypt using the iPython `%time` built-in "magic" command.

For the matching-based algorithm, we tested three choices of rounds, 50, 500, and 1000. The results are shown in Table 2. For the ranking-based solutions, we additionally measured the time to generate the C_d tables and also the eventual sizes of those tables. We determined the size by applying the Python `sys.getsizeof` method to each integer in the table, plus the result of applying `sys.getsizeof` to the table data structure itself. The results are shown in Table 3. It is interesting to note that the lexicographic ranking algorithm failed on the 32x32 block image encryption application and on the ratings application, due to the table size getting too large for the system to handle. The recursive block ranking, on the other hand, was successful on those two applications.

Table 2. Performance results for our implementation of the matching-based algorithm of Tajik et al.

Application	50 rounds	500 rounds	1000 rounds
10x10 image block ($n = 100, d = 255$)	0.06s	0.39s	0.77s
16x16 image block ($n = 256, d = 255$)	0.11s	0.98s	1.98s
32x32 image block ($n = 1024, d = 255$)	0.41s	3.91s	7.81s
Exam scores ($n = 300, d = 100$)	0.12s	1.14s	2.26s
Salaries ($n = 30, d = 100000$)	0.02s	0.13s	0.25s
Ratings ($n = 5000, d = 4$)	1.84s	18.5s	37.8s

Table 3. Performance results for our implementation of the ranking algorithms from the previous section. Encryption (Enc.) time is the time to do rank-encipher-unrank.

Application	Lexicographic			Recursive Block		
	Table size	Table time	Enc. time	Table size	Table time	Enc. time
10x10	162 MB	1.81s	0.009s	9 MB	0.87s	0.06s
16x16	1885 MB	13.1s	0.013	32 MB	5.8s	.18s
32x32	fail	-	-	271 MB	316s	3.50s
Exams	988MB	7.17s	0.011s	15 MB	3.81s	.096s
Salaries	4756 MB	79s	0.34s	672 MB	40.5s	4.2s
Ratings	fail	-	-	25 MB	271 s	0.40 s

5.3 Discussion

Looking at the results in Tables 2 and 3, we can come to a few conclusions.

For thumbnail encryption, ranking is likely the better choice. We can see from the tables that for the 10x10 and 16x16 image block encryption applications, the lexicographic ranking encryption time is faster than the matching-based solution with only 50 rounds. Even at the 32x32 block size, the recursive block ranking has faster encryption than 500 rounds of the matching-based solution. Since a large image contains thousands of such blocks, using the faster ranking-based solutions could be especially beneficial and well-worth the time and memory cost of generating and storing the table.

For small n but large d, the matching-based algorithm is a good choice. From the salaries application $n = 30, d = 100000$ we can see that the matching-based algorithm performs well and, even at 1000 rounds, encrypts in 1/4 of a second. The ranking solutions, on the other hand, are starting to hit their upper limits as far as table size. Specifically, the lexicographic ranking table took nearly 80 seconds to generate and ended up at 4756 MB in size. Even with this large table, encryption time for lexicographic ranking was still slightly slower than 1000 rounds of the matching-based algorithm. We also tried increasing n and d slightly to 50 and 250,000 and, as expected, the table size ended up being too large for our testing machine.

Ranking solutions should scale better if more security is desired. We have done the performance tests with NR (nonce-respecting) security in mind, which is consistent with previous work. However, if we instead want to target something like strong PRP security, then the ranking solutions should still be reasonably performant, while the matching solution will likely significantly slow down. The reason for this is that with the matching solution encryptions happen $n/2$ times (once for each pair of points) each round, and each of these, which are just additions mod N with NR security, would likely need to be replaced with a stronger cipher. With the ranking solutions, there is just a single cipher call that would need to be swapped for something stronger. One challenge with this, however, is that the numbers are very large with the ranking solutions (e.g., 2000 bit numbers in the 16x16 image example), so the inner cipher in rank-encipher-unrank would need to support such large sizes. A strong, variable-length cipher such as those in [16] may be a good choice. A closer look at these issues and whether NR security or strong PRP security is the right target for sum-preserving encryption applications would be interesting future work.

For larger n, new constructions are likely necessary. We can see from the ratings application that as n gets large both the matching solution and the ranking solutions start to struggle. The matching solution with 1000 rounds takes over 30 seconds to encrypt the vector with 5000 ratings; if we instead wanted to encrypt a vector with 1 million ratings, we might estimate it to take 10 minutes or longer! With the ranking solutions, our lexicographic table generation already failed on the ratings application with $n = 5000$. The recursive block ranking, at first glance, appears to be an acceptable solution, but the table generation time appears to drastically increase as n grows. Already at $n = 5000$ the table took almost 5 minutes to generate, and in additional tests with $n = 10000$ the table was not finished after 30 minutes. Scaling this up to something like 1 million ratings is just not practical. A better solution for very large n is likely to be a combination of the matching and ranking solutions. The matching algorithm is already based on the idea of applying rank-encipher-encrypt to smaller subvectors (of size 2), so a natural extension is to apply our ranking algorithms to increase the size of the subvectors enciphered each round. It is not immediately clear to us whether our proof from Section 3 could be adapted to this situation, so we leave proving a mixing time bound on this combined algorithm to future work.

Acknowledgements We thank the anonymous Asiacrypt 2022 reviewers for providing detailed comments and suggestions for improving the presentation of our results.

References

1. Abramson, M.: Restricted combinations and compositions. The Fibonacci Quarterly **14** (01 1976)

2. Bellare, M., Ristenpart, T., Rogaway, P., Stegers, T.: Format-preserving encryption. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 295–312. Springer, Heidelberg (Aug 2009). https://doi.org/10.1007/978-3-642-05445-7˙19
3. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (Apr 2009). https://doi.org/10.1007/978-3-642-01001-9˙13
4. Brightwell, M., Smith, H.: Using datatype-preserving encryption to enhance data warehouse security. In: National Information Systems Security Conference (NISSC) (1997)
5. Cox, D.A., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer Publishing Company, Incorporated, 3rd edn. (2010)
6. Dworkin, M.: Recommendation for block cipher modes of operation: Methods for format preserving-encryption. NIST Special Publication 800-38G, `http://dx.doi.org/10.6028/NIST.SP.800-38G` (2016)
7. Dyer, M., Greenhill, C.: A more rapidly mixing Markov chain for graph colorings. Random Structures & Algorithms **13**, 285–317 (1998)
8. El Kahoui, M., Rakrak, S.: Structure of grobner bases with respect to block orders. Math. Comput. **76**, 2181–2187 (10 2007). https://doi.org/10.1090/S0025-5718-07-01972-2
9. Hoang, V.T., Rogaway, P.: On generalized Feistel networks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 613–630. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7˙33
10. Institute, A.N.S.: Financial services - symmetric key cryptography for the financial services industry - format-preserving encryption. ANSI X9.124 Standard, `https://webstore.ansi.org/standards/ascx9/ansix91242020` (2020)
11. Levin, D.A., Peres, Y., Wilmer, E.L.: Markov chains and mixing times. American Mathematical Society (2006)
12. Marohn, B., Wright, C.V., Feng, W., Rosulek, M., Bobba, R.B.: Approximate thumbnail preserving encryption. In: Multimedia Privacy and Security – MPS@CCS 2017. pp. 33–43. ACM (2017)
13. Opdyke, J.: A unified approach to algorithms generating unrestricted and restricted integer compositions and integer partitions. Journal of Mathematical Modelling and Algorithms **9**, 53–97 (03 2010). https://doi.org/10.1007/s10852-009-9116-2
14. Page, D.: Generalized algorithm for restricted weak composition generation: Generation algorithm for second-order restricted weak compositions. Journal of Mathematical Modelling and Algorithms **12**, 345–372 (07 2012). https://doi.org/10.1007/s10852-012-9194-4
15. Riordan, J.: An Introduction to Combinatorial Analysis. Dover Publications, Inc. (12 1980). https://doi.org/10.1515/9781400854332
16. Shrimpton, T., Terashima, R.S.: A modular framework for building variable-input-length tweakable ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 405–423. Springer, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42033-7˙21
17. Stein, T.: Uniform random samples for second-order restricted k-compositions (January 2020), `http://essay.utwente.nl/80718/`
18. Tajik, K., Gunasekaran, A., Dutta, R., Ellis, B., Bobba, R.B., Rosulek, M., Wright, C.V., chi Feng, W.: Balancing image privacy and usability with thumbnail-preserving encryption. In: NDSS 2019. The Internet Society (Feb 2019)

19. Walsh, T.: Loop-free sequencing of bounded integer compositions. JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing **33** (01 2000)
20. Wright, C.V., Feng, W., Liu, F.: Thumbnail-preserving encryption for JPEG. In: ACM Workshop on Information Hiding and Multimedia Security – IH&MMSec 2015. pp. 141–146. ACM (2015)