Randomized Data Partitioning with Efficient Search, Retrieval and Privacy-Preservation

M. Oğuzhan Külekci[1][0000−0002−4583−6261]

Dept. of Computer Science, Indiana University Bloomington, IN, USA
okulekci@iu.edu

Abstract. We introduce a new data representation that serves mainly for privacy preserving data storage with efficient search and retrieval capabilities over the distributed systems. The cornerstone of the proposed scheme is based on a novel algorithm that splits an input bit sequence $\mathtt{B}[1..\mathtt{n}]$ into two as left and right partitions with well–control over the the partition sizes, and the reconstruction of $\mathtt{B}$ in absence of either partition is hard to achieve. The algorithm processes the input bit stream in blocks of $\mathtt{d}$–bits, where initially each block is replaced with another $\mathtt{d}$–bit according to a randomly chosen permutation of the set $\{0, 1, ..2^{\mathtt{d}}-1\}$. Following the replacement, the leftmost bits of each block up until and including the $\mathtt{q}$th set bit are appended to the left and the remaining bits to the right partition. We prove that the expected length of the left partition is $\ell \approx 2\mathtt{qn}/\mathtt{d}$ bits and the right partition becomes of length $|\mathtt{R}| = \mathtt{n} - \ell$ bits. Therefore, there is no overhead on the new representation with respect to original input. We also show that due to the randomization step, the input data $\mathtt{B}$ is not required to follow any special probability distribution to have the mentioned partitioning ratio $\rho = 2\mathtt{q}/\mathtt{d}$ and it is possible to tune the parameters $\mathtt{d}$ and $\mathtt{q}$ to support any desired ratio ρ on the input. We consider recursive application of that splitting algorithm on each partitions, which can be viewed as generating a full binary tree with $\mathtt{k}$–leaves such that at each internal node the data is subject to the proposed splitting operation. Such a construction represents an input bit sequence $\mathtt{B}[1..\mathtt{n}]$ with $\mathtt{k}$ partitions as $\mathtt{P}_1, \mathtt{P}_2, \ldots, \mathtt{P}_\mathtt{k}$, where it is hard to reconstruct the original data in absence of any $\mathtt{P}_\mathtt{i}$.

Keywords: randomization · data representation · data partitioning · distributed data storage · cloud storage · privacy-preserving data representation

1 Introduction

We introduce a randomized data representation that preserves the privacy of the data, while still supporting efficient access and retrieval. The building block of the proposed scheme is a novel randomized algorithm that splits an input $\mathtt{n}$–bit long sequence into two pieces, which we refer as the *left* and *right* partitions. The algorithm processes the input in blocks of a predetermined length $\mathtt{d}$ bits. The initial randomization step replaces each block on the input with its counterpart

according to a randomly generated permutation [8] of $\mathtt{d}$-bit integers. Following the replacement, the leftmost bits of each block up until and including the $\mathtt{q}$th set bit are appended to the left, and the remaining bits to the right partition.

We prove that the randomization step guarantees the *expected* size of the left partition to be $\ell = \mathtt{n} \cdot \rho$ bits, while the right partition becomes of length $(\mathtt{n} - \ell)$ bits according to the input parameter $\mathtt{0} < \rho < \mathtt{1}$ such that the $\mathtt{q}$ value can be chosen to satisfy $\rho = \mathtt{2q/d}$. Access to any block on the original input can be achieved efficiently with the intrinsic addressing in the construction. We show that it is also possible to search for a queried pattern by filtering on one partition and verifying on the other as well.

The privacy of the data in the proposed representation depends on combination of several factors, which can be summarized as follows. First, the random permutation used in the construction is assumed to be secret, which can be maintained by using a secret initial seed in the random number generator [8]. It is known that the permutations are vulnerable to statistical attacks [1, 16], but due to the splitting mechanism it becomes hard to make a frequency analysis on the left and right partitions. On the right partition, each block is represented with variable-length bits, where the code-word boundaries cannot be determined as these codes are not prefix-free, that lacks the possibility of frequency analysis. Actually, the left partition encodes those code-word boundary information. On the left partition, the code-word boundaries can be determined, however, since multiple randomly chosen symbols are represented with the same code, again running a frequency analysis becomes hard as $\mathtt{2^d}$ symbols on the input are mapped to $\mathtt{d}$ symbols on the left partition, which introduces again an ambiguity. Therefore, in absence of either partition and the secret permutation it is hard to reconstruct the original input, that provides the privacy in the proposed scheme. It is noteworthy that even the prefix-free codes have been found to be hard to decode without proper information [9, 18, 10], where recent works on combining compression and security [4, 7] considers the hardness of decoding asymetric numeral codes [6], which produce variable-length non-prefix-free codes.

By applying the proposed split operation recursively to each partition, we observe that an input sequence can be split into more than two partitions. This can be achieved by generating a full binary tree with $\mathtt{k}$–leaves, where the root node is the original input data. Each internal node has two children and the data on internal nodes are subject to the proposed splitting operation. Notice that in such a multiple partition it becomes more and more difficult to analyze and reproduce the original in absence of any partition as deeper the tree the ambiguity in the partitions increases.

There might be several applications of the proposed scheme, where a basic scenario is that a user wants to store a large file on the cloud or on a decentralized network such as the IPFS [3], where keeping data secret from the owners of the storage medium is a serious concern. Splitting data into multiple pieces and encrypting those partitions before saving to the cloud or the distributed network provides the ultimate secrecy. However, search and retrieval, which are vital operations on any data management scheme, becomes difficult to achieve on

such encrypted data. The encryption algorithms, mainly the homomorphic [14] and multi-party computation [5] schemes, support those operations, but their computational load might be prohibiting, particularly on distributed networks or IoT environments with limited resources [11, 19]. On such a case, the proposed scheme would serve as an alternative solution.

Such a scenario have been previously considered and data splitting have been proposed by Li *et al.* [13], where the authors proposed to use a key and create two bit streams from the input by using this key. It becomes hard to decode the original again in absence of the partitions and the key. Each bit stream is of equal length with the input, and hence, total storage requirement increases to $2n$ bits for n bit input. Yet another point is although random access might be supported, the search operation is not possible. Therefore, when compared to [13], the randomized algorithm in this study becomes advantageous in two dimensions by not creating an overhead storage and supporting efficient search and access.

The outline of the paper is as follows. In section 2 we introduce the proposed randomized data splitting algorithm and provide the proofs of well-control over the sizes of the partitions. The support mechanisms for efficient random access and search operations are explained in section 3, which id=s followed by the analysis of the privacy aspects in section 4. We describe how to partition a data into multiple splits according to desired proportions in section 5. We conclude by a summary and discussions of possible further research directions.

2 The Split Coding and Its Properties

Definition 1 (Split–Coding). *The split coding of an input* $\mathtt{d}$*-bit sequence* $\mathtt{A} = \mathtt{a}_1\mathtt{a}_2..\mathtt{a}_\mathtt{d}$ *according to the parameter* $\mathtt{q}$*,* $\mathtt{d}>\mathtt{q}>0$*, is denoted by* $\mathtt{S}(\mathtt{A},\mathtt{q}) \rightarrow \langle \mathtt{L}_\mathtt{A}, \mathtt{R}_\mathtt{A}\rangle$ *and produces the left partition* $\mathtt{L}_\mathtt{A}$ *and right partition* $\mathtt{R}_\mathtt{A}$ *as follows*

i) If $\mathtt{A}$ *has at least* $\mathtt{q}$ 1 *bits, then* $\mathtt{L}_\mathtt{A}=\mathtt{a}_1\mathtt{a}_2..\mathtt{a}_\mathtt{i}=(0^*1)^\mathtt{q}$ *and* $\mathtt{R}_\mathtt{A}=\mathtt{a}_{\mathtt{i}+1}...\mathtt{a}_\mathtt{d}=(0|1)^{\mathtt{d}-\mathtt{i}}$
ii) Otherwise, $\mathtt{L}_\mathtt{A} = \mathtt{a}_1\mathtt{a}_2..\mathtt{a}_\mathtt{d} = \mathtt{A}$ *and* $\mathtt{R}_\mathtt{A} = \emptyset$ *(empty).*

,where (0^*1) *denotes any number of* 0 *bits followed by a* 1 *bit.* $(0|1)$ *means either* 0 *or* 1 *bit. The superscripts over them represent how many times they are repeated.*

The split–coding that partitions an input $\mathtt{d}$–bit integer into two according to a parameter $\mathtt{q}$ is given in Definition 1, where some examples are $\mathtt{S}(01001101,1) \rightarrow \langle 01, 001101\rangle$, $\mathtt{S}(01001101,2) \rightarrow \langle 01001, 101\rangle$, $\mathtt{S}(01001101,4) \rightarrow \langle 01001101, \emptyset\rangle$. Let the n-bits long input bit sequence is shown by $\mathtt{B}[1..\mathtt{n}] = \mathtt{b}_1\mathtt{b}_2\ldots\mathtt{b}_\mathtt{n}$ and $\mathtt{d}\in\mathtt{O}(\log \mathtt{n})$ is a chosen fixed block-length, which we assume $\mathtt{d}|\mathtt{n}$, as $\mathtt{B}$ can be padded with random bits otherwise. The input $\mathtt{B}$ can be shown by $\mathtt{B} = \mathtt{D}_1\mathtt{D}_2\ldots\mathtt{D}_\mathtt{m}$, where $\mathtt{D}_\mathtt{i} = \mathtt{B}[\mathtt{d}(\mathtt{i}-1)+1..\mathtt{i}\cdot\mathtt{d}]$ is a block for $1 \leq \mathtt{i} \leq \mathtt{m}$, and $\mathtt{m} = \mathtt{n}/\mathtt{d}$. The $\mathtt{d}$–bits long blocks are the binary encoding of the integers from the set $\mathtt{U} = \{0, 1, .., 2^\mathtt{d}-1\}$.

Let $\pi[0..2^\mathtt{d}-1] = \{\pi_0, \pi_1, \ldots, \pi_{2^\mathtt{d}-1}\}$ be a *randomly generated* permutation of the set $\mathtt{U}$. We apply the permutation π on the input $\mathtt{B} = \mathtt{D}_1\mathtt{D}_2\ldots\mathtt{D}_\mathtt{m}$ that

i	1	2	3	4	5	6	7	8
D_i	100	101	001	000	010	110	111	011
$D'_i = \pi(D_i)$	011	111	010	101	000	100	110	001
L_i	01	1	01	1	000	1	1	001
R_i	1	11	0	01		00	10	

Fig. 1. The split-coding of $\mathtt{B} = 100101001000010110111011$ with $\mathtt{d} = 3$, $\mathtt{q} = 1$, and $\pi = \langle 5, 2, 0, 1, 3, 7, 4, 6\rangle$ as $\mathtt{P}(\mathtt{B}, \mathtt{d}, \pi, \mathtt{q}) \rightarrow (\mathtt{L} = 01101100011001, \mathtt{R} = 1110010010)$. The decoding process to merge $\mathtt{L_B}$ and $\mathtt{R_B}$ partitions to restore $\mathtt{B}$ can also be traced on the example strings. For example, scanning the initial bits from L returns 2 bits sequence $\mathtt{01}$ as we hit a set bit. Therefore, we need to append $1 = 3-2$ bits from R, which creates $\mathtt{D'_1} = \mathtt{011}$. Applying the inverse permutation $\mathtt{D_1} = \pi^{-1}(3) = 4$ restores the original bits $\mathtt{100}$.

produces $\mathtt{B'} = \mathtt{D'_1 D'_2 \ldots D'_m}$ such that $\mathtt{D'_i} = \pi[\mathtt{D_i}]$. The random permutation π can be constructed by using a pseudo-random number generator that is initialized with a selected *seed*. Following the permutation step, we apply split–coding to each $\mathtt{d}$-bits long block $\mathtt{D'_{1 \le k \le m}}$ that produces the left and right partitions as $\mathtt{S}(\mathtt{D'_k}, \mathtt{q}) \rightarrow \langle \mathtt{L_k}, \mathtt{R_k}\rangle$ according to the chosen parameter $\mathtt{q}$, $0<\mathtt{q}<\mathtt{d}$. For each block, the left partition $\mathtt{L_k}$ is the sequence of leftmost bits up until and including the $\mathtt{q}$th set bit, and the right partition $\mathtt{R_k}$ is the remaining bits succeeding the $\mathtt{q}$th set bit. As stated in definition 1, if $\mathtt{q}$th set bit appears on the last position or there are less than $\mathtt{q}$th set bits on the block, then the right partition will be empty, and the block is copied into left partition and nothing is written to the right partition. We concatenate the left partitions $\mathtt{L_1}$ to $\mathtt{L_m}$ in order and generate the L–partition as $\mathtt{L_B} = \mathtt{L_1 L_2 .. L_m}$. Similarly, $\mathtt{R_B} = \mathtt{R_1 R_2 .. R_m}$ denotes the R–partition. This complete process is represented by $\mathtt{SplitEncode}(\mathtt{B}, \mathtt{d}, \pi, \mathtt{q}) \rightarrow \langle \mathtt{L}, \mathtt{R}\rangle$, which is also shown in pseudocode Algorithm **??** in the appendix.

Given the L– and R–partitions, the permutation π, and the parameters $\mathtt{q}$ and $\mathtt{d}$, the decoding process reconstructs $\mathtt{B}$ by simply scanning the L- and R–partitions from left-to-right. The decoding procedure starts by initializing the read pointers on $\mathtt{L_B}$ and $\mathtt{R_B}$ to their first positions, and reads bits from $\mathtt{L_B}$–partition until $\mathtt{q}$ set bits are observed or total number of bits examined reaches $\mathtt{d}$ with less than q set bits. Assuming $\mathtt{t}$ bits are read on $\mathtt{L_B}$, the remaining $(\mathtt{d}-\mathtt{t})$ bits are read from the $\mathtt{R_B}$–partition in order. The concatenation of these bits creates a $\mathtt{d}$–bits long block, say $\mathtt{T}$, from which the original block is restored via $\pi^{-1}[T]$. This procedure is repeatedly applied until all $\mathtt{m}$ blocks are restored as listed on Algorithm **??** in the appendix. A sample split encoding and decoding is depicted on Figure 1 with the parameters $\mathtt{q} = 1$, $\mathtt{d} = 3$, and $\pi = \langle 5, 2, 0, 1, 3, 7, 4, 6\rangle$ on a bitmap $\mathtt{B} = 100101001000010110111011$.

We now analyze the expected bit lengths of the partitions created by split–coding process on a $\mathtt{n}$ bits long input $\mathtt{B}$ and prove that the expected length of the left partition is $\mathtt{n} \cdot 2\mathtt{q}/\mathtt{d}$ bits long, which leaves the right partition to be $\approx \mathtt{n} \cdot (\mathtt{d} - 2\mathtt{q})/\mathtt{d}$ bits long.

Proposition 1 (Length Invariance). *The sum of the lengths of the L– and R– partitions created by the split-coding* $\texttt{SplitEncode}(\texttt{B}, \texttt{d}, \pi, \texttt{q}) \rightarrow \langle \texttt{L}_\texttt{B}, \texttt{R}_\texttt{B} \rangle$ *is equal to the length of the input bit string* B.

Proof. Each d-bits long block $\texttt{D}_\texttt{i}$ on B is split into two bit strings $\texttt{L}_\texttt{i}$ and $\texttt{R}_\texttt{i}$ without any insertion or deletion of bits. Therefore, per each block, the lengths of its corresponding bits on $\texttt{L}_\texttt{B}$ and $\texttt{R}_\texttt{B}$ partitions sum up to d as $\texttt{L}_\texttt{i} + \texttt{R}_\texttt{i} = \texttt{d}$ for all $\texttt{i} = 1$ to m. Thus, $\sum_{1 \leq \texttt{i} \leq \texttt{m}} (\texttt{L}_\texttt{i} + \texttt{R}_\texttt{i}) = \texttt{m} \cdot \texttt{d} = |\texttt{B}|$ holds. □

Lemma 1 (Expected bit lengths of partitions for $\texttt{q} = 1$). *For* $\texttt{q} = 1$ *and* $\texttt{d} \geq 1$*, the* ***expected*** *bit lengths of the* L *and* R *partitions of a randomly chosen integer from* $\texttt{U} = \{0, 1, ..., 2^\texttt{d} - 1\}$ *are* $\texttt{E}(|\texttt{L}|) = 2 - \frac{1}{2^{\texttt{d}-1}}$ *and* $\texttt{E}(|\texttt{R}|) = \texttt{d} - 2 + \frac{1}{2^{\texttt{d}-1}}$.

Proof. A randomly selected d-bit integer i can take any value in $\{0, 1, \ldots, 2^\texttt{d} - 1\}$ with probability $1/2^\texttt{d}$. For all values of $\texttt{i} \geq 2^{\texttt{d}-1}$, the L partition will be equal to 1, and thus, $|\texttt{L}| = 1$. Since there are $2^{\texttt{d}-1}$ such i values, the probability that the length of the L partition will be one is $2^{\texttt{d}-1}/2^\texttt{d} = 1/2$. Similarly, if $2^{\texttt{d}-2} \leq \texttt{i} < 2^{\texttt{d}-1}$, then L will be 01 that will dictate $|\texttt{L}| = 2$ with probability $2^{\texttt{d}-2}/2^\texttt{d} = 1/4$, which can be generalized to $|\texttt{L}| = \ell$ with probability $1/2^\ell$ for $1 \leq \ell \leq (\texttt{d} - 1)$. When $\texttt{i} = 0$ or $\texttt{i} = 1$, the L partitions will completely include i, and thus, the corresponding length $|\texttt{L}|$ will be d with probability $2/2^\texttt{d}$. Therefore, the expected length $\texttt{E}(|\texttt{L}|)$ by the split coding can be calculated as follows.

$$\texttt{E}(|\texttt{L}|) = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \cdots + \frac{1}{2^{\texttt{d}-1}} \cdot (\texttt{d} - 1) + \frac{2}{2^\texttt{d}} \cdot \texttt{d} \;=\; \frac{2}{2^\texttt{d}} \cdot \texttt{d} + \sum_{\ell=1}^{\texttt{d}-1} \ell \cdot \frac{1}{2^\ell} \quad (1)$$

The algebraic identity $\sum_{\texttt{i}=1}^{\texttt{n}} \texttt{i} \cdot \texttt{r}^\texttt{i} = \frac{\texttt{r} - \texttt{r}^{\texttt{n}+1}(1+\texttt{n}-\texttt{nr})}{(1-\texttt{r})^2}$ with $\texttt{r}=1/2$ and $\texttt{n}=\texttt{d}-1$, reduces Equation 1 to

$$\texttt{E}(|\texttt{L}|) = \frac{2\texttt{d}}{2^\texttt{d}} + 4 \cdot \left(\frac{1}{2} - \frac{\texttt{d}+1}{2^{\texttt{d}+1}}\right) \;=\; \frac{2\texttt{d}}{2^\texttt{d}} + 2 - \frac{2\texttt{d}+2}{2^\texttt{d}} \;=\; 2 - \frac{1}{2^{\texttt{d}-1}} \quad (2)$$

For any integer i the sum of the lengths of L and R sum up to d as $|\texttt{L}| + |\texttt{R}| = \texttt{d}$ by definition and $\texttt{E}(|\texttt{R}|) = \texttt{E}(\texttt{d} - |\texttt{L}|) = \texttt{d} - \texttt{E}(|\texttt{L}|) = \texttt{d} - 2 + 1/2^{\texttt{d}-1}$ holds. □

Lemma 2 (Expected bit lengths of the partitions on a d-bit block). *For any* $0 < \texttt{q} < d/2$*, the* ***expected*** *bit lengths of the* L– *and* R– *partitions of a randomly chosen* d*-bit integer from* $\texttt{U} = \{0, 1, \ldots, 2^\texttt{d} - 1\}$ *are*

$$\texttt{q} \leq \texttt{E}(|\texttt{L}|) \leq 2\texttt{q} - \frac{\texttt{q}}{2^{\texttt{d}-1}} \quad \textit{and} \quad (\texttt{d} - \texttt{q}) > \texttt{E}(|\texttt{R}|) > \texttt{d} - 2\texttt{q} + \frac{\texttt{q}}{2^{\texttt{d}-1}}$$

Proof. Let $\texttt{p}_1, \texttt{p}_2, ..\texttt{p}_\texttt{q}$ denote the first q set bit positions on the randomly chosen integer $\texttt{i} \in \{0, 1, .., 2^\texttt{d} - 1\}$, and $\texttt{x}_1, \texttt{x}_2, ..\texttt{x}_\texttt{q}$ represent the differences in between the positions such that $\texttt{x}_1 = \texttt{p}_1$, and $\texttt{x}_\texttt{j} = \texttt{p}_\texttt{j} - \texttt{p}_{\texttt{j}-1}$. For instance, if the positions

of the first four set bits are, say $2, 5, 6, 8$, then the $\langle \mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3, \mathtt{x}_4 \rangle = \langle 2, 3, 1, 2 \rangle$. The length of the $\mathtt{L}$ partition is then $|\mathtt{L}| = \mathtt{x}_1 + \mathtt{x}_2 + .. + \mathtt{x}_\mathtt{q}$.

We observe that $\mathtt{x}_\mathtt{j}$ is the length of the $\mathtt{L}$–partition of a $\mathtt{d}_\mathtt{j} = \mathtt{d} - \sum_{\mathtt{r}=1}^{\mathtt{j}-1} \mathtt{x}_\mathtt{r}$ bits long sequence. Due to Lemma 1, $\mathtt{E}(\mathtt{x}_\mathtt{j}) = 2 - 1/2^{\mathtt{d}_\mathtt{j}-1}$. Since the lengths of the sequences decrease at each step as $\mathtt{d}_1 > \mathtt{d}_2 > .. > \mathtt{d}_\mathtt{q}$, the expected values decline accordingly $\mathtt{E}(\mathtt{x}_1) > \mathtt{E}(\mathtt{x}_2) > .. > \mathtt{E}(\mathtt{x}_\mathtt{q})$, e.g., for $\mathtt{d}$=8, $\mathtt{E}(|\mathtt{L}|) = 2 - 1/2^7 = 1.992$, where for $\mathtt{d}$=6, $\mathtt{E}(|\mathtt{L}|) = 2 - 1/2^7 = 1.984$. That defines $\mathtt{E}(|\mathtt{L}|)$ to be upper bounded by $\mathtt{q}{\cdot}\mathtt{E}(\mathtt{x}_1)$ due to $\mathtt{E}(\mathtt{x}_1){>}\mathtt{E}(\mathtt{x}_2){>}..{>}\mathtt{E}(\mathtt{x}_\mathtt{q})$ as shown in equation 5. Therefore, the expected value $\mathtt{E}(|\mathtt{L}|)$ can be represented as in equation 3 due to the linearity of expectations.

$$\mathtt{E}(|\mathtt{L}|) = \mathtt{E}(\mathtt{x}_1 + \mathtt{x}_2 + .. + \mathtt{x}_\mathtt{q}) = \sum_{\mathtt{j}=1}^{\mathtt{q}} \mathtt{E}(\mathtt{x}_\mathtt{j}) \tag{3}$$

$$= 2\mathtt{q} - \Big(\frac{1}{2^{\mathtt{d}_1-1}} + \frac{1}{2^{\mathtt{d}_2-1}} + .. + \frac{1}{2^{\mathtt{d}_\mathtt{q}-1}} \Big) \tag{4}$$

$$\leq \mathtt{q} \cdot \mathtt{E}(\mathtt{x}_1) = 2\mathtt{q} - \frac{\mathtt{q}}{2^{\mathtt{d}-1}} \tag{5}$$

By definition 1, the length of the $\mathtt{L}$–partition is at least one bit, and thus, $\mathtt{x}_\mathtt{i} \geq 1$ holds for each $\mathtt{i}$, which provides the lower bound for $\mathtt{E}(|\mathtt{L}|) \geq \mathtt{q}$. Again by using $|\mathtt{L}| + |\mathtt{R}| = \mathtt{d}$, $\mathtt{E}(|\mathtt{R}|) = \mathtt{d} - \mathtt{E}(|\mathtt{R}|)$, the upper– and lower–bounds for $\mathtt{E}(|\mathtt{R}|)$ is obtained. □

Theorem 1. *The split-coding of a given* $\mathtt{n}$ *bits long* $\mathtt{B}$ *with a randomly chosen permutation* π*, block length* $\mathtt{d}$*, and parameter* $0 < \mathtt{q} \leq \mathtt{d}/2$ *generates left partition* $\mathtt{L}_\mathtt{B}$ *with an expected length of* $\mathtt{E}(|\mathtt{L}_\mathtt{B}|) \leq n \cdot (\frac{2q}{d} - \frac{q}{d \cdot 2^{d-1}})$*. Accordingly, the right partition is expected to be* $\mathtt{E}(|\mathtt{R}_\mathtt{B}|) > \mathtt{n} \cdot (1 - \frac{2\mathtt{q}}{\mathtt{d}} + \frac{\mathtt{q}}{\mathtt{d} \cdot 2^{\mathtt{d}-1}})$ *bits long.*

Proof. Split-coding operates on the $\mathtt{n}$-bits long input sequence by $\mathtt{n}/\mathtt{d}$ blocks, each of which is a $\mathtt{d}$-bit integer. The blocks are replaced by their corresponding integers in the set $\{0, 1, 2, \ldots, 2^\mathtt{d} - 1\}$ with the the randomly generated permutation π. Let $\mathtt{X}_1, \mathtt{X}_2, \ldots \mathtt{X}_{2^\mathtt{d}}$ denote the lengths of the $\mathtt{L}$–partitions of randomly assigned integers to each distinct $\mathtt{d}$-bit integers and $\mathtt{f}_1, \mathtt{f}_2, \ldots, \mathtt{f}_{2^\mathtt{d}}$ represent the frequencies of each integer. The expected length $\mathtt{E}(|\mathtt{L}(\mathtt{B})|)$ for the $\mathtt{L}$–partition of input $\mathtt{B}$ can be expressed as

$$\mathtt{E}(|\mathtt{L}_\mathtt{B}|) = \mathtt{E}\Big(\sum_{\mathtt{i}=1}^{2^\mathtt{d}} \mathtt{f}_\mathtt{i} \cdot \mathtt{X}_\mathtt{i} \Big) = \sum_{\mathtt{i}=1}^{2^\mathtt{d}} \mathtt{f}_\mathtt{i} \cdot \mathtt{E}(\mathtt{X}_\mathtt{i}) \qquad \text{by linearity of expectations}$$

$$\leq \sum_{\mathtt{i}=1}^{2^\mathtt{d}} \mathtt{f}_\mathtt{i} \cdot (2\mathtt{q} - \frac{\mathtt{q}}{2^{\mathtt{d}-1}}) \qquad \text{by substituting } \mathtt{E}(\mathtt{X}_\mathtt{i}) \text{ due to Lemma 2}$$

$$\leq (2\mathtt{q} - \frac{\mathtt{q}}{2^{\mathtt{d}-1}}) \cdot \frac{\mathtt{n}}{\mathtt{d}} \qquad \text{since } \sum_{\mathtt{i}=1}^{2^\mathtt{d}} \mathtt{f}_\mathtt{i} = \frac{\mathtt{n}}{\mathtt{d}} \text{ by definition}$$

$$\leq \mathtt{n} \cdot \Big(\frac{2\mathtt{q}}{\mathtt{d}} - \frac{\mathtt{q}}{\mathtt{d} \cdot 2^{\mathtt{d}-1}} \Big)$$

Since $|\mathtt{L_B}| + |\mathtt{R_B}| = \mathtt{n}$, $\mathtt{E}(|\mathtt{R_B}|) = \mathtt{n} - \mathtt{E}(|\mathtt{L_B}|) > \mathtt{n} \cdot \left(1 - \frac{2\mathtt{q}}{\mathtt{d}} + \frac{\mathtt{q}}{\mathtt{d} \cdot 2^{\mathtt{d}-1}}\right)$. □

Corollary 1. *The lengths of the* $\mathtt{L_B}$ *and* $\mathtt{R_B}$ *partitions of an input* $\mathtt{n}$*-bits long* $\mathtt{B}$ *approach respectively to* $2\mathtt{nq}/\mathtt{d}$ *and* $\mathtt{n}(\mathtt{d}-2\mathtt{q})/\mathtt{d}$ *bits as the term* $\mathtt{q}/\mathtt{d} \cdot 2^{\mathtt{d}-1}$ *in Theorem 1 becomes insignificant especially on larger values of* $\mathtt{d}$*, e.g.,assuming* $\mathtt{d} = 8$ *and* $\mathtt{q} = 1$*,* $\mathtt{q}/\mathtt{d} \cdot 2^{\mathtt{d}-1} = 1/(8 \cdot 2^7) = 1/2^{10} = 0.00097$*, which makes* $\mathtt{E}(|\mathtt{L}(\mathtt{B})|) = 0.24903\mathtt{n}$ *and* $\mathtt{E}(|\mathtt{R}(\mathtt{B})|) = 0.75097\mathtt{n}$ *bits that can be assumed to be* $0.25n$ *and* $0.75n$*.*

3 The Random Access and Search Operations

In this section we investigate the random access and search mechanisms of the split-coding scheme.

Lemma 3 (Random access support). *Let* $\mathtt{Q} = \binom{\mathtt{d}}{0} + \binom{\mathtt{d}}{1} + .. + \binom{\mathtt{d}}{\mathtt{q}-1}$ *denote the number of distinct* $\mathtt{d}$*-bit integers that has less than* $\mathtt{q}$ *set bits. Given the* $\mathtt{L_B}$ *and* $\mathtt{R_B}$ *partitions of an input sequence* $\mathtt{B}$*, if the number of* ***unique*** $\mathtt{d}$*-bit blocks on* $\mathtt{B}$ *is less than* $2^{\mathtt{d}} - \mathtt{Q}$*, then the random access to any block* $\mathtt{D_i}$ *of* $\mathtt{B} = \mathtt{D_1 D_2} \ldots \mathtt{D_m}$ *can be achieved in* $\mathtt{O}(\log |\mathtt{L_B}|)$*–time by using an extra space of* $\mathtt{o}(|\mathtt{L_B}|)$ *bits. Otherwise, it takes* $\mathtt{O}(\mathtt{q}\log(\mathtt{n}/\mathtt{d}))$*–time at the expense of* $\mathtt{O}((\mathtt{n}/\mathtt{d}) \cdot (\mathtt{Q}/2^{\mathtt{d}}) \cdot \log \mathtt{n})$ *additional bits space.*

Proof. Once the $\mathtt{L_B}$ and $\mathtt{R_B}$ partitions are given, the retrieval of any $\mathtt{d}$–bit block $\mathtt{D'_i}$ on $\mathtt{B'} = \mathtt{D'_1 D'_2}..\mathtt{D'_m}$ can be done by using the observation that $\mathtt{D'_1 D'_2}..\mathtt{D'_{i-1}}$ is $\mathtt{w} = \mathtt{d} \cdot (\mathtt{i}-1)$ bits long, where ℓ of those bits have been deposited on $\mathtt{L_B}$ and the rest $(\mathtt{w} - \ell)$ on $\mathtt{R_B}$. Therefore, given ℓ, we know the starting bit position of the left and right partitions of queried $\mathtt{D'_i}$ on $\mathtt{L_B}$ and $\mathtt{R_B}$, respectively.

We start reading the bits on $\mathtt{L_B}$ from the first position and keep scanning until we read either $\mathtt{d}$ bits or encounter $\mathtt{q}$ set bits, whichever happens first. Once, one of these cases occurs, then the bits are actually the left partition of the first block of the input. Re-initiating the same procedure starting from the next bit position will retrieve the left partition of the second block, and so on. After repeating this operation $(\mathtt{i}-1)$ times, we reach the first bit of the left partition $\mathtt{L_i}$ of the queried $\mathtt{D'_i}$. If we have scanned ℓ bits until we reach the beginning of $\mathtt{L_i}$, then it is immediate that the $\mathtt{R_i}$ starts just after $(\mathtt{w} - \ell)$ bits on R_B. Therefore, once we retrieve $\mathtt{L_i}$ from $\mathtt{L_B}$, we read $\mathtt{R_i}$ by depositing the remaining $(\mathtt{d} - |\mathtt{L_i}|)$ bits from the starting bit position $(\mathtt{w}-\ell+1)$ of the $\mathtt{R_B}$. Combining the extracted partitions will reconstruct $\mathtt{D'_i} = \mathtt{L_i R_i}$ and deploying the inverse permutation would return the original block $\mathtt{D_i} = \pi^{-1}(\mathtt{D'_i})$.

The time complexity of that random access procedure depends on finding ℓ, since once ℓ is known, rest is simple constant time operation. Let's first assume each block in $\mathtt{D'_1 D'_2}..\mathtt{D'_{i-1}}$ includes at least $\mathtt{q}$ set bits. Then, the left partitions of these blocks have exactly $\mathtt{q}$ set bits and each end with a set bit on $\mathtt{L_B}$, e.g., the initial bits of $\mathtt{L_B}$ up until and including the $\mathtt{q}$th set bit are the left partition of $\mathtt{D'_1}$. Therefore, $\ell+1$ is the position of the first bit after the $\mathtt{q} \cdot (\mathtt{i}-1)$th set bit on $\mathtt{L_B}$. Finding the position of the $\mathtt{k}$th set bit on a static bitmap of $\mathtt{n}$ bits can

be achieved in constant or logarithmic time by constructing a dictionary structure (the *rank/select*(R/S) dictionary) that occupies $\mathtt{o(n)}$ bits space [15, 20, 17]. Therefore, the position of the $\mathtt{q \cdot (i-1)}$th set bit on $\mathtt{L_B}$ can be returned by creating that R/S data structure in expense of $\mathtt{o(|L_B|)}$ additional space. In this case, detecting the ℓ value can be achieved in $\mathtt{O(\log |L_B|)}$ in the worst case, and thus, the random access. On the other hand, possible occurrences of the blocks with less than $\mathtt{q}$ set bits, which are directly copied to the $\mathtt{L_B}$, corrupts the above procedure. That requires maintaining the indices of such blocks separately. Assume $\mathtt{S_z}$ is the sorted list of block ids that have $\mathtt{z}$ set bits for all $\mathtt{z} < \mathtt{q}$. On list $\mathtt{S_z}$, let the number of items less than the queried $\mathtt{i}$ is represented by $\mathtt{c_z}$. Therefore, the total number of set bits of $\mathtt{D'_1 D'_2 .. D'_{i-1}}$ on $\mathtt{L_B}$ is $\ell = \mathtt{q} \cdot (\mathtt{i} - \sum_{z=0}^{q-1} \mathtt{c_z}) + \sum_{z=0}^{q-1} (\mathtt{z} \cdot \mathtt{c_z})$. As a last step to find the initial position of $\mathtt{D'_i}$, it is necessary to check whether the immediately preceding blocks are in $\mathtt{S_0}$, and the correct position is computed accordingly. Algorithm 1 lists the complete pseudo-code to retrieve $\mathtt{D_i}$.

Algorithm 1: RandomAccess($\mathtt{i}, \mathtt{L_B}, \mathtt{R_B}, \mathtt{d}, \pi, \mathtt{q}, \mathtt{S_0}, \mathtt{S_1}, .., \mathtt{S_{q-1}}$)

input : $\mathtt{i}$ is the queried block index, $\mathtt{L_B}$ and $\mathtt{R_B}$ are the partitions, $\mathtt{d}, \mathtt{q}, \pi$ are the parameters of the coding, $\mathtt{S_z}$ is the list of block indices that has $z < q$ set bits.

output: The original $\mathtt{d}$–bit block $\mathtt{D_i}$.

```
1  C ← 0
2  r ← 1
3  for z = 0 to (q − 1) do
4  |  t ← number of indices less than i in S_z
5  |  C ← C + z · t
6  |  r ← r + t
7  end
8  ℓ ← q · (i − r) + C
9  p ←1+ The position of the ℓth set bit on L_B
10 r ← i − 1
11 while (r ∈ S_0) do p ← p + d, r ← r − 1
12 detected ← 0, step ← 0
13 while ((detected < q) ∧ (step < d)) do
14 |  if L_B[p + step] == 1 then detected++
15 |  step++
16 end
17 L ← L_B[p..p + step − 1]
18 w ← d · (i − 1) − p + 1
19 R ← R_B[w..w + (d − step) − 1]
20 D'_i ← L ⌢ R
21 return π^{-1}(D'_i)
```

The space overhead for the random access support by Algorithm 1 is the sum of the space occupied by the $\mathtt{S_z}$ lists and the additional space used by rank/select

(R/S) data structure for efficient execution of line 9. The number of unique $\mathtt{d}$–bit sequences that have less than $\mathtt{q}$ set bit is $\mathtt{Q} = \binom{\mathtt{d}}{0} + \binom{\mathtt{d}}{1} + .. + \binom{\mathtt{d}}{\mathtt{q}-1}$. Assuming all distinct $2^{\mathtt{d}}$ blocks are equally likely among observed $(\mathtt{n}/\mathtt{d})$ blocks of the input, the size of the $\mathtt{S_z}$ list is expected to be $\approx (\mathtt{n}/\mathtt{d}) \cdot (\mathtt{Q}/2^{\mathtt{d}})$. Therefore, the space required to store the $\mathtt{S_z}$ lists becomes around $\approx (\mathtt{n}/\mathtt{d}) \cdot (\mathtt{Q}/2^{\mathtt{d}}) \cdot \log \mathtt{n}$ bits, where the space overhead of the (R/S) dictionaries can be kept around $\mathtt{o}(|\mathtt{L_b}| \approx 2\mathtt{qn}/\mathtt{d})$ with the state-of-the-art implementations. The time complexity of Algorithm 1 depends on detecting the blocks with less than $\mathtt{q}$ set bits via the `for` loop in between lines 3 and 7 and also the `select` query run on line 9. The `for` loop takes roughly $\mathtt{O}(\mathtt{q}\log(\mathtt{n}/\mathtt{d}))$–time where the select query can be executed in $\mathtt{O}(1)$ or $\mathtt{O}(\log(2\mathtt{qn}/\mathtt{d}))$ time according to the R/S data structure used. Thus, the total time complexity is upper bounded by $\mathtt{O}(\mathtt{q}\log(\mathtt{n}/\mathtt{d}))$. □

It is important to note that if the number of distinct $\mathtt{d}$–bit symbols on the input is less than $2^{\mathtt{d}} - \mathtt{Q}$, then the random permutation over integers $\{\mathtt{Q}, \mathtt{Q}+1, .., 2^{\mathtt{d}}-1\}$ can be used for the observed blocks, which guarantees to have at least $\mathtt{q}$ set bits per block on $\mathtt{L_B}$, and thus, removes the necessity of maintaining $\mathtt{S_z}$ lists . In such a case, the time and space complexity of the random access is determined solely by the integrated R/S dictionary data structure, where the state of the art solutions provide $\mathtt{O}(\log |\mathtt{L_B}|)$–time with $\mathtt{o}(|\mathtt{L_B}|)$ extra space.

Lemma 4 (Search on split encoded sequences). *When the number of distinct* $\mathtt{d}$ *bit blocks on input is less than* $2^{\mathtt{d}} - \mathtt{Q}$*, it is possible to efficiently detect all occurrences of a queried pattern* $\mathtt{P}$ *on a split encoded sequence* $\mathtt{B}$ *by maintaining a rank-support on left partition* $\mathtt{L_B}$*.*

Proof. Let $\mathtt{P} = \mathtt{p_1 p_2 \ldots p_m}$ be a binary sequence that we would like to find its all occurrences on the split encoded $\mathtt{B}$. We apply split encoding to $\mathtt{P}$ with the same parameters as $\mathtt{SplitEncode}(\mathtt{P}, \mathtt{d}, \pi, \mathtt{q}) \rightarrow (\mathtt{L_P}, \mathtt{R_P})$. The basic idea of the search is to find the matching positions of $\mathtt{L_P}$ on $\mathtt{L_B}$, and then verifying each match by comparing $\mathtt{R_P}$ with the corresponding position of the candidate on $\mathtt{R_B}$. The first step is to search $\mathtt{L_P}$ on $\mathtt{L_B}$. Notice that not all matching positions are valid left partitions as a match can appear bridging two consecutive blocks. Hence, while scanning for $\mathtt{L_P}$, we should accept only completely matching left partitions on $\mathtt{L_B}$. Since we assume the number of distinct $\mathtt{d}$ bit blocks on input is less than $2^{\mathtt{d}} - \mathtt{Q}$, the permutation π can be constructed over integers $\{\mathtt{Q}, \mathtt{Q}+1, .., 2^{\mathtt{d}}-1\}$, which guarantees that each block has exactly $\mathtt{q}$ set bits on $\mathtt{L_B}$. Therefore, the validity of a matching position can be confirmed by checking whether *i)* the bit preceding the match position on $\mathtt{L_B}$ is a set bit, and *ii)* total number of previous set bits is a multiple of $\mathtt{q}$. For each such valid matching left partition on $\mathtt{L_B}$, we can find the corresponding positions on $\mathtt{R_B}$ to extract its right partition via Algorithm 1. We then extract its corresponding right partition from $\mathtt{R_B}$ and verify whether the constructed pattern matches the query. The performance of the search process greatly depends the efficiency of the filtering phase, where short $\mathtt{P}$ sequences would result in larger set of verification points, and longer ones less. □

4 The Privacy Aspects

Each $\mathtt{d}$–bit block of input $\mathtt{B}$ is represented on $\mathtt{R_B}$ with variable number of bits whose lengths are ranging from $\mathtt{0}$ to $\mathtt{d-1}$. We observe that these variable-length blocks are not prefix-free, and thus, the code–word boundaries on $\mathtt{R_B}$ are uncertain, which can only be determined by using the $\mathtt{L_B}$ partition, assuming the parameters $\mathtt{d}$ and $\mathtt{q}$ are known. For instance, in Figure 1, the code–word boundaries on $\mathtt{R_B} = \mathtt{1110010010}$ can be detected by using the $\mathtt{L_B} = \mathtt{01101100011001}$. One can simply use the random access function Algorithm 1 to detect the bits of a queried ith block on $\mathtt{R_B}$. It is important to address that even after the construction the $\mathtt{d}$–bit block, the permutation function π should be known to reach the original $\mathtt{d}$–bit on $\mathtt{B}$ by finding the inverse permutation mapping of the block. Therefore,in absence of the $\mathtt{L_B}$ partition and the permutation π, it seems hard to reconstruct the original data $\mathtt{B}$ even we assume all other parameters $\mathtt{d}, \mathtt{q}, \mathtt{n}$ are known in advance. We investigate this hardness by analyzing the information released by each partition about the other partition by counting the number of possible bit sequences of length $\mathtt{n}$ that produce the same left or right partition. Ideally, given the left partition, all bit sequences of length $\mathtt{n} - |\mathtt{L_B}|$ should be a valid right partition and decode a distinct $\mathtt{B}$ sequence. Therefore, the correct right partition is indistinguishable in the search space of size $2^{\mathtt{n}-|\mathtt{L_B}|}$. Lemma 5 proves that this is indeed the case.

Lemma 5 (Number of distinct sequences with the same $\mathtt{L_B}$). *Given the left partition $\mathtt{L_B}$ of an input $\mathtt{n}$ bits long sequence $\mathtt{B}$, there are exactly $2^{\mathtt{n}-|\mathtt{L_B}|}$ different bit sequences of length $\mathtt{n}$ that generates the same $\mathtt{L_B}$.*

Proof. Let's assume we have chosen a random binary string $\mathtt{B}$ of length $\mathtt{n}$ and generated the left partition $\mathtt{L_B}$ and right-partition $\mathtt{R_B}$. Since there is no restriction on the possible bits of the right partition, any binary string $\mathtt{X}$ of length $\mathtt{n} - |\mathtt{L_B}|$ will be valid and produce a different $\mathtt{n}$–bit sequence by the decoding process. In other words, the $\mathtt{L}$ partition of all these $2^{\mathtt{n}-|\mathtt{L_B}|}$ sequences will be equal, where the correct right partition is only one of them. □

However, we observe interestingly that the same does not hold for the right partition by Lemma 6, which means the knowledge of right partition narrows the possibilities of left partition. Although this exhibits an information leakage, it still provides a reasonable privacy with large enough search space.

Lemma 6 (Number of distinct sequences with the same $\mathtt{R_B}$). *Given the right partition $\mathtt{R_B}$ of an input $\mathtt{n}$ bits long sequence $\mathtt{B}$, there are less than $2^{\mathtt{n}-|\mathtt{R_B}|}$ different bit sequences of length $\mathtt{n}$ that generates the same $\mathtt{R_B}$. It is also expected that the number of bit sequences sharing the same $\mathtt{R_B}$ is also more than $2^{2\mathtt{qn}/\mathtt{d}}/(2\mathtt{qn}/\mathtt{d}+1)$.*

Proof. There are $\mathtt{n}/\mathtt{d}$ blocks and the length of the $\mathtt{L_B}$ partition is $(\mathtt{n} - |\mathtt{R_B}|)$ bits. Each block has at least zero and at most $\mathtt{q}$ set bits on on $\mathtt{L_B}$, which means the left partition should include certain number of set bits. Therefore, the binary

strings of length $(\mathtt{n} - |\mathtt{R_B}|)$ that include less number of set bits then dictated by the original input are not valid and can not be decoded properly, which means the number of possible $\mathtt{n}$ bit input strings generating the known $\mathtt{R_B}$ is less than $2^{\mathtt{n}-|\mathtt{R_B}|}$.

If we assume all blocks are represented by $\mathtt{q}$ set bits on $\mathtt{L_B}$ (in other words, the number of observed distinct blocks on the input is less than $2^{\mathtt{d}} - \mathtt{Q}$ and we used only blocks with at least $\mathtt{q}$ set bits in the permutation step), then the number of $|\mathtt{L_B}|$ bits long binary strings with $\mathtt{qn/d}$ set bits are all valid left partitions for the given right partition and each of those strings spell a different $\mathtt{B}$. According to Theorem 1, the length of $\mathtt{L_B}$ is expected to be $\approx 2\mathtt{qm}$ bits with $\mathtt{m} = \mathtt{n/d}$. The number of sequences of length $2\mathtt{qm}$ bits with $\mathtt{qm}$ set bits is greater than $2^{2\mathtt{qn/d}}/(2\mathtt{qn/d}+1)$ due to the simple bounds of the central binomial coefficient $4^{\mathtt{x}} = (1+1)^{2\mathtt{x}} = \sum_{\mathtt{k}=0}^{2\mathtt{x}} \binom{2\mathtt{x}}{\mathtt{k}}$. □

Due to Lemma 5, given the left partition $\mathtt{L_B}$ of a split encoded bit sequence, there is no information released about the right partition $\mathtt{R_B}$, since any bit sequence of length $\mathtt{n} - |\mathtt{L_B}|$ is a valid $\mathtt{R_B}$ partition and correctly decodes with the given $\mathtt{L_B}$. On the other hand, due to Lemma 6, once the right partition $\mathtt{R_B}$ is given, not all bit sequences of length $\mathtt{n} - |\mathtt{R_B}|$ are valid left partitions, and thus, the knowledge of $\mathtt{R_B}$ narrows down the search space of $\mathtt{L_B}$. However, this reduction is limited and on large input sequences, still it can provide some privacy. It is noteworthy that the above analyses assume the permutation π is known, where actually keeping that π secret improves security significantly.

5 Splitting Beyond Two Partitions

We observe that recursive application of the splitting mechanism on each partition will generate further splits. As long as the depth of this partitioning increases the partitions gets smaller and it gets harder to reconstruct the data in absence of the one piece. Such a cascaded splitting operation might make sense as in distributed storage systems or recent block-chain style storage like IPFS [3]. We generalize our observation and provide a mechanism such that an input bit stream is divided into $\mathtt{k}$ partitions, where any ratio on the size of the partitions can be supported, e.g., creating 5 partitions that will occupy approximately $4/16, 5/16, 1/16, 3/16, 3/16$ of the input size, respectively.

Lemma 7 (Splitting into multiple pieces). *Let* $\{\mathtt{r_1}, \mathtt{r_2}, \ldots, \mathtt{r_k}\}$ *represent the given ratios such that* $\sum_{\mathtt{i}=1}^{\mathtt{k}} \mathtt{r_i} = 1$ *and* $k > 2$. *An input bit string of length* $\mathtt{n}$ *can be split into* $\mathtt{k}$ *pieces* $\mathtt{p_1}, \mathtt{p_2}, .., \mathtt{p_k}$ *by recursive application of the split coding, where* $|\mathtt{p_i}| \approx \mathtt{r_i} \cdot \mathtt{n}$.

Proof. Assume a *full* binary tree of $\mathtt{s} = 2\mathtt{k} - 1$ elements, which is shown by an array $\mathtt{A}[1..\mathtt{s}]$. In this tree, the leaf nodes represent the final partitions we aim to generate. We place the ratios to the last $\mathtt{k}$ elements of the array, i.e., $\mathtt{A}[\mathtt{s} - \mathtt{i}] = \mathtt{r_{k-i}}$ for $\mathtt{i} = 0..(\mathtt{k} - 1)$. We compute the values of the vacant positions $\mathtt{A}[\mathtt{t}]$ in the array starting from $\mathtt{t} = \mathtt{s} - \mathtt{k}$ down to the root $\mathtt{t} = 1$ by

summing up their corresponding children, where the indices of the children of $\mathtt{A}[\mathtt{t}]$ are $\mathtt{A}[2\mathtt{t}]$ and $\mathtt{A}[2\mathtt{t}+1]$. Notice that the already filled positions are the leaf nodes and thus, the internal nodes can be computed from them. As an example, for $\langle \mathtt{r}_1, \mathtt{r}_2, \mathtt{r}_3, \mathtt{r}_4, \mathtt{r}_5 \rangle = \langle 4/16, 5/16, 1/16, 3/16, 3/16 \rangle$, the array will end up with $\mathtt{A}[1..9] = [\mathbf{1}, \mathbf{10/16}, \mathbf{6/16}, \mathbf{6/16}, 4/16, 5/16, 1/16, 3/16, 3/16]$.

Considering that the root node $\mathtt{A}[1]$ corresponds to the input bit string, we apply the split coding at each internal node with the most appropriate $\mathtt{q}$ and $\mathtt{d}$ parameters to support the ratios mentioned in its children. For instance, in the example case, since $\mathtt{A}[2] = 10/16$ and $\mathtt{A}[3] = 6/16$, choosing $\mathtt{d} = 16$ and $\mathtt{q} = 5$ provides us the left partition to be close to $\mathtt{n} \cdot 10/16$ bits as desired and right partition automatically scales to the remaining $\mathtt{n} \cdot 6/16$. We keep applying the split coding to the internal nodes until we produce all the leaves. In our sample case, for instance, we need to split the $\mathtt{A}[2]$ according to its childrens's ratios as $\mathtt{A}[4] = 6/16$ and $\mathtt{A}[5] = 4/16$, which dictates the left partition $\mathtt{A}[4]$ is desired to be around $6/10$ of its parent $\mathtt{A}[2] = 10/16$. This can be achived by setting $\mathtt{d} = 10$ and $\mathtt{q} = 3$. The bits deposited at the leaf nodes finalize the splitting construction, where reconstruction of the original input requires traversing the tree from the leaves to the root and applying split decode procedure for each internal node. □

6 Conclusions and Further Studies

We have presented a randomized data splitting algorithm that provides well–control on the sizes of the partitions with efficient search and access mechanisms. Reconstruction of the original input is hard without having access to both partitions and the permutation used in the construction.

Such a privacy-preserving data splitting scheme may find application areas in secure and searchable massive data storage on the cloud. For example, the user may keep the permutation secret, and then store the left and right partitions on different remote servers. Due to the privacy aspects, the admins of those remote locations will not be able to extract the content, while the user will still be able to achieve search and retrieval with the mechanisms described in this study. It is also possible to maintain the left partition on-premise and save right partition encrypted on a cloud to make it completely secure. The user can still perform search operations on the encrypted cloud storage by filtering the candidates from the data on-premise, and then verify the candidates by fetching and decrypting them from the cloud. Obviously, such applications require further analysis of some different possible architectures. Similar applications might be considered for other distributed storage schemes including block-chain storage systems[12]. The recursive application of the proposed split coding may be used to privacy-preserving partitioning of the data into some desired proportions. This can also serve as an alternative in secret sharing schemes [2].

References

1. Bard, G.V., Ault, S.V., Courtois, N.T.: Statistics of random permutations and the cryptanalysis of periodic block ciphers. Cryptologia **36**(3), 240–262 (2012)

2. Beimel, A.: Secret-sharing schemes: A survey. In: Coding and Cryptology: Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings 3. pp. 11–46. Springer (2011)
3. Benet, J.: Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014)
4. Camtepe, S., Duda, J., Mahboubi, A., Morawiecki, P., Nepal, S., Pawłowski, M., Pieprzyk, J.: Compcrypt–lightweight ans-based compression and encryption. IEEE Transactions on Information Forensics and Security **16**, 3859–3873 (2021)
5. Du, W., Atallah, M.J.: Secure multi-party computation problems and their applications: a review and open problems. In: Proceedings of the 2001 workshop on New security paradigms. pp. 13–22 (2001)
6. Duda, J.: Asymmetric numeral systems. arXiv preprint arXiv:0902.0271 (2009)
7. Duda, J., Niemiec, M.: Lightweight compression with encryption based on asymmetric numeral systems. arXiv preprint arXiv:1612.04662 (2016)
8. Durstenfeld, R.: Algorithm 235: Random permutation. Commun. ACM **7**(7), 420 (jul 1964). https://doi.org/10.1145/364520.364540, https://doi.org/10.1145/364520.364540
9. Fraenkel, A.S., Klein, S.T.: Complexity aspects of guessing prefix codes. Algorithmica **12**, 409–419 (1994)
10. Gillman, D.W., Mohtashemi, M., Rivest, R.L.: On breaking a huffman code. IEEE Transactions on Information theory **42**(3), 972–976 (1996)
11. Kaaniche, N., Laurent, M.: Data security and privacy preservation in cloud storage environments based on cryptographic mechanisms. Computer Communications **111**, 120–141 (2017)
12. Li, R., Song, T., Mei, B., Li, H., Cheng, X., Sun, L.: Blockchain for large-scale internet of things data storage and protection. IEEE Transactions on Services Computing **12**(5), 762–771 (2018)
13. Li, Y., Gai, K., Qiu, L., Qiu, M., Zhao, H.: Intelligent cryptography approach for secure distributed big data storage in cloud computing. Information Sciences **387**, 103–115 (2017). https://doi.org/https://doi.org/10.1016/j.ins.2016.09.005, https://www.sciencedirect.com/science/article/pii/S0020025516307319
14. Martins, P., Sousa, L., Mariano, A.: A survey on fully homomorphic encryption: An engineering perspective. ACM Computing Surveys (CSUR) **50**(6), 1–33 (2017)
15. Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. In: 2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX). pp. 60–70. SIAM (2007)
16. Plackett, R.L.: The analysis of permutations. Journal of the Royal Statistical Society Series C: Applied Statistics **24**(2), 193–202 (1975)
17. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms (TALG) **3**(4), 43–es (2007)
18. Rubin, F.: Cryptographic aspects of data compression codes. Cryptologia **3**(4), 202–205 (1979)
19. Sharma, P., Jindal, R., Borah, M.D.: Blockchain technology for cloud storage: A systematic literature review. ACM Computing Surveys (CSUR) **53**(4), 1–32 (2020)
20. Vigna, S.: Broadword implementation of rank/select queries. In: Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7. pp. 154–168. Springer (2008)