UDC 004.056.55+004.421.5
DOI: 10.20535/SRIT.2308-8893.2024.3.03

DATA SCRAMBLER KNIGHT TOUR ALGORITHM

V.V. ROMANUKE, S.A. YAREMKO, O.M. KUZMINA, H.A. YEHOSHYNA

Abstract. Nowadays, data scrambling remains a vital technique to protect sensitive information by shuffling it in a way that makes it difficult to decipher or reverse-engineer while still maintaining its usability for legitimate purposes. As manipulating the usability of the scrambled data remains a challenge on the background of risking losing data and getting them re-identified by attackers, scrambling and descrambling should be accomplished faster by not increasing data loss and re-identification risks. A scrambling algorithm must have a linear time complexity, still shuffling the data to minimize the risks further. A promising approach is based on the knight open tour problem, whose solutions appear like a random series of knight positions. Hence, a knight open tour algorithm is formalized, by which the knight seems to move chaotically across the chessboard. The formalization is presented as an indented pseudocode to implement it efficiently, whichever programming language is used. The output is a square matrix representing the knight open tour. Based on the knight tour matrix, data scrambler and descrambler algorithms are presented in the same manner. The algorithms have a linear time complexity. The knight-tour scrambling has a sufficiently low guess probability if an appropriate depth of scrambling is used, where the data is re-scrambled repetitively. The scrambling depth is determined by repetitive application of the chessboard matrix, whose size usually increases as the scrambling is deepened. Compared to the pseudorandom shuffling of the data along with storing the shuffled indices, the knight-tour descrambling key is stored and sent far simpler yet ensures proper data security.

Keywords: data scrambling, knight open tour problem, linear time complexity, guess probability, scrambling depth.

INTRODUCTION

Data scrambling, also known as data obfuscation or data anonymization, is a technique used to protect sensitive information by altering or shuffling it in a way that makes it difficult to decipher or reverse-engineer while still maintaining its usability for legitimate purposes [1; 2]. Data scrambling is an essential component of data protection strategies, helping organizations safeguard sensitive information while still benefiting from its utility [3; 4]. Scrambling is similar to encryption and ciphering, but these techniques differ in their fundamental approaches and purposes [5; 6].

Scrambling is a data protection technique that aims to balance data privacy with usability by partially obscuring data, making it reversible in most cases [7]. Encryption, on the other hand, focuses on data confidentiality by converting it into ciphertext, which is typically not usable without decryption [8]. Ciphering is a broader term that encompasses both scrambling and encryption, as it refers to the process of transforming data to protect its confidentiality or privacy [9]. While scrambling may involve various transformations, such as shuffling, substitution, or masking, to make the data less readable [10], encryption is primarily used to secure data by converting it into a ciphertext using cryptographic algorithms [8; 11; 12]. Encrypted data is typically not usable or meaningful without the corresponding decryption key, as it appears as random ciphertext [5; 6; 9; 11].

The choice between scrambling and encryption depends on the specific use case and requirements regarding data protection and usability. The main purposes of data scrambling are data privacy and compliance. It protects, for example, personal identification data, financial records, proprietary business data, from unauthorized access or disclosure. Data scrambling helps organizations comply with data protection regulations and privacy laws, such as GDPR or HIPAA [13; 14], which require the safeguarding of sensitive data.

One common method of data scrambling is shuffling data using an appropriate algorithm [15]. Knowing this particular algorithm allows descrambling the scrambled data. Other methods are masking that replaces parts of sensitive data with placeholders or pseudonyms, tokenization that replaces sensitive data with tokens or references, which are meaningless without the associated mapping, and data perturbation [16; 17]. The latter is a technique that adds random noise or perturbation to numerical data to protect its privacy while preserving statistical properties.

Data scrambling is successfully used in secure storage, where data at rest is protected from unauthorized access in databases, file systems, or backups. Data sharing is another use case, where organizations share data with third parties for analysis or collaboration without revealing sensitive details. In addition, scrambled data can be used in non-production and test environments to simulate real data without exposing sensitive information [1; 2; 4; 7; 10].

The development of data scrambling includes understanding data sensitivity priority, strengthening encryption, selecting appropriate anonymization techniques based on the specific data and use case, and continuously monitoring and auditing data scrambling processes to ensure their effectiveness and compliance. However, determining a balance between data protection and data usability is challenging [18]. The other two main challenges are data loss and re-identification risks [19; 20]. Thus, improper implementation of data scrambling techniques can lead to data loss or degradation of data quality. Besides, which is the most important and where ones must be the most cautious, attackers may still re-identify individuals or sensitive data if not properly scrambled [21].

While scrambling is frequently used for images due to its visual nature, it is not limited to images and can be applied effectively to text and numerical data as well to protect sensitive information while maintaining data usability [22]. This is often seen in scenarios like redacting personally identifiable information in documents or anonymizing user-generated content in social media moderation [8; 9; 13; 23], academic performance [24], and recommender system profiles [25]. Numerical data, such as financial records, health records, or scientific research data, may also require protection through scrambling techniques [26; 27]. This is essential for compliance with data privacy regulations like GDPR or HIPAA [13; 14].

PROBLEM STATEMENT

As manipulating the usability of the scrambled data remains a challenge on the background of risking to lose data and get them re-identified by attackers, scrambling and descrambling should be accomplished faster by not increasing data loss and re-identification risks. A scrambling algorithm must be of a linear time complexity still shuffling the data so that to further minimize the risks. A promising approach is based on the knight open tour problem [28; 29] whose solutions appear like a random series of knight positions. The goal of the research is to apply this property of the solutions to data scrambling. For achieving the goal, the following five tasks are to be fulfilled:

1. To formalize a knight open tour algorithm, by which the knight is seemed to move chaotically across the chessboard. The formalization is to be presented as an indented pseudocode to efficiently implement it, whichever programming language is used. The output is a square matrix representing the knight open tour.
2. To algorithmize a data scrambler and descrambler based on the knight tour matrix. Both the algorithms must be given as indented pseudocodes.
3. To estimate the time complexity of the algorithms. In addition, to compare their performance to other approach of scrambling by shuffling the data, including the probability of illegitimately descrambling by attackers.
4. To discuss the significance and practical applicability of the suggested knight open tour algorithm. The proper contribution to the field of data scrambling should be emphasized.
5. To conclude on the suggestion and findings along with mentioning a possibility to extend and advance the research.

KNIGHT TOUR MATRIX

The directions the knight can move on the chessboard are completely described by the horizontal and vertical sets

$$S_{\text{hor}} = \{s_l^{(\text{hor})}\} = \{-2, -1, 1, 2, -2, -1, 1, 2\} \tag{1}$$

and

$$S_{\text{vert}} = \{s_l^{(\text{vert})}\} = \{1, 2, 2, 1, -1, -2, -2, -1\}\,. \tag{2}$$

Sets (1) and (2) are such that

$$s_l^{(\text{hor})} = s_{l+4}^{(\text{hor})} \quad \forall\, l = \overline{1, 4}\,,$$

and

$$s_l^{(\text{vert})} = -s_{l+4}^{(\text{vert})} \quad \forall\, l = \overline{1, 4}\,.$$

If the knight starts its open tour at horizontal position x and vertical position y on a chessboard of size $M \times M$, the knight open tour algorithm finds a sequence of the remaining $M^2 - 1$ chessboard positions that constitute the tour. The sequence is written by the set of integers from 1 to M^2, where 1 corresponds to the starting position of the knight. These integers are put on the chessboard, forming thus an $M \times M$ matrix

$$\mathbf{B}_M = K(M, x, y) = [b_{rt}]_{M\times M}\,, \tag{3}$$

where $b_{rt} \in \{\overline{1, M^2}\}$ and $K(M, x, y)$ is the algorithm mapping the size of the chessboard and the knight starting position into matrix $\mathbf{B}_M$ by (3).

While the knight tour problem is NP-hard in general [28], there is a number of heuristic algorithms that allow finding a solution in linear time — that is, in a time amount proportional to number M^2. One of such heuristics is the Warnsdorff's rule [30; 31]. This rule finds a single solution. According to the Warnsdorff's rule, the knight is moved so that it always proceeds to the position from which the knight will have the fewest onward moves. The priority queue of available neighbors is stored as a set

$$Q = \{[q_{w1} \quad q_{w2}]\}_{w=1}^{W}, \ W \in \mathrm{N} \cup \{0\} . \tag{4}$$

The algorithm is presented as an indented pseudocode (Algorithm 1), where set (4) dynamically changes its size inside the outer loop.

Algorithm 1. The Warnsdorff's rule indented pseudocode

```
for $k = 1$ with step 1 to $k = M^2$ do
    $b_{yx} = k$, $Q = \varnothing$
    for $l = 1$ with step 1 to $l = M$ do
        $m_{\text{hor}} = x + s_l^{(\text{hor})}$, $m_{\text{vert}} = y + s_l^{(\text{vert})}$
        if $m_{\text{hor}} \geqslant 1$ and $m_{\text{hor}} < M + 1$ and $m_{\text{vert}} \geqslant 1$ and $m_{\text{vert}} < M + 1$
            if $b_{m_{\text{vert}} m_{\text{hor}}} = 0$
                $c = 0$
                for $u = 1$ with step 1 to $u = M$ do
                    $g_{\text{hor}} = m_{\text{hor}} + s_u^{(\text{hor})}$, $g_{\text{vert}} = m_{\text{vert}} + s_u^{(\text{vert})}$
                    if $g_{\text{hor}} \geqslant 1$ and $g_{\text{hor}} < M + 1$ and $g_{\text{vert}} \geqslant 1$ and
                    $g_{\text{vert}} < M + 1$
                        if $b_{g_{\text{vert}} g_{\text{hor}}} = 0$ then $c^{(\text{obs})} = c$,
                        $c = c^{(\text{obs})} + 1$
                if $Q = \varnothing$ then $Q = \{[c \quad l]\}$
                else
                    if $q_{w1} > c$ then $Q^{(\text{obs})} = Q$, $Q = \{[c \quad l], Q^{(\text{obs})}\}$
                    else
                        if $q_{w1} = c$
                            if $q_{w2} > l$ then $Q^{(\text{obs})} = Q$,
                            $Q = \{[c \quad l], Q^{(\text{obs})}\}$
                            else $Q^{(\text{obs})} = Q$,
                            $Q = \{Q^{(\text{obs})}, [c \quad l]\}$
                        if $q_{w1} < c$ then $Q^{(\text{obs})} = Q$,
                        $Q = \{Q^{(\text{obs})}, [c \quad l]\}$
    if $Q \neq \varnothing$ then $l^* = q_{12}$, $x^{(\text{obs})} = x$, $x = x^{(\text{obs})} + s_{l^*}^{(\text{hor})}$, $y^{(\text{obs})} = y$,
    $y = y^{(\text{obs})} + s_{l^*}^{(\text{vert})}$
    else break
```

The knight open tour resulting from Algorithm 1 mostly seems to be rather chaotic. An example of the tour for 16×16 chessboard is shown in Fig. 1, where the entries of the knight tour matrix $\mathbf{B}_{16}$ are also given (odd and even numbers differ in their color). Some traceries are still noticeable, though. Especially near the margins — where the knight has fewer possibilities to move onward.

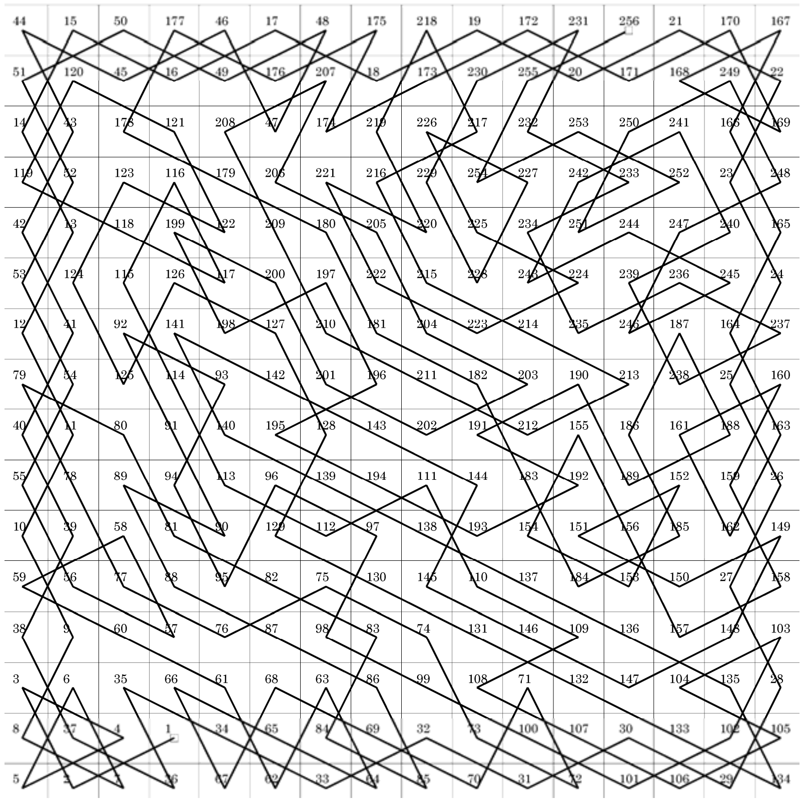

Fig. 1. A knight open tour built by the Warnsdorff's rule algorithm for 16×16 chessboard

When we have two-dimensional data of N points in each dimension, it can be formally presented as a square matrix

$$\mathbf{D} = [d_{ij}]_{N \times N} , \tag{5}$$

where d_{ij} is either real or complex number. In particular, if $M = N$, the data in matrix (5) can be scrambled using the pattern of the knight tour matrix (3) by just shuffling the entries of $\mathbf{D}$ in accordance with $\mathbf{B}_M$. When $M \neq N$, we can shuffle not an entry but an $a \times a$ square of a^2 entries, where $a = \frac{N}{M}$ and it obviously must be integer. This is the core of the algorithm for data scrambling and de-scrambling based on the knight tour matrix (3).

DATA SCRAMBLER AND DESCRAMBLER

A data scrambler is an operator that maps data matrix (5) using the knight tour matrix (3) into an $N \times N$ matrix

$$\mathbf{H} = F(\mathbf{D}, \mathbf{B}_M) = [h_{ij}]_{N \times N} . \tag{6}$$

Operator $F(\mathbf{D}, \mathbf{B}_M)$ in (6) is realized by a data scrambler algorithm (Algorithm 2), which, using matrices $\mathbf{D}$ and $\mathbf{B}_M$, dynamically builds an $M^2 \times 1$ array

$$\mathbf{Q}_{\text{square}} = [q_{k1}^{(\text{square})} \quad q_{k2}^{(\text{square})}]_{M^2 \times 1} \tag{7}$$

containing coordinates of successive positions to shuffle.

Algorithm 2. A data scrambler by (6) using the knight open tour matrix (3)

```
for $l = 1$ with step 1 to $l = M$ do
    for $u = 1$ with step 1 to $u = M$ do
        $k = (l-1) \cdot M + u$, $q_{k1}^{(\text{square})} = l$, $q_{k2}^{(\text{square})} = u$
for $l = 1$ with step 1 to $l = M$ do
    for $u = 1$ with step 1 to $u = M$ do
        $k = b_{lu}$, $l_{\text{scr}} = q_{k1}^{(\text{square})}$, $u_{\text{scr}} = q_{k2}^{(\text{square})}$
        $h_{i^* j^*} = d_{i^{**} j^{**}}$ for $i^* = (l-1) \cdot a + m$, $j^* = (u-1) \cdot a + m$, $m = \overline{1, a}$
        $i^{**} = (l_{\text{scr}} - 1) \cdot a + m$, $j^{**} = (u_{\text{scr}} - 1) \cdot a + m$
```

It is worth noting that the data can be a three-dimensional matrix as well. Then the data scrambler algorithm processes the third-dimension layers in parallel, using, for instance, the data vectorization approach [32]. Thus, the color image in Fig. 2 is scrambled by using an ordinary chessboard matrix $\mathbf{B}_8$ with its

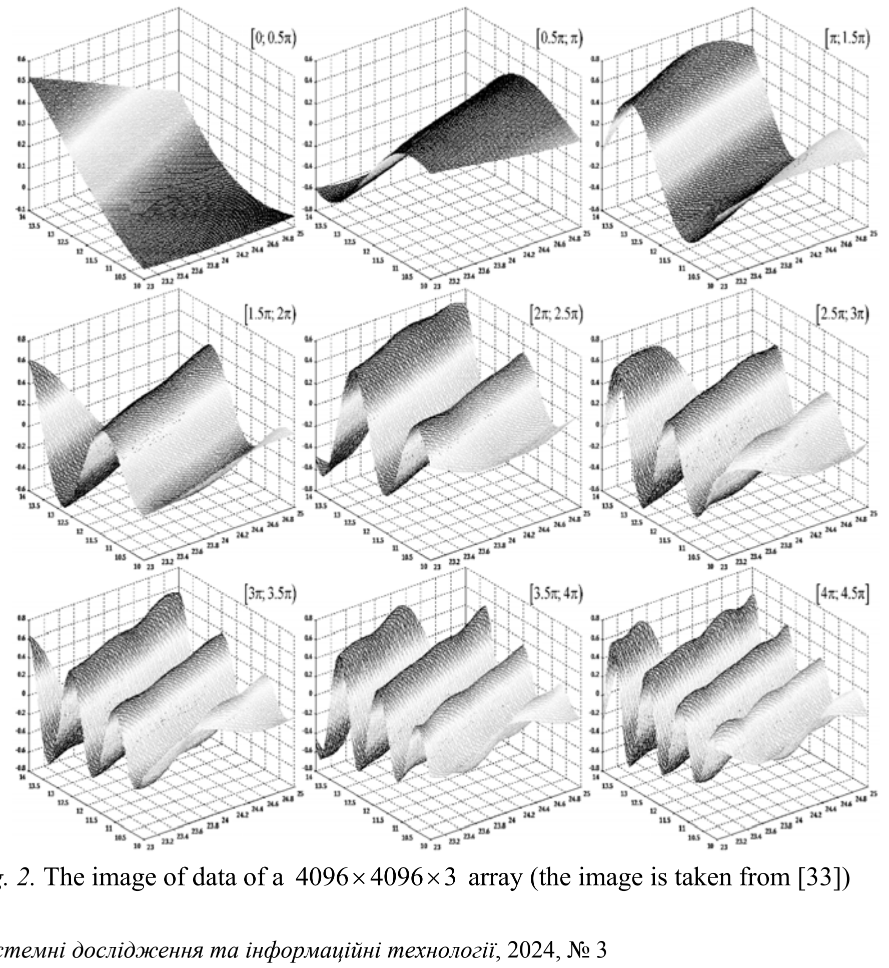

Fig. 2. The image of data of a $4096 \times 4096 \times 3$ array (the image is taken from [33])

starting position $\{7, 6\}$ ($x = 6$, $y = 7$). The scrambled image is shown in Fig. 3. In this scientific-data-like example, the scrambling result may not seem that convincing, but the mosaic in Fig. 3 is nonetheless pretty hard to assemble it back to the original image in Fig. 2. Nevertheless, many mosaic squares contain parts of consistent information. This is due to every mosaic square is a 512×512 color image.

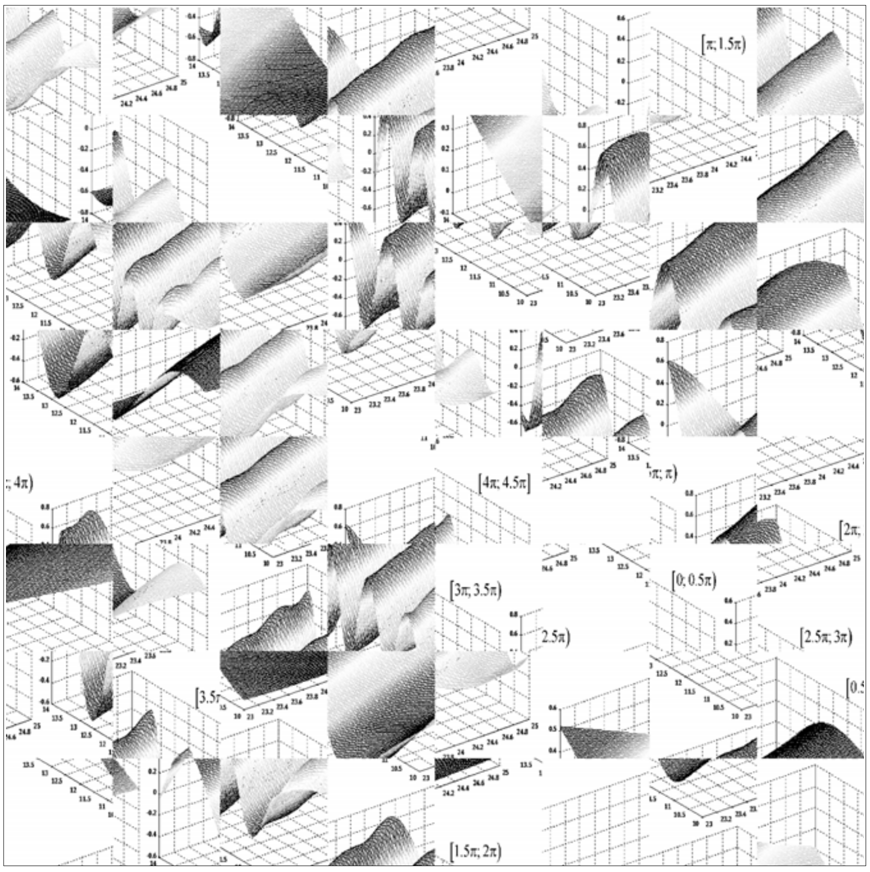

Fig. 3. The scrambled image in Fig. 2 by using $\mathbf{B}_8$ with its starting position $\{7, 6\}$

To decrease the data readability, the size of the chessboard should be increased. On the other hand, the scrambler can be applied once again to the scrambled data. Thus, Fig. 4 presents a result of scrambling the scrambled image in Fig. 3 by using matrix $\mathbf{B}_{16}$ with its starting position $\{7, 12\}$ ($x = 12$, $y = 7$). Now the readability of the image parts is lower, although many parts and their local information are still distinguishable. However, assembling the mosaic in Fig. 4 back to the original image is much more difficult compared to the mosaic in Fig. 3.

Indeed, the mapping of data (5) by (6) and (3) is a very simple approach whose guess probability is M^{-2}. To go deeper in scrambling, consider its depth Θ, where $\Theta \in \mathrm{N} \bigcup \{0\}$, and an additional sequence of integers $\{M_\gamma\}_{\gamma=1}^{\Theta}$, where usually

$$M < M_\gamma < M_{\gamma+1} \text{ by } \gamma = \overline{1, \Theta - 1}\,. \tag{8}$$

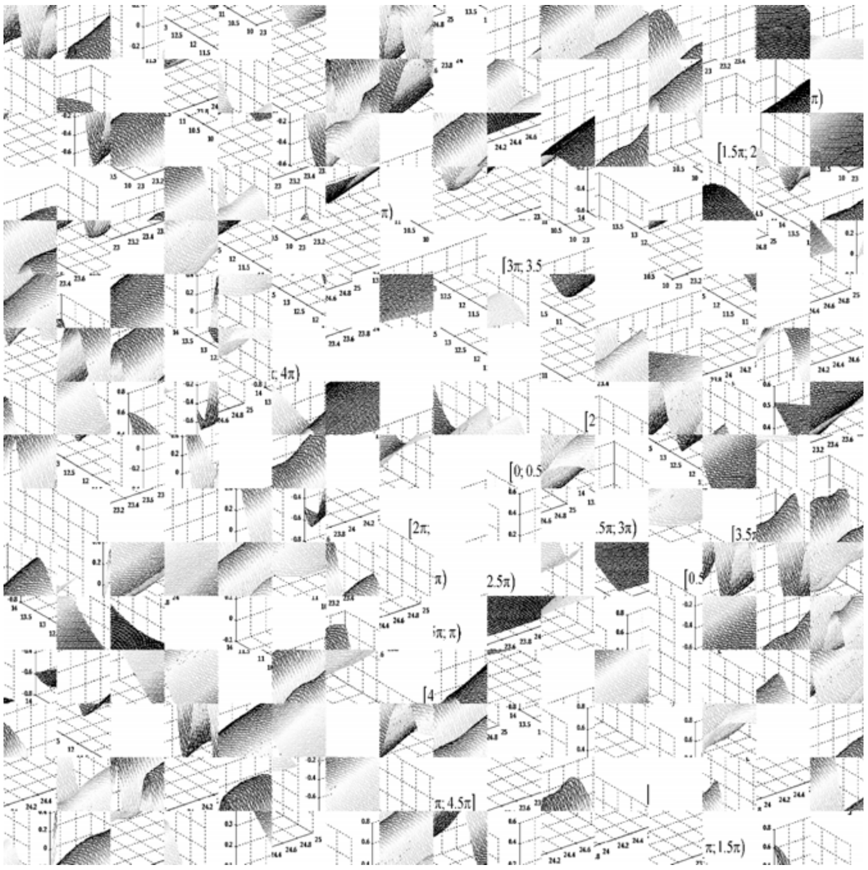

Fig. 4. The scrambled image in Fig. 3 by using $\mathbf{B}_{16}$ with its starting position $\{7,12\}$

Then the scrambling of depth Θ is recursively fulfilled as

$$\mathbf{H}_{\gamma+1} = F(\mathbf{H}_{\gamma}, \mathbf{B}_{M_{\gamma}}) \text{ for } \gamma = \overline{1, \Theta} \tag{9}$$

by $\mathbf{H}_1 = \mathbf{H}$. The case $\Theta = 0$ is the simplest possible scrambling here. Overall, there are $\Theta + 1$ transformations by operator F. They result in a set $\{\mathbf{H}_{\gamma}\}_{\gamma=1}^{\Theta+1}$ of scrambled data matrices. The last matrix in this set, $\mathbf{H}_{\Theta+1}$, is the final result of the data scrambling of depth Θ. The guess probability herein becomes equal to

$$\left(M^2 \cdot \prod_{\gamma=1}^{\Theta} M_{\gamma}^2 \right)^{-1} . \tag{10}$$

So, the image in Fig. 4 is the scrambling result of depth 1, and its guess probability is

$$(M^2 \cdot M_1^2)^{-1} = (8^2 \cdot 16^2)^{-1} = 2^{-14} = 0.00006103515625 .$$

Going deeper by using matrix $\mathbf{B}_{32}$ with its starting position $\{6, 21\}$ ($x = 21$, $y = 6$) further strengthens the encryption (Fig. 5) decreasing the guess probability by 1024 times.

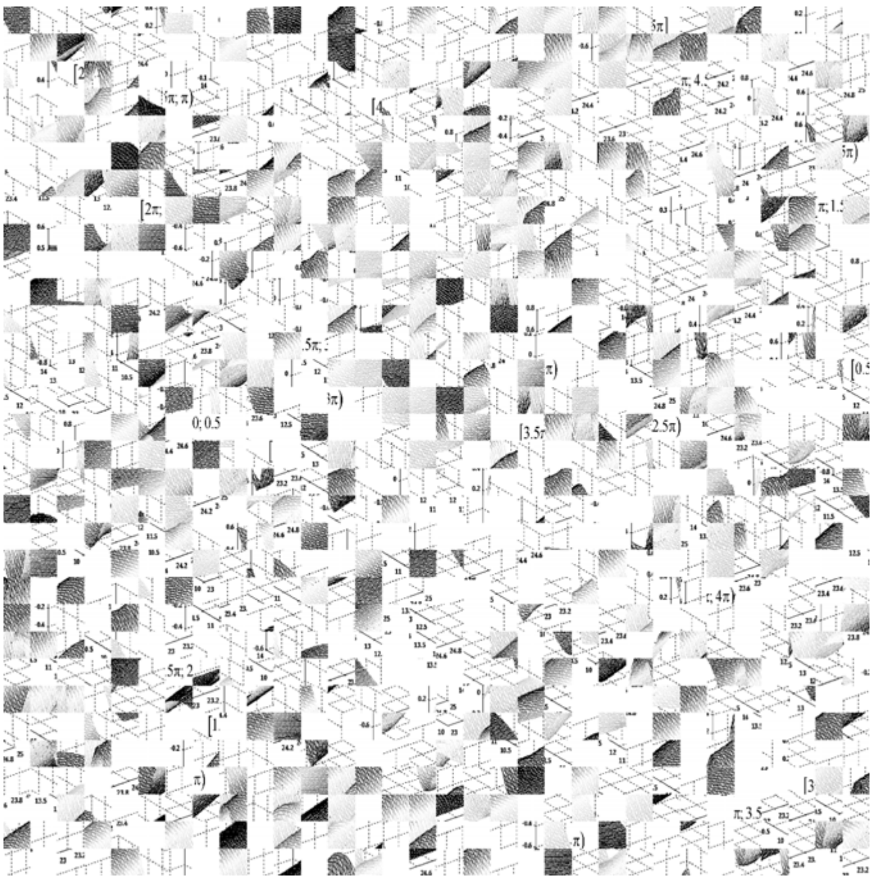

Fig. 5. The scrambled image in Fig. 4 (depth 2) by using $\mathbf{B}_{32}$ with its starting position $\{6, 21\}$, where the guess probability is $2^{-24} = 0.000000059604644775390625$

Herein, while obeying inequality (8), conditions

$$M_{\gamma+1} = 2 \cdot M_{\gamma} \text{ by } \gamma = \overline{1, \Theta - 1} \text{ and } M_1 = 2 \cdot M \tag{11}$$

are followed. This is convenient to establish the guess probability such as it is desired to be. Scrambling to depth 3 (Fig. 6), where matrix $\mathbf{B}_{64}$ has its starting position $\{3, 46\}$ ($x = 46$, $y = 3$), makes the guess practically impossible within a reasonable amount of time (unless the latest advanced computational techniques are applied, like, e. g., quantum computing):

$$(M^2 \cdot M_1^2 \cdot M_2^2 \cdot M_3^2)^{-1} = (8^2 \cdot 16^2 \cdot 32^2 \cdot 64^2)^{-1} = 2^{-36} =$$

$$= \frac{1}{68719476736} < 1.5 \cdot 10^{-11}.$$

Besides, while the usability is maintained, the data in Fig. 6 is not readable even partially — every mosaic square is just a 64×64 color image.

Fig. 7 shows that at depth 4 for this particular example the scrambled image is perceived as noise. Every mosaic square is just a 32×32 color image. The guess probability is 16384 times lower than that for the scrambled image in Fig. 6:

$$(M^2 \cdot M_1^2 \cdot M_2^2 \cdot M_3^2 \cdot M_4^2)^{-1} = (8^2 \cdot 16^2 \cdot 32^2 \cdot 64^2 \cdot 128^2)^{-1} = 2^{-50} =$$

$$= \frac{1}{1125899906842624} < 8.9 \cdot 10^{-16}.$$

The data readability is nearly at naught. Therefore, Fig. 7 is the final result of scrambling the data in Fig. 2.

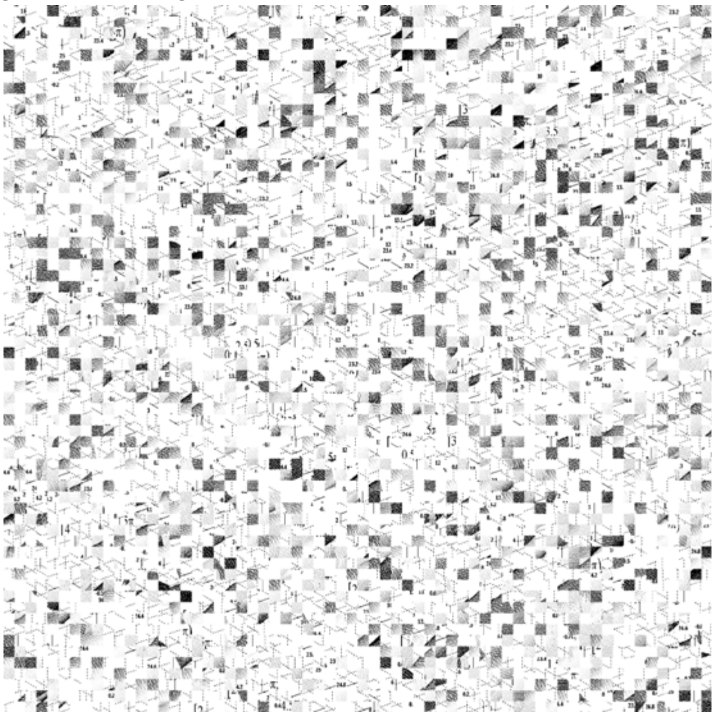

Fig. 6. The scrambled image in Fig. 5 (depth 3) by using $\mathbf{B}_{64}$ with its starting position $\{3, 46\}$, where the guess probability is 2^{-36}

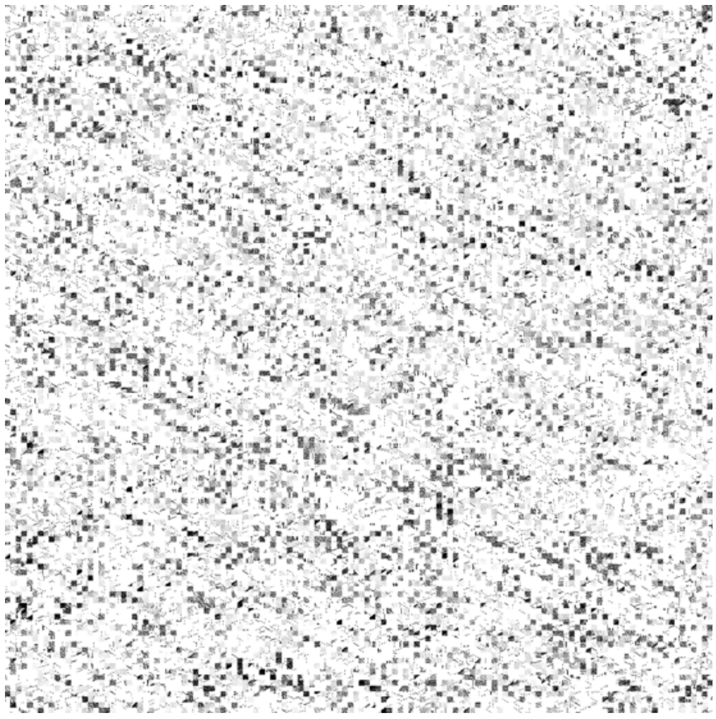

Fig. 7. The scrambled image in Fig. 6 (depth 4) by using $\mathbf{B}_{128}$ with its starting position $\{6, 36\}$, where the guess probability is 2^{-50}

A data descrambler is an operator that maps scrambled data matrix (6) using the knight tour matrix (3) back into matrix (5):

$$\mathbf{D} = G(\mathbf{H}, \mathbf{B}_M). \qquad (12)$$

Operator $G(\mathbf{H}, \mathbf{B}_M)$ in (12) is realized by a data descrambler algorithm (Algorithm 3), which, using matrices $\mathbf{H}$ and $\mathbf{B}_M$, dynamically builds array (7).

Algorithm 3. A data descrambler by (7) using the knight open tour matrix (3)

```
for $l = 1$ with step 1 to $l = M$ do
    for $u = 1$ with step 1 to $u = M$ do
        $k = (l-1) \cdot M + u$, $q_{k1}^{(\text{square})} = l$, $q_{k2}^{(\text{square})} = u$
for $l = 1$ with step 1 to $l = M$ do
    for $u = 1$ with step 1 to $u = M$ do
        $k = b_{lu}$, $l_{\text{scr}} = q_{k1}^{(\text{square})}$, $u_{\text{scr}} = q_{k2}^{(\text{square})}$
        $d_{i^{**} j^{**}} = h_{i^* j^*}$ for $i^* = (l-1) \cdot a + m$, $j^* = (u-1) \cdot a + m$, $m = \overline{1, a}$
        $i^{**} = (l_{\text{scr}} - 1) \cdot a + m$, $j^{**} = (u_{\text{scr}} - 1) \cdot a + m$
```

The descrambling of depth Θ is recursively fulfilled as

$$\mathbf{H}_{\Theta-\gamma+1} = G(\mathbf{H}_{\Theta-\gamma+2}, \mathbf{B}_{M_{\Theta-\gamma+1}}) \text{ for } \gamma = \overline{1, \Theta}. \qquad (13)$$

Obviously, there are $\Theta + 1$ transformations by operator G restoring the backward set of scrambled data matrices $\{\mathbf{H}_{\Theta-\gamma+1}\}_{\gamma=1}^{\Theta}$. The last matrix in this set, $\mathbf{H}_1 = \mathbf{H}$, is put into operator (12) and then the final result of the data descrambling is obtained.

Henceforth, the data scrambler by (6) and (9) requires knowing $\Theta + 1$ chessboard matrices

$$\{\mathbf{B}_M, \{\mathbf{B}_{M_\gamma}\}_{\gamma=1}^{\Theta}\} \qquad (14)$$

whose sizes obey inequality (8) or, in particular, the doubling by (11). This ensures the guess probability equal to (10) or, if the doubling by (11) is used, the guess probability equal to

$$\left(M^2 \cdot (2M)^2 \cdot \prod_{\gamma=2}^{\Theta} (2M_{\gamma-1})^2 \right)^{-1} = \left(M^2 \cdot 4M^2 \cdot \prod_{\gamma=2}^{\Theta} 4M_{\gamma-1}^2 \right)^{-1} =$$

$$= \left(\sqrt{4^{\Theta(\Theta+1)}} M^{2 \cdot (\Theta+1)} \right)^{-1}. \qquad (15)$$

Each of matrices (14) is defined by its size and the starting position of the knight. Therefore, the data scrambler is defined by the sizes and starting positions

$$\{M, \{M_\gamma\}_{\gamma=1}^{\Theta}\} \text{ and } \{\{y, x\}, \{y_\gamma, x_\gamma\}_{\gamma=1}^{\Theta}\}, \qquad (16)$$

respectively. Consequently, the data descrambler by (12) and (13) is defined by sets (16). Knowing (16) allows descrambling data in matrix $\mathbf{H}_{\Theta+1}$ without any data losses.

KNIGHT-TOUR SCRAMBLING VERSUS SHUFFLER SCRAMBLING

It is obvious that, despite a linear time complexity, too deep scrambling may take significant amounts of time. However, the deep scrambling is really needful only for decreasing the guess probability as much as possible. In other situations, the data scrambler can be applied just once to produce a scrambled data matrix of sufficiently low readability. For example, the data image in Fig. 2 scrambled with a matrix $\mathbf{B}_{128}$ looks blurred (Fig. 8) and its scientific data is not readable almost much as the image in Fig. 7, although the guess probability is just 2^{-14} (relatively, it is pretty high).

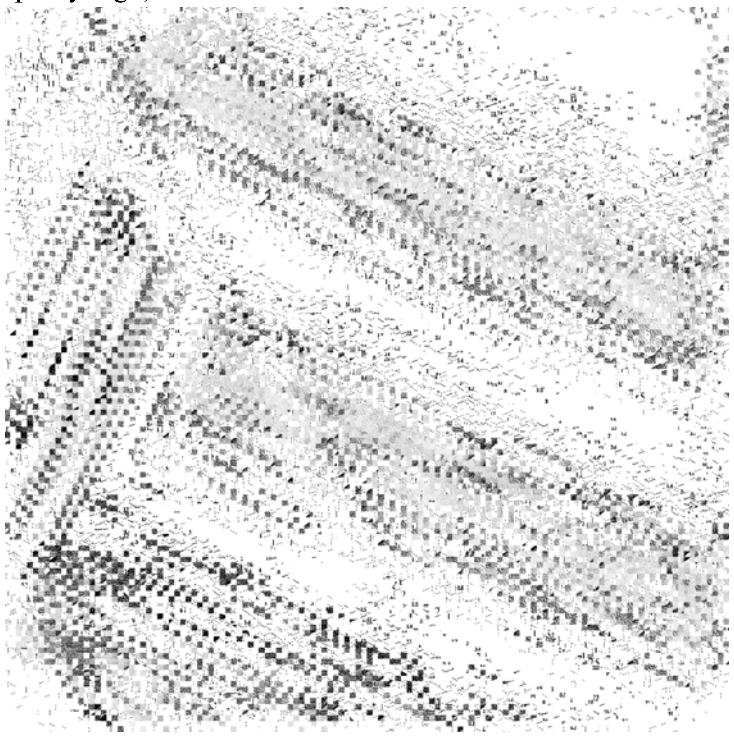

Fig. 8. The scrambled image in Fig. 2 by using the chessboard matrix of size 128 (the zero depth of scrambling, $\Theta = 0$), where the guess probability is the same as for the scrambled image in Fig. 4 with the scrambling depth 1

Could the same effect be when, instead of the data scrambler knight tour algorithm, a more random and less sophisticated approach is used? For instance, such an approach is the pseudorandom shuffling of the data along with storing the shuffled indices. The shuffler scrambling is defined just by an $N \times N$ matrix

$$\mathbf{S} = [\sigma_{ij}]_{N \times N} \tag{17}$$

of randomly permutated unique N^2 integers from 1 to N^2. Then the shuffler scrambling and descrambling algorithms are those Algorithms 2 and 3, respectively, where only $a = 1$ and matrix (17) is used instead of matrix (3). The guess probability in this case is N^{-2} as the shuffler algorithm is presumed to be known and it can be defined by the position of an integer value from 1 to N^2.

Obviously, if the knight-tour scrambling is of depth 0, then the shuffler scrambling results in a lower guess probability unless $M = N$:

$$M^{-2} > N^{-2} \text{ for } M < N .$$

However, as the scrambling is developed deeper, the guess probability may significantly drop, as it has been demonstrated above by Figs. 4–7.

Consider a range of the data set size as follows:

$$N \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8\} = \{8, 16, 32, 64, 128, 256\} . \tag{18}$$

Then data matrix (5) with integers

$$d_{ij} \in [0; 255] \subset \mathbf{Z}$$

is re-generated for 100 repetitions for each of the six values of N in (18). The depth of scrambling is

$$\Theta = \log_2 N - 3 .$$

So, the knight-tour data scrambler is applied as deep as possible, i. e.

$$\{M, \{M_\gamma\}_{\gamma=1}^{\Theta}\} = \{8, \{2^{\gamma+3}\}_{\gamma=1}^{\Theta}\} = \{8, \{2^{\gamma+3}\}_{\gamma=1}^{\Theta-1}, N\} .$$

That is, the knight tour matrix $\mathbf{B}_{M_\Theta} = \mathbf{B}_N$, whichever N is. This is expectedly the worst-case scenario with respect to the computation time of the knight-tour data scrambler compared to the shuffler. Table 1 presents statistics after averaging over the 100 repetitions of the knight-tour data scrambler versus shuffler. The similarity index ratio is calculated as the ratio of the averaged similarity rate by the knight-tour data scrambler to the averaged similarity rate by the shuffler, where the averaged similarity rate is calculated as the element-wise number of coincidences in matrices $\mathbf{D}$ and the scrambled data matrix divided by N^2. The knight-tour-scrambling guess probability herein is calculated by (15). The statistics remain almost the same for any other 100-repetition generations. Although the knight-tour-to-shuffler time ratios reveal some favorability of the shuffler computation time, the shuffler is a far less reliable scrambler due to its guess probability is too high compared to the knight-tour-scrambling guess probability. Besides, the knight-tour-to-shuffler similarity index ratio confirms that the knight-tour data scrambler produces a scrambled object which resembles the original less than a scrambled object by the shuffler resembles it.

Table 1. Knight-tour scrambling versus shuffler scrambling

N	**Knight-tour-to-shuffler scrambling time ratio**	**Knight-tour-to-shuffler descrambling time ratio**	**Similarity index ratio**	**Knight-tour-scrambling guess probability**	**Shuffler-scrambling guess probability**
8	0.5797	1.1134	0.2308	$8^{-2} = 2^{-6} = 0.015625$	0.015625
16	1.3375	1.3913	1.0055	$8^{-2} \cdot 16^{-2} = 2^{-14} = = 6.103515625 \cdot 10^{-5}$	0.00390625
32	1.3805	1.416	0.7135	$8^{-2} \cdot 16^{-2} \cdot 32^{-2} = = 2^{-24} < 6 \cdot 10^{-8}$	$9.765625 \cdot 10^{-4}$
64	1.3392	1.3928	0.9337	$8^{-2} \cdot 16^{-2} \cdot 32^{-2} \cdot 64^{-2} = = 2^{-36} < 1.5 \cdot 10^{-11}$	$2.44140625 \cdot 10^{-4}$
128	1.2306	1.2232	0.995	$2^{-50} < 8.9 \cdot 10^{-16}$	$6.103515625 \cdot 10^{-5}$
256	1.1258	1.1157	1.0079	$2^{-66} < 1.4 \cdot 10^{-20}$	$1.52587890625 \cdot 10^{-5}$

Consider another series of 100 simulations for the knight-tour data scrambler. Let

$$N \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\} = \{8, 16, 32, 64, 128, 256, 512, 1024\} \quad (19)$$

and the remaining simulation parameters be the same. Let the depth be varied from 0 to 5. Then, for each repetition, there is an instance of some size in (19) and $\Theta \in \{\overline{0, 5}\}$. The averaged interrelation between scrambling and descrambling computation times is shown in Table 2. The interrelation varies within 20 % on average for any other 100-repetition generations. The knight-tour data descrambler seemingly operates faster owing to the memory allocation and code compilation that are done following the scrambling.

Table 2. Knight-tour scrambling-to-descrambling time ratio

N	**Depth of scrambling**					
	0	1	2	3	4	5
8	0.9248					
16	1.0638	1.0302				
32	1.0756	1.0231	1.056			
64	1.0765	1.0578	1.039	1.0206		
128	1.151	1.179	1.1399	1.0888	1.0859	
256	1.1518	1.361	1.3882	1.3129	1.102	1.0409
512	1.2119	1.5472	1.6715	1.732	1.42	1.1119
1024	1.1883	1.6135	1.8532	2.0819	1.9851	1.3881

Obviously, as the size of the data matrix increases, the computation time grows. Table 3 shows the computation time relative growth with respect to the computation time taken to scramble with a chessboard matrix of size N. At the first glance, there is a quadratic time complexity with respect to size N along each column, but the real size of the algorithm input is N^2, so Table 3 confirms a linear time complexity of Algorithm 2. So does Table 4 showing the computation time relative growth with respect to the computation time taken to descramble with a chessboard matrix of size N. However, it is worth noting that Algorithm 3 being of a linear time complexity seems to be faster (this is clearly seen when Table 4 is cell-by-cell compared to Table 3). This effect has been already explained above — the data descrambler by Algorithm 3 virtually utilizes the memory pre-allocation and code pre-compilation, done for Algorithm 2 in this case.

Table 3. Knight-tour scrambling time relative growth as N increases

N	**Depth of scrambling**					
	0	1	2	3	4	5
8	1					
16	0.5562	1				
32	0.4834	1.0351	1			
64	0.5661	1.1575	1.1084	1		
128	1.0355	1.832	1.5465	1.2523	1	
256	2.7697	4.2452	2.86	1.8061	1.1855	1
512	9.1934	12.8017	7.7231	3.8901	1.8098	1.1376
1024	35.2239	42.9207	23.9562	11.245	4.1936	1.5951

Table 4. Knight-tour descrambling time relative growth as N increases

N	**Depth of scrambling**					
	0	1	2	3	4	5
8	1					
16	0.4835	1				
32	0.4156	1.0423	1			
64	0.4863	1.1273	1.1265	1		
128	0.832	1.6007	1.4327	1.174	1	
256	2.2238	3.2135	2.1756	1.4041	1.1681	1
512	7.0152	8.524	4.8793	2.2924	1.3839	1.0649
1024	27.412	27.4043	13.6509	5.5127	2.2939	1.1961

Another important property is how the computation time grows as the scrambling depth is increased. Table 5 shows the computation time relative growth with respect to the scrambling depth. It appears that the data scrambler has a quadratic time complexity with respect to the scrambling depth. Roughly the same time complexity is observed for the data descrambler (Table 6). This is particularly explained with that the chessboard matrix size is increased twice along a dimension, i. e. it is increased fourfold if to count its entries.

Table 5. Knight-tour scrambling time relative growth as the depth is increased

N	**Depth of scrambling**					
	0	1	2	3	4	5
8	1					
16	1	3.57				
32	1	4.2513	15.4973			
64	1	4.0595	14.6675	59.8536		
128	1	3.5128	11.1889	40.9824	202.9467	
256	1	3.0433	7.7362	22.0973	89.947	782.9858
512	1	2.7648	6.2936	14.3384	41.3688	268.3466
1024	1	2.4194	5.0952	10.8178	25.0193	98.2041

Table 6. Knight-tour descrambling time relative growth as the depth is increased

N	Depth of scrambling					
	0	1	2	3	4	5
8	1					
16	1	3.6864				
32	1	4.4696	15.785			
64	1	4.1312	15.1969	63.1311		
128	1	3.4293	11.2978	43.3244	215.123	
256	1	2.5756	6.4187	19.3856	94.0116	866.4042
512	1	2.1657	4.5632	10.0329	35.3074	292.477
1024	1	1.7818	3.2672	6.1745	14.9772	84.0675

The knight-tour scrambling similarity index showing the unit-normalized part of the number of coincidences in the data matrix and the scrambled data matrix appears to be quite stable regardless of the data set size and the scrambling depth (Table 7). The scrambling similarity index scarcely exceeds 0.5 %. The average similarity index varies within 20 % on for any other 100-repetition generations. A maximum variation is spotted at a 53 % rate.

Table 7. Knight-tour scrambling similarity index

N	**Depth of scrambling**					
	0	1	2	3	4	5
8	0.0033					
16	0.0037	0.0043				
32	0.0041	0.0041	0.0051			
64	0.0038	0.0038	0.0048	0.0041		
128	0.0039	0.0039	0.0048	0.0041	0.004	
256	0.0039	0.0039	0.0048	0.0042	0.0039	0.004
512	0.0039	0.0039	0.0049	0.0041	0.0039	0.0039
1024	0.0039	0.0039	0.0049	0.0041	0.0039	0.0039

The final remark of the knight-tour scrambling versus shuffler scrambling relates to the descrambling key. While the knight-tour data descrambler must hold the $\Theta+1$ sizes and starting positions (16), its memory size requirement is very low. In the double precision format, it is just

$$8\cdot 3\cdot(\Theta+1)=24\cdot(\Theta+1) \text{ bytes,}$$

and it is

$$4\cdot 3\cdot(\Theta+1)=12\cdot(\Theta+1) \text{ bytes}$$

in the single precision format. In practice, nevertheless, values (16) are integers (strictly speaking, naturals), so they are written with the unsigned 16-bit precision at most, when they are allowed to vary between 1 and 65535. In this case the data descrambler key occupies just

$$2\cdot 3\cdot(\Theta+1)=6\cdot(\Theta+1) \text{ bytes.}$$

If it is known beforehand that

$$\max\{M, \max\{M_\gamma\}_{\gamma=1}^{\Theta}, \max\{y, x\}, \max\{y_\gamma, x_\gamma\}_{\gamma=1}^{\Theta}\}<256$$

then the data descrambler key occupies half of the latter by being written with the unsigned 8-bit precision — it is $3\cdot(\Theta+1)$ bytes. The shuffler descrambling, on the other hand, needs to know matrix (17) of unique N^2 integers from 1 to N^2, which would occupy $8N^2$ or $4N^2$ bytes if the double or single precision format was used, respectively. Inasmuch as using the unsigned 8-bit precision is unlikely here, then either $2N^2$ or $4N^2$ bytes are occupied depending on whether the unsigned 16-bit or 32-bit precision is respectively used. Therefore, the memory occupation by the shuffler descrambling key may seriously affect the data transfer rate. For instance, if 1024×1024 images are shuffling-scrambled and transferred, then an additional amount of 4 megabytes of information should be transferred along or afterwards for descrambling. This is quite unacceptable due to a 1024×1024 color image represented in a 8-bit scale itself is 3 megabytes of information, whereas a 1024×1024 grayscale image is just 1 megabyte. Contrariwise, the knight-tour scrambling at the depth of, say, 5, is accompanied with only 36 bytes ensuring an acceptably low guess probability.

DISCUSSION OF THE CONTRIBUTION

In addition to the knight-tour scrambling advantages, a convention for scrambling keys can be applied to avoid sending values (16) for descrambling. Such conventions are really plausible and effective when a small group of data chunks is sent over. Theoretically, a convention for the entries of matrix (17) might be applied, too. But once the shuffler scrambling key must be unexpectedly changed (e. g., by the reason of an oncoming cyberthreat), it will be required nonetheless at the descrambling side, unless there is another convention for the case of changing the key. Anyway, an information signal about the change should be sent. Such inconvenience makes the shuffler scrambling too bulky and far inefficient.

However, a deeper shuffling still can be applied by the analogy to the scrambling of some depth. For such a scrambling, matrix (17) may be used repeatedly, or a series of such matrices is used. A twofold application of the shuffling decreases the guess probability down to N^{-4}, which becomes acceptable for images or data sets comprising at least a few hundred (numeric or symbolic) elements.

The suggested knight open tour algorithm is a significant contribution to the field of data scrambling by three reasons. First, its implementation by Algorithm 2 for scrambling and Algorithm 3 for descrambling, with Algorithm 1 for generating a knight open tour, is very simple and easy to understand. Second, the knight-tour scrambling maintains a linear time complexity. Third, its guess probability is sufficiently low if an appropriate depth of scrambling is used. Eventually, the data scrambler knight tour algorithm can be combined with other linear-time-complexity scrambling algorithms to further strengthen the data protection.

In addition to the simplicity, the data scrambler and descrambler by the knight open tour algorithm are practically symmetric. Hence, they are easy applicable yet ensuring proper security of data. Despite the knight open tour algorithm is presented for the square matrix, it is scalable to any size. After all, the chessboard matrix can be used partially for non-square data matrices. Moreover, the data can be reshaped into a vector, either row or column, comprising an odd number of entries, whereupon a part of the chessboard matrix is still used to scramble the data.

CONCLUSION

A data scrambling algorithm has been suggested based on a knight open tour problem solution, whose structure seems chaotic enough to substitute ordinary shuffling. The linear time complexity of the algorithm inspires its practical embedding mainly owing to the scrambling non-sophisticated subtlety and low guess probability. The latter is regulated by changing the scrambling depth determined by repetitive application of the chessboard matrix. The size of this matrix is usually increased as the scrambling is deepened.

The research is possible to extend and advance in the way of studying more favorable knight open tours for the given chessboard matrix. As the computation time grows in a quadratic manner by deepening the scrambling, but the guess probability is lowered far stronger, a tradeoff between the depth and re-identification risk might be ascertained. Another open question is which strategy of increasing the chessboard matrix size would be optimal to attain the tradeoff at the shallowest possible depth.

REFERENCES

1. S.A. Abdelhameed, S.M. Moussa, and M.E. Khalifa, "Privacy-preserving tabular data publishing: A comprehensive evaluation from web to cloud," *Computers & Security*, vol. 72, pp. 74–95, 2018. doi: https://doi.org/10.1016/j.cose.2017.09.002
2. N. Li, N. Zhang, S.K. Das, and B. Thuraisingham, "Privacy preservation in wireless sensor networks: A state-of-the-art survey," *Ad Hoc Networks*, vol. 7, issue 8, pp. 1501–1514, 2009. doi: https://doi.org/10.1016/j.adhoc.2009.04.009
3. L. Zhang, S. Hao, J. Zheng, Y. Tan, Q. Zhang, and Y. Li, "Descrambling data on solid-state disks by reverse-engineering the firmware," *Digital Investigation*, vol. 12, pp. 77– 87, 2015. doi: https://doi.org/10.1016/j.diin.2014.12.003
4. S. Kim, M. Kyoung Sung, and Y. Dohn Chung, "A framework to preserve the privacy of electronic health data streams," *Journal of Biomedical Informatics*, vol. 50, pp. 95–106, 2014. doi: https://doi.org/10.1016/j.jbi.2014.03.015
5. S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati, "Chapter 18 — Digital infrastructure policies for data security and privacy in smart cities," in *J.R. Vacca (Ed.), Smart Cities Policies and Financing*. Elsevier, 2022, pp. 249–261. doi: https://doi.org/10.1016/B978-0-12-819130-9.00007-3
6. R. Amdouni, M. Gafsi, R. Guesmi, M.A. Hajjaji, A. Mtibaa, and E.-B. Bourennane, "High-performance hardware architecture of a robust block-cipher algorithm based on different chaotic maps and DNA sequence encoding," *Integration*, vol. 87, pp. 346–363, 2022. doi: https://doi.org/10.1016/j.vlsi.2022.08.002
7. I.C. Semanjski, "Chapter 5 — Data analytics," in *I.C. Semanjski (Ed.), Smart Urban Mobility*. Elsevier, 2023, pp. 121–170. doi: https://doi.org/10.1016/B978-0-12-820717-8.00008-7
8. P. Pandya, "Chapter 15 — Advanced Data Encryption," in *J.R. Vacca (Ed.), Cyber Security and IT Infrastructure Protection*. Syngress, 2014, pp. 325–345. doi: https://doi.org/10.1016/B978-0-12-416681-3.00015-X
9. P. Sun, "Security and privacy protection in cloud computing: Discussions and challenges," *Journal of Network and Computer Applications*, vol. 160, Article ID 102642, 2020. doi: https://doi.org/10.1016/j.jnca.2020.102642
10. N. Sa'adah, "Trusted Data Transmission Using Data Scrambling Security Method with Asymmetric Key Algorithm for Synchronization," *EMITTER International Journal of Engineering Technology*, vol. 6, issue 2, p. 217, 2018. doi: https://doi.org/10.24003/emitter.v6i2.267
11. P. Pandya, "Chapter 70 — Advanced Data Encryption," in *J.R. Vacca (Ed.), Computer and Information Security Handbook* (Second Edition). Morgan Kaufmann, 2013, pp. 1127–1138. doi: https://doi.org/10.1016/B978-0-12-394397-2.00070-2
12. T. Stapko, "CHAPTER 8 — Choosing and Optimizing Cryptographic Algorithms for Resource-Constrained Systems," in *T. Stapko (Ed.), Practical Embedded Security*. Newnes, 2008, pp. 149– 171. doi: https://doi.org/10.1016/B978-075068215-2.50009-4
13. D.S. Guamán, D. Rodriguez, J.M. del Alamo, and J. Such, "Automated GDPR compliance assessment for cross-border personal data transfers in android applications," *Computers & Security*, vol. 130, Article ID 103262, 2023. doi: https://doi.org/10.1016/j.cose.2023.103262
14. R. Mia et al., "A comparative study on HIPAA technical safeguards assessment of android mHealth applications," *Smart Health*, vol. 26, Article ID 100349, 2022. doi: https://doi.org/10.1016/j.smhl.2022.100349
15. D. Salomon, *Data Privacy and Security: Encryption and Information Hiding*. Springer-Verlag, Berlin, Heidelberg, 2003, 480 p.
16. H. Zhong, C. Gu, Q. Zhang, J. Cui, C. Gu, and D. He, "Conditional privacy-preserving message authentication scheme for cross-domain Industrial Internet of Things," *Ad Hoc Networks*, vol. 144, Article ID 103137, 2023. doi: https://doi.org/10.1016/j.adhoc.2023.103137

17. W. Wang, H. Huang, Z. Yin, T. R. Gadekallu, M. Alazab, and C. Su, "Smart contract token-based privacy-preserving access control system for industrial Internet of Things," *Digital Communications and Networks*, vol. 9, issue 2, pp. 337–346, 2023. doi: https://doi.org/10.1016/j.dcan.2022.10.005
18. N. Khalid, A. Qayyum, M. Bilal, A. Al-Fuqaha, and J. Qadir, "Privacy-preserving artificial intelligence in healthcare: Techniques and applications," *Computers in Biology and Medicine*, vol. 158, Article ID 106848, 2023. doi: https://doi.org/10.1016/j.compbiomed.2023.106848
19. W.E. Yancey, W.E. Winkler, and R.H. Creecy, "Disclosure Risk Assessment in Perturbative Microdata Protection," in *J. Domingo-Ferrer (Ed.), Inference Control in Statistical Databases. Lecture Notes in Computer Science*, vol. 2316. Springer, Berlin, Heidelberg, 2002. doi: https://doi.org/10.1007/3-540-47804-3_11
20. V. Torra, "Microaggregation for Categorical Variables: A Median Based Approach," in *J. Domingo-Ferrer and V. Torra (Eds.), Privacy in Statistical Databases. PSD 2004. Lecture Notes in Computer Science,* vol. 3050. Springer, Berlin, Heidelberg, 2004. doi: https://doi.org/10.1007/978-3-540-25955-8_13
21. J. Domingo-Ferrer and V. Torra, "A Quantitative Comparison of Disclosure Control Methods for Microdata," in *P. Doyle, J.I. Lane, J.J.M. Theeuwes, and L.M. Zayatz (Eds.), Confidentiality, Disclosure, and Data Access: Theory and Practical Applications for Statistical Agencies*. Elsevier, Amsterdam, 2001, pp. 111–133.
22. W. Wu, Q. Qi, and X. Yu, "Deep learning-based data privacy protection in software-defined industrial networking," *Computers and Electrical Engineering*, vol. 106, Article ID 108578, 2023. doi: https://doi.org/10.1016/j.compeleceng.2023.108578
23. H.M. Ghadirli, A. Nodehi, and R. Enayatifar, "An overview of encryption algorithms in color images," *Signal Processing*, vol. 164, pp. 163–185, 2019. doi: https://doi.org/10.1016/j.sigpro.2019.06.010
24. S. Yaremko, E. Kuzmina, N. Savina, D. Yaremko, V. Kuzmin, and O. Adler, "Development of a Smart Education System for Analysis and Prediction of Students' Academic Performance," in *S. Babichev and V. Lytvynenko (Eds.), Lecture Notes in Computational Intelligence and Decision Making. ISDMCI 2021. Lecture Notes on Data Engineering and Communications Technologies*, vol. 77. Springer, Cham, 2022. doi: https://doi.org/10.1007/978-3-030-82014-5_52
25. H. A. Yehoshyna, V. V. Romanuke, "Constraint-based Recommender system for commodity realization," *Journal of Communications Software and Systems*, vol. 17, no. 4, pp. 314–320, 2021. doi: https://doi.org/10.24138/jcomss-2021-0102
26. U. Sopaoglu, O. Abul, "Classification utility aware data stream anonymization," *Applied Soft Computing*, vol. 110, Article ID 107743, 2021. doi: https://doi.org/10.1016/j.asoc.2021.107743
27. G. Loukides, A. Gkoulalas-Divanis, "Utility-preserving transaction data anonymization with low information loss," *Expert Systems with Applications*, vol. 39, issue 10, pp. 9764–9777, 2012. doi: https://doi.org/10.1016/j.eswa.2012.02.179
28. I. Parberry, "An efficient algorithm for the knight's tour problem," *Discrete Applied Mathematics*, vol. 73, issue 3, pp. 251–260, 1997. doi: https://doi.org/10.1016/S0166-218X(96)00010-8
29. M. Singh, A. Kakkar, and M. Singh, "Image encryption scheme based on knight's tour problem," *Procedia Computer Science*, vol. 70, pp. 245–250, 2015. doi: https://doi.org/10.1016/j.procs.2015.10.081
30. S.-S. Lin, C.-L. Wei, "Optimal algorithms for constructing knight's tours on arbitrary $n \times m$ chessboards," *Discrete Applied Mathematics*, vol. 146, issue 3, pp. 219–232, 2005. doi: https://doi.org/10.1016/j.dam.2004.11.002
31. M. Valtorta, M. I. Zahid, "Warnsdorff's tours of a knight," *Journal of Recreational Mathematics*, vol. 25, no. 4, pp. 263–275, 1993.
32. J. Jeffers, J. Reinders, and A. Sodani, "Chapter 9 — Vectorization," in *J. Jeffers, J. Reinders, and A. Sodani (Eds.), Intel Xeon Phi Processor High Performance Pro-*

gramming (Second Edition). Morgan Kaufmann, 2016, pp. 173–212. doi: https://doi.org/10.1016/B978-0-12-809194-4.00009-0

33. V.V. Romanuke, “Finite uniform approximation of zero-sum games defined on a product of staircase-function continuous spaces,” *Annals of the University of Craiova, Mathematics and Computer Science Series*, vol. 49, issue 2, pp. 270–290, 2022. doi: https://doi.org/10.52846/ami.v49i2.1554

Received 04.10.2023

INFORMATION ON THE ARTICLE

Vadim V. Romanuke, ORCID: 0000-0001-9638-9572, Vinnytsia Institute of Trade and Economics of State University of Trade and Economics, Ukraine, e-mail: romanukevadimv@gmail.com

Svitlana A. Yaremko, ORCID: 0000-0002-0605-9324, Vinnytsia Institute of Trade and Economics of State University of Trade and Economics, Ukraine, e-mail: s.yaremko@vtei.edu.ua

Olena M. Kuzmina, ORCID: 0000-0002-0061-9933, Vinnytsia Institute of Trade and Economics of State University of Trade and Economics, Ukraine, e-mail: o.kuzmina@vtei.edu.ua

Hanna A. Yehoshyna, ORCID: 0000-0002-2381-1231, Northwestern Polytechnic, Grande Prairie, Canada, e-mail: hyehoshyna@nwpolytech.ca

АЛГОРИТМ ЦИКЛУ ШАХОВОГО КОНЯ ДЛЯ СКРЕМБЛЮВАННЯ ДАНИХ / В.В. Романюк, С.А. Яремко, О.М. Кузьміна, Г.А. Єгошина

Анотація. Скремблювання даних у наш час залишається важливою методикою для захисту конфіденційної інформації за допомогою певного перемішування, після якого розшифрування є надважким, однак зберігається можливість легітимних дій з даними після скремблювання. Оскільки повноцінне маніпулювання діями з даними після скремблювання залишається проблемою на тлі ризику втрати даних та їх несанкціонованого розшифрування, скремблювання та дескремблювання мають виконуватись ще швидше, не збільшуючи ризики втрати та розшифрування. Алгоритм скремблювання повинен мати лінійну часову складність та ще більше мінімізувати зазначені ризики. Перспективним підходом є задача побудови відкритого циклу шахового коня, розв’язки якої виглядають наче випадкові послідовності позицій коня. Відтак формалізується алгоритм відкритого циклу шахового коня, за яким кінь наче хаотично пересувається по шаховій дошці. Ця формалізація представлена у формі відформатованого псевдокоду для подальшого його ефективного впровадження незалежно від мови програмування. На виході отримано квадратну матрицю, котра відображає відкритий цикл шахового коня. На основі матриці циклу шахового коня подано алгоритми скремблера та дескремблера даних у тому ж стилі. Ці алгоритми мають лінійну часову складність. Скремблювання на основі циклу шахового коня має досить низьку імовірність випадкової дешифрації за умови, якщо використана прийнятна глибина скремблювання, де дані повторно скремблюються. Глибина скремблювання визначається багаторазовим застосуванням матриці шахової дошки, розмір якої зазвичай збільшується з поглибленням скремблювання. Порівняно з псевдовипадковим перемішуванням даних разом зі збереженням індексів перемішування, ключ дескремблювання на основі циклу шахового коня зберігається і пересилається набагато простіше, забезпечуючи в той же час прийнятний захист даних.

Ключові слова: скремблювання даних, задача побудови відкритого циклу шахового коня, лінійна часова складність, імовірність випадкової дешифрації, глибина скремблювання.