Information Sciences 327 (2016) 91–109

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

2D Sudoku associated bijections for image scrambling

Yue Wu[a], Yicong Zhou[b,*], Sos Agaian[c], Joseph P. Noonan[a]

[a] Department of Electrical and Computer Engineering, Tufts University, Medford, MA 02155, United States
[b] Department of Computer and Information Science,University of Macau, Macau 999078, China
[c] Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249, United States

A R T I C L E I N F O

Article history:
Received 11 January 2015
Revised 30 May 2015
Accepted 7 August 2015
Available online 13 August 2015

Keywords:
Sudoku matrix
Matrix element representation
Two dimensional bijection
Image scrambling

A B S T R A C T

Sudoku refers to a two-dimensional (2D) number-placement puzzle with simple constraints that there are no repeated digits in each row, each column, or each block. Motivated by this Sudoku configuration, we introduce a number of Sudoku associated matrix element representations besides the conventional representation using matrix row-column pair. Specifically, they are representations via Sudoku matrix row-digit pair, digit-row pair, column-digit pair, digit-column pair, block-digit pair, and digit-block pair. This means we can secretly represent matrix elements via a Sudoku matrix, and furthermore develop new Sudoku associated 2D parametric bijections. To demonstrate the effectiveness and randomness of bijections, we introduce a simple but effective *Sudoku Associated Image Scrambler* only using 2D Sudoku associated bijections for image scrambling without bandwidth expansion. Simulations and comparisons demonstrate that the proposed method outperforms several state-of-the-art methods.

1. Introduction

With the development of digital technologies in the last decade, many aspects of daily life are changed rapidly, for example, paper letters are not used as often as in the last century, but email communications become one essential part for many people; conventional films are almost not used in nowadays digital cameras, but memory cards in various types become the dominant storage; signals of televisions now are not analog transmitted but digital transmitted in many cases; thick and heavy paper books are not the first choice for many people, but electronic book readers like Kindle or IPAD are more popular. As emails, digital images, digital books and other digital data carriers play an important role in people's life, the demand on the secure storage and transmission on digital data become a problem must be solved.

Digital images, one typical type of 2D data, are considered to contain a huge amount of information, for example, a family photo might tell not only who are in this family and how they look like, but rough ages for each member and maybe their healthy conditions; a diagnosis CT image might tell a doctor whether the patient is healthy or not and if he/she is sick how bad he/she is; and a satellite photo might give information whether the interested region is under constructions and what these constructions could be. Because the information contained in a digital image and the information might be inferred beyond a digital image, it is very important to protect these information from any unauthorized use. One way of protecting digital images is called image scrambling, which disorders pixel relationship in the original image so that the scrambled image with rearranged pixels become unintelligent and unrecognizable.

* Corresponding author. Tel.: +853 88228458; fax: +853 8822 2426.
E-mail addresses: ywu03@ece.tufts.edu (Y. Wu), yicongzhou@umac.mo (Y. Zhou).

http://dx.doi.org/10.1016/j.ins.2015.08.013
0020-0255/© 2015 Elsevier Inc. All rights reserved.

Mathematically speaking, all image scrambling algorithms rely on a 2D bijection, which maps pixels in the original image to rearranged pixels in the scrambled image. According to the generation of a 2D bijection used in a scrambler, we can classify a scrambler into groups: chaos based [25,30], spatial transform based [12,26], matrix decomposition based [20] and cellular automata based [4,28]. According to the way that an image scrambler uses a 2D bijection, we can roughly classify it into two groups: conventional scrambling [20] and total scrambling [4,25,26,28]. Conventional scrambling considers each pixel an unit and only shuffles pixel positions, while total scrambling considers each bit of a pixel an unit and shuffles pixel bit positions and pixel positions. Generally speaking, the total scrambling scheme is considered more secure than the conventional scheme because image histograms have been changed after total scrambling, but stays unchanged after conventional scrambling. Moreover, many recent research works have shown that image scrambling is an essential component for advanced data protection techniques [3,6,7,9,16,18,19,27,29,31–33]. For example, in [8], a chaos-based image encryption algorithm, whose fundamental is the image scrambling, was proposed; A image scrambling based blind watermarking algorithm was proposed in [11].

In this paper, we extend our work in [22,24] and introduce new ways of constructing 2D bijections for digital image scrambling using Sudoku matrices. We demonstrate that an $N \times N$ Sudoku matrix can be used to define six new parametric matrix element representations for an $N \times N$ matrix. Consequently, we are able to construct a 2D Sudoku associated bijection by mapping one matrix element representation to the other. Moreover, we demonstrate that 2D Sudoku associated bijections actually performs scrambling in a guarantee way. For example, the 2D Sudoku associated bijection mapping from the row-digit pair to the row-column pair is to scramble image pixels in such a way that two pixels originally lies in the same column will be in different columns after scrambling, while the 2D Sudoku associated bijection mapping from the digit-column pair to the row–column pair is to scramble image pixels in such a way that two pixels originally lies in the same row will be in different rows after scrambling. In addition, we propose a total scrambling scheme *Sudoku Associated Image Scrambler* only using on these 2D Sudoku associated bijections for scrambling. Simulation results on various image types and datasets demonstrate the effectiveness and robustness of the proposed method. Visual and Analytical comparisons to peer algorithms suggest that the *Sudoku Associated Image Scrambler* outperforms or reach state-of-the-art methods of image scrambling. Statistical testing results also support that *Sudoku Associated Image Scrambler* successfully decorrelates pixels in the original image to random-like.

The commonly used symbols and notations in this paper are given in Table 1. In the rest of the paper, Section 2 gives a brief background about Sudoku matrices; Section 3 discusses the Sudoku associated matrix element representations and 2D bijections; Section 4 proposes the *Sudoku Associated Image Scrambler* using 2D Sudoku associated bijections; Section 5 shows extensive simulation results and compares the performance of *Sudoku Associated Image Scrambler* with peer algorithms in detail; Section 6 concludes the paper.

Table 1
Description of symbols and notations.

Symbol	Description
N	a squared integer
$I(.)$	an indication function on range {0, 1}
r	row index of an element in a matrix
c	column index of an element in a matrix
b	block index of an element in a Sudoku matrix
g	grid index of an element in a Sudoku matrix
d	digit index of an element within a Sudoku matrix
$X_{r,c}$	a pixel/element located at the intersection of the *r*th row and *c*th column of image/matrix X
S	a Sudoku matrix
$\mathbb{I}$	the set of natural numbers from 1 to N
f	a 2D bijection on $\mathbb{I} \times \mathbb{I}$
f^{-1}	the inverse bijection of f
$f_{R\to R'}$	a 2D bijection mapping from a representation R to R'
$f_{R\leftarrow R'}$	a 2D bijection mapping from a representation R' to R
fix(.)	rounding function towards to zero
rem(.)	remainder function
F	a fixed matrix element representation pair
P	a matrix element parametric representation pair
R_F	a fixed matrix element representation using pair F
R_P^S	a matrix element representation using pair P with reference Sudoku S
$\circ$	the function composition symbol
W	the width of an image
H	the height of an image
T	the number of pixels of an image
K	a scrambling key
$\Gamma(.)$	the Gamma function
$\lvert.\rvert$	the absolute value function
g(.)	the Student's t-distribution function

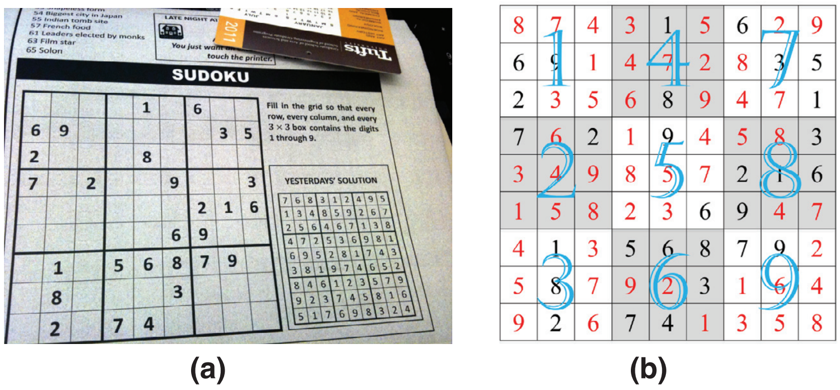


Fig. 1. A newspaper style Sudoku puzzle and its solution, (a) a Sudoku puzzle, and (b) its solution.

2. Background

The name *Sudoku* is the abbreviation of the Japanese '*Sunji wa dokushin ni kagiru*', which means 'single number'. Conventionally, Sudoku refers to a number-placement puzzle, consisting of 9×9 grids divided into nine 3×3 blocks [1][1]. The objective is to complete the grids using digits ranging from 1 to 9, in a manner that there are no repeated digits in any single row, column or block of the overall puzzle.

Fig. 1 shows a Sudoku puzzle in a newspaper and its solution. The Sudoku hints given in Fig. 1(a) are identified by black colored numerals; the nine Sudoku block indices are identified by the large blue colored numerals ranging from 1 to 9; and the blank Sudoku elements in Fig. 1(a) are identified by red colored numerals in Fig. 1(b), respectively. This is a conventional Sudoku puzzle, with a size of 9×9, to be filled with digits ranging from 1 to 9, and divided in square blocks of size 3×3.

In this paper, we are interested in a Sudoku solution instead of a Sudoku puzzle, because multiple Sudoku puzzles can be made with respect to a single Sudoku solution, and it is the Sudoku solution that satisfies all digit constraints along rows, columns and blocks. Specifically, we consider a Sudoku solution from a viewpoint of matrix and thus call it a Sudoku matrix. It is noticeable that one can easily extend the conception of a 9×9 Sudoku matrix to other squared sizes, e.g. 4×4, 16×16, 25×25 etc. Fig. 2 shows examples of large-size Sudoku matrices. Therefore, we can define an $N \times N$ Sudoku matrix with $N = n^2$ as shown in Definition 1.

3. Methodology

In this section, we mainly explore the possible matrix representations associated with a Sudoku matrix. And we show that element-wise mapping between any of two representations is actually a bijection, which ensures to shuffle and restore image pixels in an easy way.

3.1. Sudoku matrix

An $N \times N$ matrix is a Sudoku matrix if and only if its elements satisfy the constraints that the matrix elements in any row, and in any column, and in any $n \times n$ block contain exactly N digits from 1 to N.

Mathematically, we can define an $N \times N$ squared size Sudoku matrix via an indicator function I as follows:

$$I_X(r, c, d) = \begin{cases} 1, \text{ if } X_{r,c} = d \\ 0, \text{ Otherwise} \end{cases} \tag{1}$$

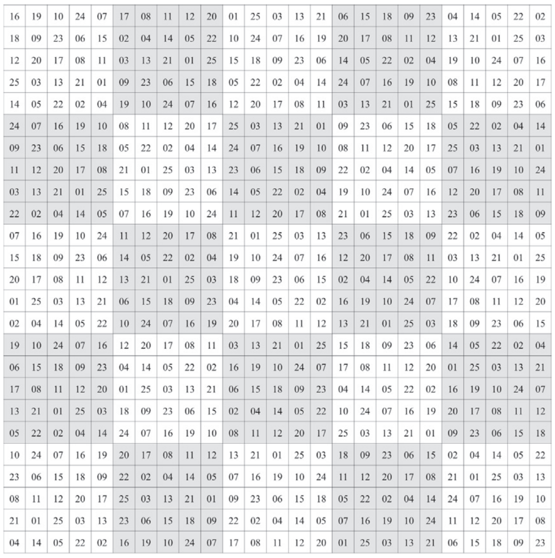
where r, c and d denote the row index, column index, and digit index of a matrix element in matrix X; and $X_{r,c}$ denotes the element at the intersection of the rth row and cth column in X. Let $\mathbb{I} = \{1, 2, \ldots, N\}$ be the index set. Then

Definition 1 (Sudoku Matrix). An $N \times N$ matrix S is a Sudoku matrix if and only if

- For arbitrary $c, d \in \mathbb{I}$, we have $\sum_{r=1}^{N} I_S(r, c, d) = 1$
- For arbitrary $r, d \in \mathbb{I}$, we have $\sum_{c=1}^{N} I_S(r, c, d) = 1$
- For arbitrary $b, d \in \mathbb{I}$, we have $\sum_{g=1}^{N} I_S(b_r^g, b_c^g, d) = 1$

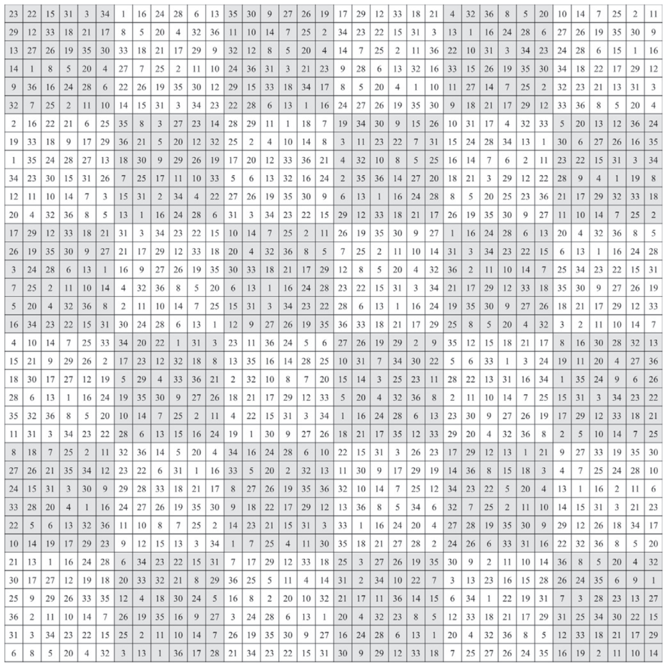
where r, c, b and d denotes the row index, column index, block index and digit index, respectively; b_r^g and b_c^g denote the row index and the column index of the gth grid in the bth block.

[1] In some literatures, this 3×3 block is referred as a *box*, or a *square*.



16	19	10	24	07	17	08	11	12	20	01	25	03	13	21	06	15	18	09	23	04	14	05	22	02
18	09	23	06	15	02	04	14	05	22	10	24	07	16	19	20	17	08	11	12	13	21	01	25	03
12	20	17	08	11	03	13	21	01	25	15	18	09	23	06	14	05	22	02	04	19	10	24	07	16
25	03	13	21	01	09	23	06	15	18	05	22	02	04	14	24	07	16	19	10	08	11	12	20	17
14	05	22	02	04	19	10	24	07	16	12	20	17	08	11	03	13	21	01	25	15	18	09	23	06
24	07	16	19	10	08	11	12	20	17	25	03	13	21	01	09	23	06	15	18	05	22	02	04	14
09	23	06	15	18	05	22	02	04	14	24	07	16	19	10	08	11	12	20	17	25	03	13	21	01
11	12	20	17	08	21	01	25	03	13	23	06	15	18	09	22	02	04	14	05	07	16	19	10	24
03	13	21	01	25	15	18	09	23	06	14	05	22	02	04	19	10	24	07	16	12	20	17	08	11
22	02	04	14	05	07	16	19	10	24	11	12	20	17	08	21	01	25	03	13	23	06	15	18	09
07	16	19	10	24	11	12	20	17	08	21	01	25	03	13	23	06	15	18	09	22	02	04	14	05
15	18	09	23	06	14	05	22	02	04	19	10	24	07	16	12	20	17	08	11	03	13	21	01	25
20	17	08	11	12	13	21	01	25	03	18	09	23	06	15	02	04	14	05	22	10	24	07	16	19
01	25	03	13	21	06	15	18	09	23	04	14	05	22	02	16	19	10	24	07	17	08	11	12	20
02	04	14	05	22	10	24	07	16	19	20	17	08	11	12	13	21	01	25	03	18	09	23	06	15
19	10	24	07	16	12	20	17	08	11	03	13	21	01	25	15	18	09	23	06	14	05	22	02	04
06	15	18	09	23	04	14	05	22	02	16	19	10	24	07	17	08	11	12	20	01	25	03	13	21
17	08	11	12	20	01	25	03	13	21	06	15	18	09	23	04	14	05	22	02	16	19	10	24	07
13	21	01	25	03	18	09	23	06	15	02	04	14	05	22	10	24	07	16	19	20	17	08	11	12
05	22	02	04	14	24	07	16	19	10	08	11	12	20	17	25	03	13	21	01	09	23	06	15	18
10	24	07	16	19	20	17	08	11	12	13	21	01	25	03	18	09	23	06	15	02	04	14	05	22
23	06	15	18	09	22	02	04	14	05	07	16	19	10	24	11	12	20	17	08	21	01	25	03	13
08	11	12	20	17	25	03	13	21	01	09	23	06	15	18	05	22	02	04	14	24	07	16	19	10
21	01	25	03	13	23	06	15	18	09	22	02	04	14	05	07	16	19	10	24	11	12	20	17	08
04	14	05	22	02	16	19	10	24	07	17	08	11	12	20	01	25	03	13	21	06	15	18	09	23

(a)



23	22	15	31	3	34	1	16	24	28	6	13	35	30	9	27	26	19	17	29	12	33	18	21	4	32	36	8	5	20	10	14	7	25	2	11
29	12	33	18	21	17	8	5	20	4	32	36	11	10	14	7	25	2	34	23	22	15	31	3	13	1	16	24	28	6	27	26	19	35	30	9
13	27	26	19	35	30	33	18	21	17	29	9	32	12	8	5	20	4	14	7	25	2	11	36	22	10	31	3	34	23	24	28	6	15	1	16
14	1	8	5	20	4	27	7	25	2	11	10	24	36	31	3	21	23	9	28	6	13	32	16	33	15	26	19	35	30	34	18	22	17	29	12
9	36	16	24	28	6	22	26	19	35	30	12	29	15	33	18	34	17	8	5	20	4	1	10	11	27	14	7	25	2	32	23	21	13	31	3
32	7	25	2	11	10	14	15	31	3	34	23	22	28	6	13	1	16	24	27	26	19	35	30	9	18	21	17	29	12	33	36	8	5	20	4
2	16	22	21	6	25	35	8	3	27	23	14	28	29	11	1	18	7	19	34	30	9	15	26	10	31	17	4	32	33	5	20	13	12	36	24
19	33	18	9	17	29	36	21	5	20	12	32	25	2	4	10	14	8	3	11	23	22	7	31	15	24	28	34	13	1	30	6	27	26	16	35
1	35	24	28	27	13	18	30	9	29	26	19	17	20	12	33	36	21	4	32	10	8	5	25	16	14	7	6	2	11	23	22	15	31	3	34
34	23	30	15	31	26	7	25	17	11	10	33	5	6	13	32	16	24	2	35	36	14	27	20	18	21	3	29	12	22	28	9	4	1	19	8
12	11	10	14	7	3	15	31	2	34	4	22	27	26	19	35	30	9	6	13	1	16	24	28	8	5	20	25	23	36	21	17	29	32	33	18
20	4	32	36	8	5	13	1	16	24	28	6	31	3	34	23	22	15	29	12	33	18	21	17	26	19	35	30	9	27	11	10	14	7	25	2
17	29	12	33	18	21	31	3	34	23	22	15	10	14	7	25	2	11	26	19	35	30	9	27	1	16	24	28	6	13	20	4	32	36	8	5
26	19	35	30	9	27	21	17	29	12	33	18	20	4	32	36	8	5	7	25	2	11	10	14	31	3	34	23	22	15	6	13	1	16	24	28
3	24	28	6	13	1	16	9	27	26	19	35	30	33	18	21	17	29	12	8	5	20	4	32	36	2	11	10	14	7	25	34	23	22	15	31
7	25	2	11	10	14	4	32	36	8	5	20	6	13	1	16	24	28	23	22	15	31	3	34	21	17	29	12	33	18	35	30	9	27	26	19
5	20	4	32	36	8	2	11	10	14	7	25	15	31	3	34	23	22	28	6	13	1	16	24	19	35	30	9	27	26	18	21	17	29	12	33
16	34	23	22	15	31	30	24	28	6	13	1	12	9	27	26	19	35	36	33	18	21	17	29	25	8	5	20	4	32	3	2	11	10	14	7
4	10	14	7	25	33	34	20	22	1	31	3	23	11	36	24	5	6	27	26	19	29	2	9	35	12	15	18	21	17	8	16	30	28	32	13
15	21	9	29	26	2	17	23	12	32	18	8	13	35	16	14	28	25	10	31	7	34	30	22	5	6	33	1	3	24	19	11	20	4	27	36
18	30	17	27	12	19	5	29	4	33	36	21	2	32	10	8	7	20	15	14	3	25	23	11	28	22	13	31	16	34	1	35	24	9	6	26
28	6	13	1	16	24	19	35	30	9	27	26	18	21	17	29	12	33	5	20	4	32	36	8	2	11	10	14	7	25	15	31	3	34	23	22
35	32	36	8	5	20	10	14	7	25	2	11	4	22	15	31	3	34	1	16	24	28	6	13	23	30	9	27	26	19	17	29	12	33	18	21
11	31	3	34	23	22	28	6	13	15	16	24	19	1	30	9	27	26	18	21	17	35	12	33	29	20	4	32	36	8	2	5	10	14	7	25
8	18	7	25	2	11	32	36	14	5	20	4	34	16	24	28	6	10	22	15	31	3	26	23	17	29	12	13	1	21	9	27	33	19	35	30
27	26	21	35	34	12	23	22	6	31	1	16	33	5	20	2	32	13	11	30	9	17	29	19	14	36	8	15	18	3	4	7	25	24	28	10
24	15	31	3	30	9	29	28	33	18	21	17	8	27	26	19	35	36	32	10	14	7	25	12	34	23	22	5	20	4	13	1	16	2	11	6
33	28	20	4	1	16	24	27	26	19	35	30	9	18	22	17	29	12	13	36	8	5	34	6	32	7	25	2	11	10	14	15	31	3	21	23
22	5	6	13	32	36	11	10	8	7	25	2	14	23	21	15	31	3	33	1	16	24	20	4	27	28	19	35	30	9	29	12	26	18	34	17
10	14	19	17	29	23	9	12	15	13	3	34	1	7	25	4	11	30	35	18	21	27	28	2	24	26	6	33	31	16	22	32	36	8	5	20
21	13	1	16	24	28	6	34	23	22	15	31	7	17	29	12	33	18	25	3	27	26	19	35	30	9	2	11	10	14	36	8	5	20	4	32
30	17	27	12	19	18	20	33	32	21	8	29	36	25	5	11	4	14	31	2	34	10	22	7	3	13	23	16	15	28	26	24	35	6	9	1
25	9	29	26	33	35	12	4	18	30	24	5	16	8	2	20	10	32	21	17	11	36	14	15	6	34	1	22	19	31	7	3	28	23	13	27
36	2	11	10	14	7	26	19	35	16	9	27	3	24	28	6	13	1	20	4	32	23	8	5	12	33	18	21	17	29	31	25	34	30	22	15
31	3	34	23	22	15	25	2	11	10	14	7	26	19	35	30	9	27	16	24	28	6	13	1	20	4	32	36	8	5	12	33	18	21	17	29
6	8	5	20	4	32	3	13	1	36	17	28	21	34	23	22	15	31	30	9	29	12	33	18	7	25	27	26	24	35	16	19	2	11	10	14

(b)

Fig. 2. Sudoku matrices with large sizes of (a) 25×25 and (b) 36×36.

An $N \times N$ Sudoku matrix conforming Definition 1 has the properties including, but not limited to the four listed below

- The N elements within each row of this Sudoku matrix is a permutation of the integer sequence $\{1, 2, \ldots, N\}$
- The N elements within each column of this Sudoku matrix is a permutation of the integer sequence $\{1, 2, \ldots, N\}$
- The N elements within each block of this Sudoku matrix is a permutation of the integer sequence $\{1, 2, \ldots, N\}$
- A subset of $N \times N$ Sudoku matrices can be parametrically generated [24]

The first three properties are directly from the definition of a Sudoku matrix. Solving a $N \times N$ Sudoku puzzle is NP-complete [1,10], but it is possible to quickly generate a random $N \times N$ Sudoku matrix which belongs to special families (see detailed methods in [21]), e.g. Keedwell Sudoku [13] and symmetric Sudoku [2].

In particular, the row index r and the column index c of the gth grid in the bth block are defined as follows,

$$f_{(b,g)\to(r,c)} : \begin{cases} b_r^e = \text{rem}(b-1, n) \cdot n + \text{rem}(g-1, n) + 1 \\ b_c^g = \text{fix}(b-1, n) \cdot n + \text{fix}(g-1, n) + 1 \end{cases} \tag{2}$$

with the rounding function towards to zero fix(.), namely $\text{fix}(p, q) = \lfloor p/q \rfloor$ and the remainder function rem(.) , namely $\text{rem}(p, q) = p - \text{fix}(p, q) \cdot q$. In this way, we define the two-dimensional mapping from the Sudoku representation $R_{(b,\, g)}$ to the conventional representation $R_{(r,\, c)}$. It is noticeable that this is a bijective mapping, and we can write the inverse mapping as

$$f_{(r,c)\to(b,g)} : \begin{cases} g = \text{rem}(r-1, n) \cdot n + \text{rem}(c-1, n) + 1 \\ b = \text{fix}(r-1, n) \cdot n + \text{fix}(c-1, n) + 1 \end{cases} \tag{3}$$

Although mathematically the bijective mapping between representation $R_{(r,\, c)}$ and representation $R_{(b,\, g)}$ can be explicitly written as Eqs. (2) and (3), such bijective mapping is actually nothing but to link two representations with respect to each matrix element.

Fig. 3 shows an example of matrix element representations $R_{(r,\, c)}$ and $R_{(b,\, g)}$ for a 4×4 matrix. As can be seen, the actual mapping from representation $R_{(r,\, c)}$ to representation $R_{(b,\, g)}$ defined in Eq. (2) can be simply described to map an (r, c) pair in Fig. 3(b) to its corresponding (b, g) pair in Fig. 3(c) with the same color. Similarly, the actual mapping from representation $R_{(b,\, g)}$ defined in Eq. (3) to representation $R_{(r,\, c)}$ is to map a (b, g) pair in Fig. 3(c) to its corresponding (r, c) pair in Fig. 3(b) with the same color. Consequently, the bijection between these two representations are shown in Fig. 3(d).

So far we have two matrix element representations, namely row-column pair representation $R_{(r,\, c)}$ and block-element pair representation $R_{(b,\, g)}$. To differentiate from those representation associated with one or more Sudoku matrices, we call these two representations *fixed representations* because they are fixed rather than parametric and use symbol F to denote a fixed representation pair, i.e. $F \in \{(r, c), (b, g)\}$.

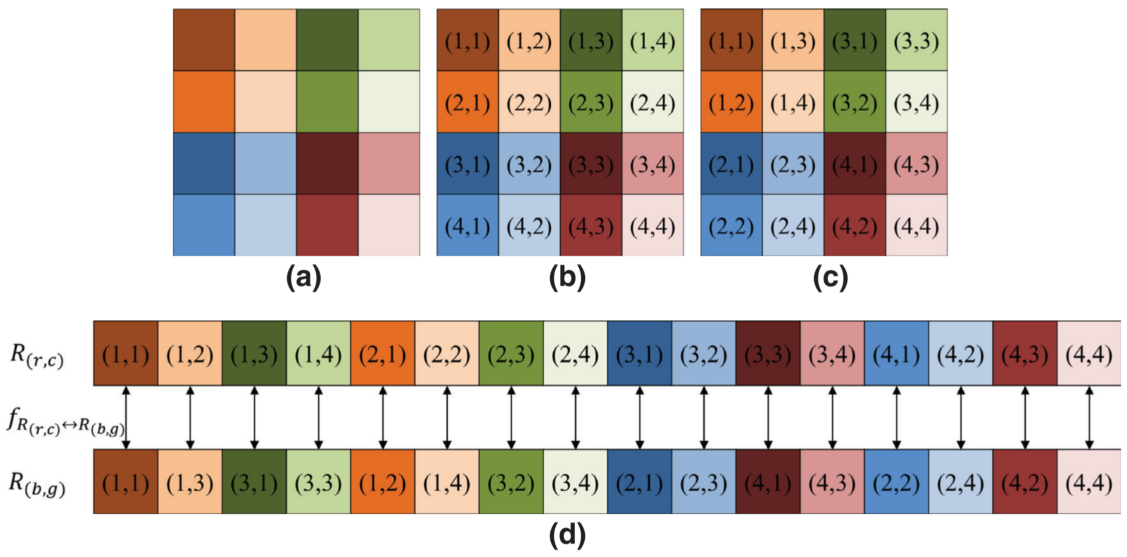


Fig. 3. Matrix element representation example; (a) a 4×4 matrix with each element identified in a unique color; (b) the conventional row-column representation $R_{(r,c)}$; (c) the Sudoku representation $R_{(b,g)}$; and (d) the bijective mapping between matrix element representation $R_{(r,c)}$ and $R_{(b,g)}$. .

3.2. Sudoku associated matrix element representations

Considering the task of digital image scrambling, the 2D bijection $f_{R_{(r,c)} \leftarrow R_{(b,g)}}$ and $f_{R_{(r,c)} \rightarrow R_{(b,g)}}$ are inappropriate in the sense that such bijection is fixed. With the help of a Sudoku matrix, however, we are able to easily define 2D parametric bijections between two matrix element representations.

Due to the digit constraints in each row, each column and each block within an $N \times N$ Sudoku matrix S, we are able to create parametric matrix element representations for an $N \times N$ matrix using the reference Sudoku matrix S. Specifically, we are able to have six parametric matrix element representations using the row-digit pair (r, d), digit-row pair (d, r), column-digit pair (c, d), digit-column pair (d, c), block-digit pair (b, d) and digit-block pair (d, b). Intuitively, matrix elements can be represented with the (r, d) pair, because each matrix can be decomposed with respect to rows, and each element in a row is with a unique digit associated with a reference Sudoku in its row, implying that one can uniquely locate a matrix element whenever he knows the row index r, the digit index d and the reference Sudoku matrix S. Similarly, it is not difficult to see that other listed Sudoku associated pairs are also valid matrix element representations. We denote these Sudoku associated matrix element representations as R_P^S where S is the associated Sudoku matrix and P denotes a Sudoku associated representation pair, i.e. $P \in \{(r, d), (d, r), (c, d), (d, c), (b, d), (d, b)\}$.

Fig. 4 shows the six Sudoku associated matrix element representations for a 4×4 matrix, when the reference Sudoku S shown in Fig. 4(b) is used. Each representation is able to uniquely locate an element in M.

Besides the existence of the six Sudoku associated matrix element representations, the Sudoku associated matrix element representation could be quite different when the reference Sudoku matrices are different. Here is a simple example illustrating this fact for interested elements in a 4×4 matrix. Fig. 5(a) shows our interested elements, and (b), (c), (d) give three reference Sudoku matrices. For each interested element, we are then able to represent it with various Sudoku associated matrix element representations. For example, the *green* element located at (2, 3) in the conventional row–column representation is denoted as (2, 2) by using the (r, d) representation pair associated with Sudoku matrix S_1, because the element located at row index $r = 2$ and digit index $d = 2$ in S_1 is the green element. In the same manner, this green element can be also denoted as (4, 3) by using the representation $R_{(d,b)}$ associated with Sudoku matrix S_3. And Table 2 shows the six matrix element representations for interested elements associated reference Sudoku matrices S_1, S_2 and S_3. As can be seen from this table, a matrix element can be represented with different Sudoku associated representation pairs including (r, d), (d, r), (c, d), (d, c), (b, d) and (d, b); and can also be represented differently by one Sudoku associated matrix element representation, if the associated reference Sudoku matrix changes. The reason behind these differences is that different Sudoku matrices have different digits in a grid. Moreover, due to the fact that these Sudoku associated matrix element representations are sensitive to the reference Sudoku matrix, we are therefore able to construct parametric 2D bijective functions for image scrambling.

It is clear that for each Sudoku matrix, there are six associated matrix element representations. Because any two matrix element representations denote the same set of matrix elements, a bijective mapping can be then constructed correspondingly like what we showed in Fig. 3(d). Since we have two fixed matrix element representations, namely row–column pair and block-grid pair, we are able to construct two new bijections denoted as $f_{R_P^S \rightarrow R_F}$ by mapping from the Sudoku associated matrix element representation R_P^S to the two fixed representations R_F, and other two new bijections by mapping from two fixed representations to the Sudoku associated matrix element representation denoted as $f_{R_P^S \leftarrow R_F}$, where S denotes the reference Sudoku matrix, R

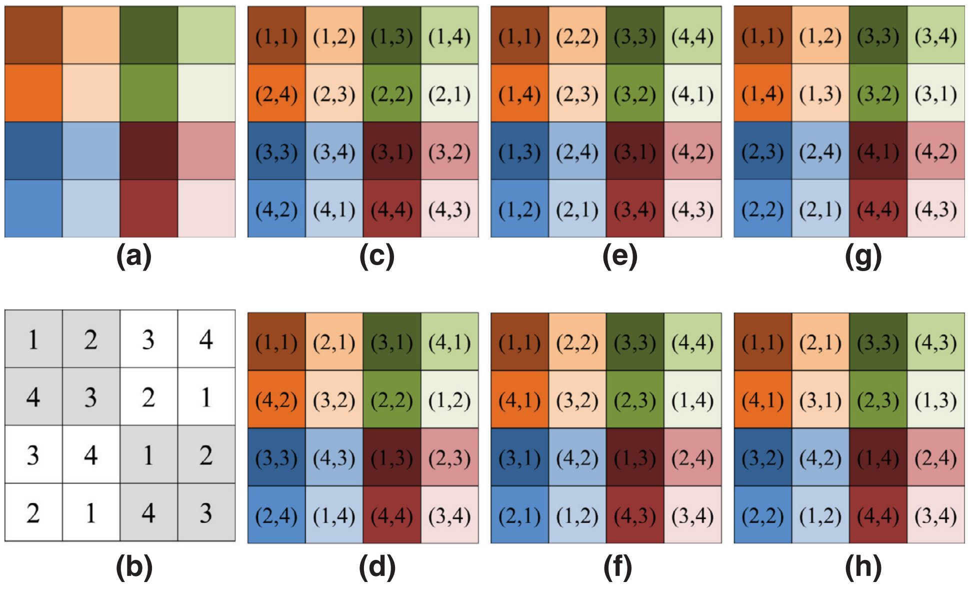


Fig. 4. Sudoku associated matrix element representations; (a) a 4 × 4 matrix M with elements identified in distinctive colors, (b) reference Sudoku matrix S, and matrix element representations (c) $R^S_{(r,d)}$; (d) $R^S_{(d,r)}$; (e) $R^S_{(c,d)}$; (f) $R^S_{(d,c)}$; (g) $R^S_{(b,d)}$; and (h) $R^S_{(d,b)}$.

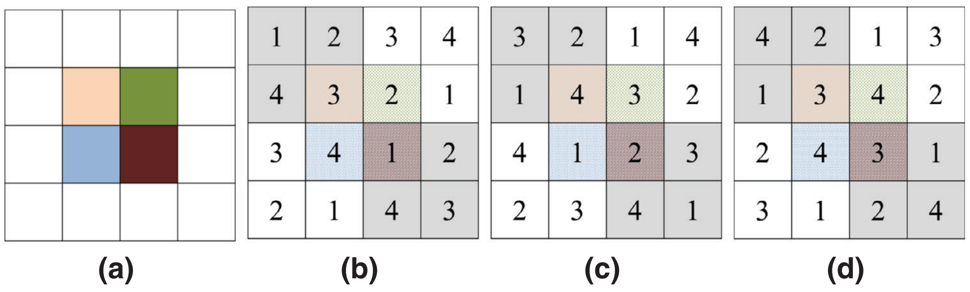


Fig. 5. A 4 × 4 matrix with colored elements and three reference Sudoku matrices; (a) interested matrix elements identified by colors, and reference Sudoku matrices (b) S_1; (c) S_2, and (d) S_3.

Table 2
Matrix element representations associated with a reference Sudoku matrix.

	S_1				S_2				S_3			
Representation pair	🟧	🟩	🟦	🟥	🟧	🟩	🟦	🟥	🟧	🟩	🟦	🟥
(r, d)	(2, 3)	(2, 2)	(3, 4)	(3, 1)	(2, 4)	(2, 3)	(3, 1)	(3, 2)	(2, 3)	(2, 4)	(3, 4)	(3, 3)
(d, r)	(3, 2)	(2, 2)	(4, 3)	(1, 3)	(4, 2)	(3, 2)	(1, 3)	(2, 3)	(3, 2)	(4, 2)	(4, 3)	(3, 3)
(c, d)	(2, 3)	(3, 2)	(2, 4)	(3, 1)	(2, 4)	(3, 3)	(2, 1)	(3, 2)	(2, 3)	(3, 4)	(2, 4)	(3, 3)
(d, c)	(3, 2)	(2, 3)	(4, 2)	(1, 3)	(4, 2)	(3, 3)	(1, 2)	(2, 3)	(3, 2)	(4, 3)	(4, 2)	(3, 3)
(b, d)	(1, 3)	(3, 2)	(2, 4)	(4, 1)	(1, 4)	(3, 3)	(2, 1)	(4, 2)	(1, 3)	(3, 4)	(2, 4)	(4, 3)
(d, b)	(3, 1)	(2, 3)	(4, 2)	(1, 4)	(4, 1)	(3, 3)	(1, 2)	(2, 4)	(3, 1)	(4, 3)	(4, 2)	(3, 4)

denotes a matrix element representation, P denotes a Sudoku associated representation pair with $P \in \{(r, d), (d, r), (c, d), (d, c), (b, d), (d, b)\}$, and F denotes a fixed matrix element representation with $F \in \{(r, c), (b, g)\}$. In summary, the total number Sudoku associated bijections can be directly constructed between a Sudoku associated matrix element representation to a fixed element representation is $6 \times 2 \times 2 = 24$.

Fig. 6 shows a naive one step image scrambling using the twenty-four Sudoku associated bijections. As can be seen from these results, different Sudoku associated bijections scramble the original *Trees* image differently. For example, the bijections $f_{R^S_{(r,d)} \rightarrow R_{(r,c)}}$ and $f_{R^S_{(r,c)} \leftarrow R_{(r,d)}}$ shuffle image pixels within each row; the bijections $f_{R^S_{(d,c)} \rightarrow R_{(r,c)}}$ and $f_{R^S_{(d,c)} \leftarrow R_{(r,c)}}$ shuffle image pixels within each column; the bijections $f_{R^S_{(d,b)} \rightarrow R_{(b,g)}}$ and $f_{R^S_{(d,b)} \leftarrow R_{(b,g)}}$ shuffle image pixels within each 16 × 16 block. It is also noticeable that $f_{R^S_{(d,b)} \rightarrow R_{(b,g)}}$, $f_{R^S_{(d,c)} \rightarrow R_{(b,g)}}$, and $f_{R^S_{(d,r)} \rightarrow R_{(b,g)}}$ are the three most powerful bijections that scramble the original *Trees* image to almost random-like from the viewpoint of human vision inspection.

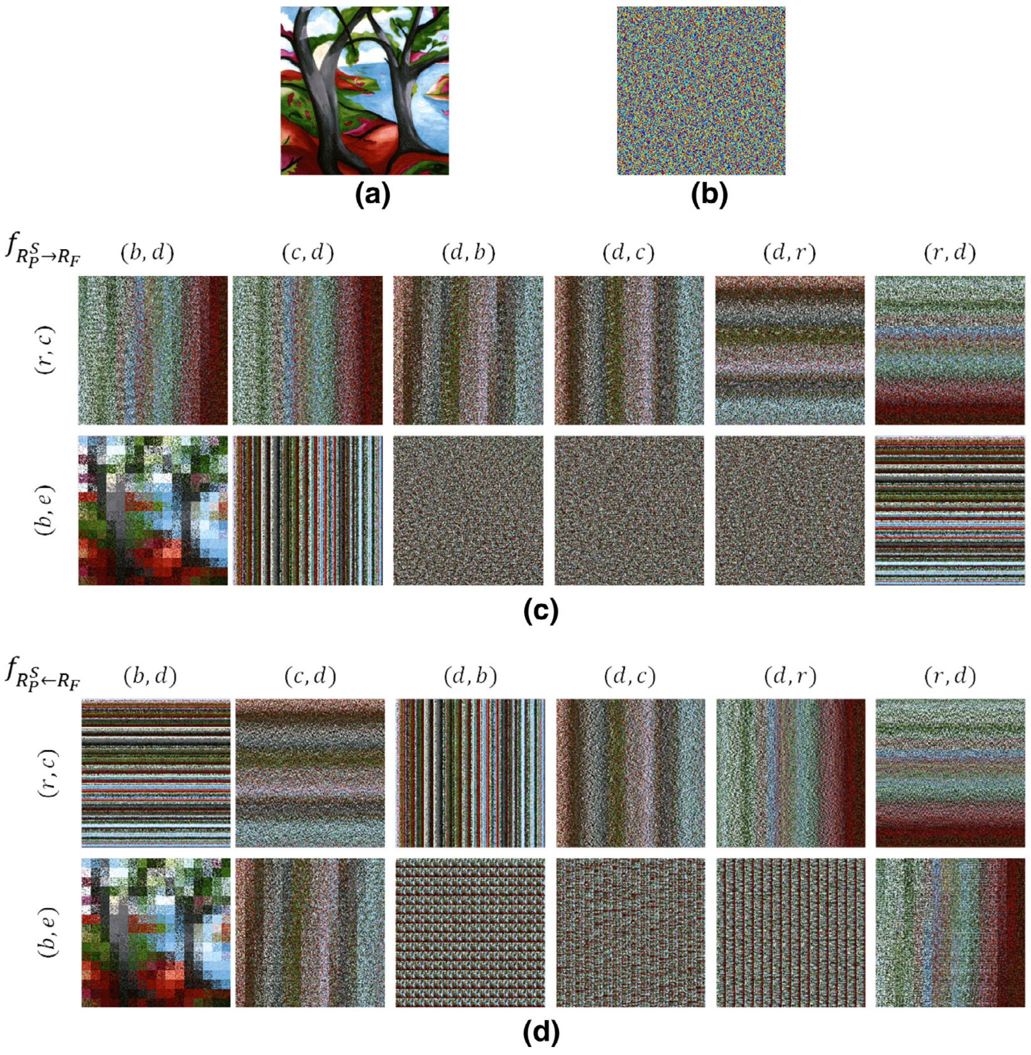


Fig. 6. One-step image scrambling results using the Sudoku associated bijections (a) input image, 256 × 256 *Trees*; (b) a 256 × 256 Sudoku matrix *S*; (c) scrambling results with the twelve bijections $f_{R_P^S \to R_F}$; and (d) scrambling results with the twelve bijections $f_{R_P^S \leftarrow R_F}$.

Table 3
The number of 2D bijective mappings associated with Sudoku matrices

# of 2D bijections: $\mathcal{N}_s^t$	# of Sudoku matrices: s			
Composition times t	1	2	$\cdots$	**m**
$t = 0$	24	48	$\cdots$	24 m
$t = 1$	24^2	48^2	$\cdots$	$(24\text{ m})^2$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$t = j$	24^{j+1}	48^{j+1}	$\cdots$	$(24\text{ m})^{j+1}$

3.3. More Sudoku associated bijections

It is well known that the composition of two bijective mapping is still a bijective mapping. In other words, if both f_0 and f_1 are two bijection on $\mathbb{I} \times \mathbb{I}$, $f_{new} = f_0 \circ f_1$ is also a bijection on $\mathbb{I} \times \mathbb{I}$ and so is $f_{new} = f_1 \circ f_0$. Therefore, we can even find more 2D bijective mappings by simply composing two or more existent bijective mappings.

Table 3 shows the relations of the number of reference Sudoku matrices and the number of 2D bijective mappings. When one Sudoku matrix is used for reference, six new matrix element representations can be found. Therefore, when m Sudoku matrices are used, there are 6 m Sudoku associated matrix element representations. Because a bijection can be constructed between a Sudoku associated matrix element representation and one fixed representation, the number of available bijections is $2 \times 2 \times 6\text{ m} = 24\text{ m}$, where 6 m is the number of Sudoku associated matrix element representations, 2 is the number of fixed matrix element representations, and 2 implies two possible ways to construct a bijection mapping either from the fixed representation to the Sudoku associated one or from the Sudoku associated one to the fixed one.

Because the composition of two bijections is just a new bijection, we can define more bijections by involving function compositions. Without loss of generality, say we are interested in the number of bijections defined by t times of function compositions from i bijections $f_1, f_2, \ldots, f_i$, i.e.

$$f_{i_0 \circ i_1 \cdots \circ i_t} = \underbrace{f_{i_0} \circ f_{i_1} \ldots \circ f_{i_t}}_{t\text{ times}}$$

where $i_0, i_1, \ldots, i_t \in \{1, 2, \ldots, i\}$. It is not difficult to see that the total number of bijections by t times of compositions is i^{t+1}. For each function f_{i_j} with $j \in \{0, 1, \ldots, t\}$ we have i candidate functions and in total we have $t+1$ functions to be determined. In Table 3 we have 24 m of possible Sudoku associated matrix element representations and j times of compositions i.e. $i = 24$ m and $t = j$, and thus the number of bijections involving j times of compositions is $i^{t+1} = (24m)^{j+1}$.

It is worthwhile to note that the number of bijections calculated in Table 3 includes self-mapping, i.e. $f_{new} = f_0 \circ f_1$ with $f_1 = f_0^{-1}$. However, as long as the number of Sudoku matrices increases, the possibility of randomly composing a pair of bijections (one is the inverse of the other) quickly approaches to zero. An alternative remedy to avoid self-mapping is to choose the two Sudoku associated bijections with different reference Sudoku matrices, so that a new composed bijection from these two bijections is impossible to be a self-mapping unless the two reference Sudoku matrices are identical.

4. Sudoku associated image scrambling scheme

In previous sections, we showed that given an $N \times N$ Sudoku matrix S for reference, there are six Sudoku associated matrix element representations, namely the row-digit pair representation $R^S_{(r,d)}$, digit-row pair representation $R^S_{(d,r)}$, column-digit pair representation $R^S_{(c,d)}$, digit-column pair representation $R^S_{(d,c)}$, block-digit pair representation $R^S_{(b,d)}$ and digit-block pair representation $R^S_{(d,b)}$. Each Sudoku S associated matrix element representation R^S_P with $P \in \{(r,d), (d,r), (c,d), (d,c), (b,d), (d,b)\}$ and a fixed representation R_F with $F \in \{(r,c), (b,g)\}$ can be used to construct a pair of bijections $f_{R^S_P \leftarrow R_F}$ and $f_{R^S_P \to R_F}$, where $f_{R^S_P \leftarrow R_F}$ denotes the bijection mapping from the fixed representation R_F to the Sudoku associated representation R^S_P, and $f_{R^S_P \to R_F}$ denotes the bijection mapping from the Sudoku associated representation R^S_P to the fixed representation R_F. Therefore, given a Sudoku matrix S and a fixed representation R_F, there are 24 Sudoku associated bijective mappings of $f^S_{R \to F}$ and $f^S_{R \leftarrow F}$. To investigate their applications, this section uses these 2D Sudoku associated bijections for digital image scrambling.

4.1. Sudoku associated image scrambler

Although a Sudoku associated image scrambler can be designed in various means, we design a Sudoku associated image scrambler as shown in Fig. 7, where *Key* is an encryption key of 192-bit length, *Parameter Generator* generates key dependent and round dependent parameters including a Sudoku associated 2D bijection $f_{R^S_P \to R_F}$ or $f_{R^S_P \leftarrow R_F}$, *Blockwise Scrambling* shuffles pixels within an $N \times N$ image block for every image block of the input image. Consequently, the image descrambling process is just the

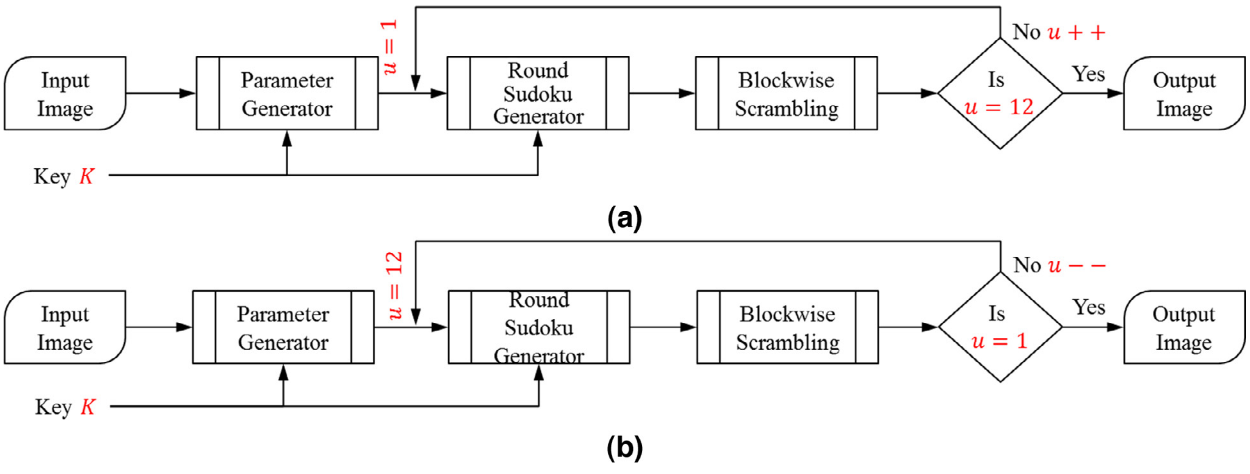


Fig. 7. Flowchart of a Sudoku associated image scrambler; (a) scrambling process; (b) descrambling process.

reverse of the scrambling process as shown in Fig. 7(b). In the following discussions, we consider an input image X is of size $W \times H$ with nB bit-depth.

Parameter Generator translates a scrambling key K to required parameters for image scrambling. Specifically, it takes a scrambling key K to generate a set of parameters including twelve Sudoku associated matrix element representation pairs $\mathbf{P^i} = \{P^i_1, P^i_2, \ldots, P^i_{12}\}$, twelve fixed representation pair $\mathbf{F^i} = \{F^i_1, F^i_2, \ldots, F^i_{12}\}$ and the twelve mapping parameters $\mathbf{m^i} = \{m^i_1, m^i_2, \ldots, m^i_{12}\}$ for the ith bit-plane in X as shown in Algorithm 1. In this way, *Parameter Generator* guarantees that for each

Algorithm 1 Parameter Generator $(\mathbf{R^i}, \mathbf{F^i}, \mathbf{m^i}) = BPG(K, i)$.

```
Require: $K$ is a key of 192 bits composed of twenty-four subkeys $\{K^{(1)}, K^{(2)}, \ldots, K^{(24)}\}$
Require: $i$ is the $i$th bit-plane to be processed
Ensure: $(\mathbf{P}, \mathbf{F}, \mathbf{m})$ is a list of twelve pairs of Sudoku associated matrix element representation pairs, fixed matrix element representation pairs, and mapping directions.
  $[K^{(1)}, K^{(2)}, \ldots, K^{(24)}] = K$ % Divide key $K$ to twenty-four subkeys $K^{(1)}, K^{(2)}, \ldots, K^{(24)}$
  $Q = [K^i, K^{mod(i+1,24)+1}, \ldots, K^{mod(i+12,24)+1}]$ % Extract a twelve element subkey sequence starting at $K^{(i)}$
  $idx = sort(Q)$ % Sort this sequence $Q$ and generate the element index sequence in the sorted sequence
  for $u = 1 : 1 : 12$ do
    $q = idx(j)$ % For each index in $idx$, pair it to a $(R, F)$ representations
    if $mod(q, 2) == 1$ then
      $m^i_u = 1$
    else
      $m^i_u = 0$
    end if
    if $q == 1$ then
      $P^i_u = (b, d), F^i_u = (r, c)$
    end if
    if $q == 2$ then
      $P^i_u = (b, d), F^i_u = (b, g)$
    end if
    if $q == 3$ then
      $P^i_u = (c, d), F^i_u = (r, c)$
    end if
    if $q == 4$ then
      $P^i_u = (c, d), F^i_u = (b, g)$
    end if
    if $q == 5$ then
      $P^i_u = (d, b), F^i_u = (r, c)$
    end if
    if $q == 6$ then
      $P^i_u = (d, b), F^i_u = (b, g)$
    end if
    if $q == 7$ then
      $P^i_u = (d, c), F^i_u = (r, c)$
    end if
    if $q == 8$ then
      $P^i_u = (d, c), F^i_u = (b, g)$
    end if
    if $q == 9$ then
      $P^i_u = (d, r), F^i_u = (r, c)$
    end if
    if $q == 10$ then
      $P^i_u = (d, r), F^i_u = (b, g)$
    end if
    if $q == 11$ then
      $P^i_u = (r, d), F^i_u = (r, c)$
    end if
    if $q == 12$ then
      $P^i_u = (r, d), F^i_u = (b, g)$
    end if
  end for
```

bit-plane in X all six possible Sudoku associated matrix element representation pairs and two possible fixed representation pairs are used in the twelve round scrambling. Meanwhile, these parameters are bit-plane dependent, implying different bit-planes will be scrambled in a different way in the future processing.

Furthermore, *Round Sudoku Generator* takes the image size and the scrambler round u to generate an $N \times N$ Sudoku matrix as shown in Algorithm 2.

Algorithm 2 Round Sudoku Generator $S_u = RSG(K, u, W, H)$.

```
Require: $K$ is a key of 256 bits
Require: $u$ is the cipher round number
Ensure: $S_u$ is an $N \times N$ Sudoku matrix
  $K_u = RoundKeyGenerator(K, u)$ % Generate a round key[2]
  $N = \lfloor \sqrt{\min(W, H)} \rfloor$ % Determine Sudoku size
  $S_u = SudokuGenerator(K_u, N)$ % Generate an $N \times N$ Sudoku matrix
```

Consequently, a 2D Sudoku associated bijection f_u^i can be found for the ith bit-plane in uth scrambler round as follows

$$f_u^i = \begin{cases} f_{R^{S_u}_{P_u^i} \leftarrow F_u^i}, & \text{if } m_u^i = 0 \\ f_{R^{S_u}_{P_u^i} \rightarrow F_u^i}, & \text{if } m_u^i = 1 \end{cases} \tag{4}$$

Once the $N \times N$ parametric Sudoku associated 2D bijection f_u^i is obtained for the ith bit-plane in the uth scrambler round, we are then able to scramble image pixels with in $N \times N$ image within the $W \times H$ input image for all its bit-planes. In order to obtain pixel shuffling between blocks, we apply an image shift function imageShift(.) for each cipher round as shown in Eq. (5).

$$Y = \text{imageShift}(X, w, h) = \left[\begin{array}{c|c} X(w:W, h:H) & X(w:W, 1:h-1) \\ \hline X(1:w-1, h:H) & X(1:w-1, 1:h-1) \end{array}\right] \tag{5}$$

Pseudo-code of *Blockwise Scarmbling* algorithm is given in Algorithm 3 in the MATLAB fashion. Consequently, we then shuffled every $N \times N$ image blocks within the input image X.

Algorithm 3 Blockwise Scrambling $Y = BlkScarmbling\,(X, \mathbf{f_u}, N, nB)$.

```
Require: $I$ is an image of size $W \times H$
Require: $\mathbf{f_u} = \{f_u^1, f_u^2, \ldots, f_u^{nB}\}$ is a vector of $nB$ bijections with each bijection for one bit-plane
Require: $N$ is a squared integer
Require: $nB$ is the number of bit-planes contained in image $X$
Ensure: $Y$ is an scrambled image of $X$ at the same size
  $nRow = \lceil W/N \rceil$
  $nCol = \lceil H/N \rceil$
  $w = \lceil W/12 \rceil$
  $h = \lceil H/12 \rceil$
  $X = imgShift(X, w, h)$ % shift image $X$ $w$ pixels along rows and $h$ pixels along columns
  for $i = 1:1:nRow$ do
    if $i\tilde{} = nRow$ then
      $r0 = (i-1)N + 1$ % 1st row index of the current block
      $r1 = iN$ % last row index of the current block
    else
      $r0 = W - N + 1$ % 1st row index of the current block
      $r1 = W$ % last row index of the current block
    end if
    for $j = 1:1:nCol$ do
      if $j\tilde{} = nCol$ then
        $c0 = (j-1)N + 1$ % 1st column index of the current block
        $c1 = jN$ % last column index of the current block
      else
        $c0 = H - N + 1$ % 1st column index of the current block
        $c1 = H$ % last column index of the current block
      end if
      $tBlk = I(r0:r1, c0:c1)$ % extract the current block
      $tmp = 0$ % create a temporary intermediate variable
      for $l = 1:1:nB$ do
        $tBP = bitget(tBlk, l)$ % extract the $u$th bit-plane
        $oBP(r0:r1, c0:c1) = f_u^l(tBP)$ % output scrambled bit-plane
        $tmp = tmp + oBP \cdot 2^{l-1}$ % store results in temporary variable
      end for
      $Y(r0:r1, c0:c1) = tmp$ % output scrambled block
    end for
  end for
```

In order to scramble an arbitrary $W \times H$ image without bandwidth expansion, we scramble blocks on the edge with a certain amount of overlapping with previous blocks as shown in Fig. 8. The block indices listed in Fig. 8 indicate the processing order

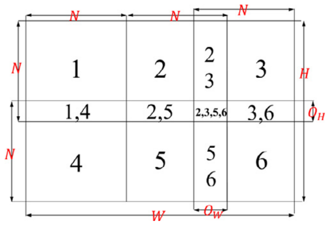


Fig. 8. Overlapped blocks in *Blockwise Scrambling*.

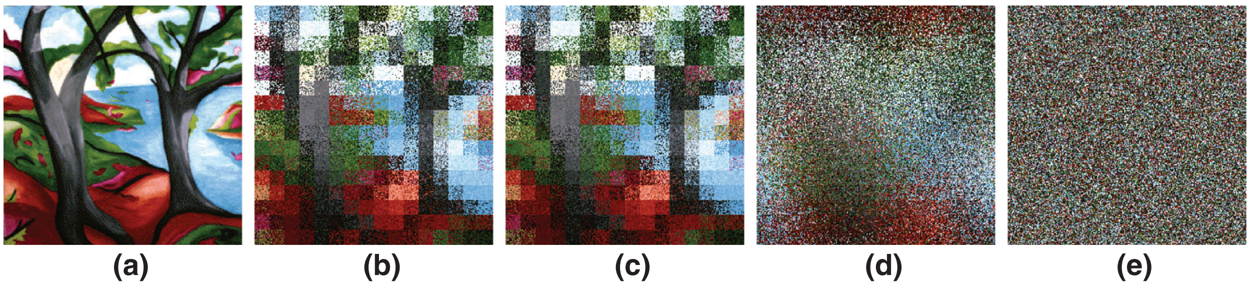


Fig. 9. Discussion of Image scrambling results; (a) image *Trees*; (b) scrambled image after one round $f_{R^S_{(b,d)} \to R_{(b,g)}}$; (c) scrambled image after twelve rounds $f_{R^S_{(b,d)} \to R_{(b,g)}}$; (d) scrambled image after twelve rounds $f_{R^S_{(b,d)} \to R_{(b,g)}}$ with image shifting; and (e) scrambled image after twelve rounds scrambling in *Sudoku Associated Image Scrambler*.

of *Blockwise Scrambling*, implying to first shuffle image pixels in the upper-left block marked as index 1 in the figure, then block 2, block 3, so on and so forth until block 6. Since W and H may not be necessarily to be divisible by N, we process blocks on the image edge with overlapping, e.g. block 2 and 3 are overlapped, and block 3 and block 6 are also overlapped. In summary, there are $O_H \times O_W$ pixels belongs to more than one block to be processed. In this way, we are able to fit a 2D Sudoku associated bijection defined on $\mathbb{I} \times \mathbb{I}$ for an input image with an arbitrary size $W \times H$ without bandwidth expansion.

In regarding of descrambling, we can simply substitute a certain Sudoku associated bijection with its inverse. In other words, if the bijection used for scrambling is $f_{R^S_P \leftarrow R_F}$, then the bijection used for descrambling is $f_{R^S_P \to R_F}$, and vice versa.

4.2. Discussions

Sudoku Associated Image Scrambler is a multiround scrambler that shuffles image pixels with respect to one Sudoku associated bijection for each round. This multiround scrambling process is equivalent to create a new bijection by taking the function composition of multiple bijections, which is discussed in Section 3.3. It is not difficult to validate such equivalence between the multiround Sudoku associated image scrambler and the bijection by composing multiple bijections. Without loss of generality, denote the twelve 2D Sudoku associated bijections for the ith bit-plane are $f^i_1, f^i_2, \ldots, f^i_{12}$ in *Sudoku Associated Image Scrambler*. Then running the scrambler for multiple times is nothing but to construct a new bijection f_{new} as follows.

$$f_{new}(x, y) = f^i_{12}(f^i_{11}(\cdots f^i_2(f^i_1(x, y)))) = f^i_{12} \circ f^i_{11} \cdots \circ f^i_2 \circ f^i_1(x, y) \tag{6}$$

which is actually the composition of the twelve bijections.

Meanwhile, we want to emphasize that the Sudoku associated bijections used in *Sudoku Associated Image Scrambler* from one round to another is not independent, but dependent in the sense that in the twelve round the fixed representation $R_{(r,c)}$ and $R_{(b,g)}$ appears exact six times respectively and the six Sudoku associated representations appears exact twice for each. We set up this constraint to guarantee the strong scrambling effect. Although different Sudoku associate bijections have different scrambling impact on an image, many of them are of limited scrambling effect in the sense that scrambled images are not random-like as shown in Fig. 6. Therefore, if we allow random Sudoku associated bijections for each round, it is possible to pick twelve identical Sudoku associated matrix element representation pairs, i.e. $P_1 = P_2 = \cdots = P_{12}$ and also twelve identical fixed representation pairs $F_1 = F_2 \cdots = F_{12}$. Although the reference Sudoku changes for each round, the scrambled image might not change that much. Fig. 9 shows an example of using bijection $f_{R^S_{(b,d)} \to R_{(b,g)}}$ for image scrambling. As this mapping implies, it only scrambles pixels within a Sudoku block. And thus scrambled image gets blurry soon after applying this bijection once. Consequently, the scrambled image after applying this bijection twelve times shown in Fig. 9(c) is more or less the same as the previous, indicating that simply composing a number of Sudoku bijections does not necessarily improves scrambling quality. Fig. 9(d) shows the scrambled image with additional image shifting operation defined in Eq. (5). This result shows that this operation does help improve scrambling quality by generating a more evenly distributed scrambled image. Fig. 9(e) shows the scrambled image using the twelve round *Sudoku Associated Image Scrambler* (in this particular case we consider image *Trees* to be bit-depth 1), which is completely unrecognizable and random-like.

In regarding of scrambler security, we suggest to change a scrambler key K frequently. Because an image scrambler itself is not semantically secure and is vulnerable under certain attacks, for example, a simple but effective algorithm based on the chosen plaintext attack (CPA) model could be described as follows,

1. Construct a all-zero image X of size $W \times H$ except the pixel $X_{i,j} \neq 0$
2. Scramble this image X by using an image scrambler and a unknown key K and obtain scrambled image Y
3. Find out nonzero value pixel in Y and record its position (r_{idx}, c_{idx}) as the output of a bijection f, i.e. $f(i, j) = (r_{idx}, c_{idx})$, where $idx = (i-1)H + j$
4. Repeat the above three steps by changing the position of the none-zero pixel in X, for all $i \in \{1, 2, \ldots, H\}$ and $j \in \{1, 2, \ldots, W\}$

Consequently, the bijective mapping f is the equivalent bijection of the used scrambler for a $W \times H$ image under the key K. In other words, all $W \times H$ images scrambled by the used image scrambler under key K can be perfectly cracked by inverse mapping scrambled image pixels using mapping f^{-1}. And the complexity of the above attack is $W \times H$.

It is worthwhile to note that the above attack is a generic attack for all image scramblers. However, one remedy to this type of attacks is to change encryption key frequently for a scrambler. For example, say we use the *Sudoku Associated Image Scrambler* for high-definition television (HDTV) with the format 1080*p*, namely 1920×1080 pixels per frame. Then an adversary need to take $1080 \times 1920 = 2073600$ frames for fully crack an scrambling key K. We therefore set up a key change for every $518400 = 2073600/4$ frames, which is equivalent to 6 h of HDTV programs if the frame rate is 24 frames per second. As a result, an adversary may at most recover 25% of pixels in a HDTV frame, which is definitely of poor visual quality.

5. Analysis and comparison results

Digital image scrambling is to rearrange image pixels in a deterministic way but with a random-like appearance. In this section we focus our effects on performance analysis and comparisons for the *Sudoku Associated Image Scrambler*.

5.1. Experiment settings

The following simulations are all performed under the *Windows 7* operation system with the *Intel Core2 CPU @2.66GHz* and 6GB memory. In order to make easy comparisons, we run our *Sudoku Associated Image Scrambler* on test images (8-bit grayscale) whose scrambling results are widely reported and compare our results with other peer algorithms including the scrambling method based on 2D cellular automata proposed by Abu Dalhoum et al. [4], the scrambling method based on chaos map proposed by Ye [25], the scrambling method of using cellular automata proposed by Ye et al. [28], and the scrambling method based on ASCII code of matrix element proposed by Ye et al. [26]. These test images includes *Cameraman*, *Barbara*, *Lenna*, and test images #157055, #69015 and #239096 in the Berkeley image segmentation dataset[2] [17]. Since the results about these test images have been previous reported, we make comparison of the proposed scrambler with peer methods by analyzing these test images.

Additionally, we also test the performance of the proposed scrambler on other image types than (8-bit grayscale) to validate that the proposed scrambler is able to deal with an image with an arbitrary bit depth, including binary images *CCITT-3* and *CCITT-7* from the CCITT fax image compression dataset [3], 16-bit grayscale knee MRI images, and color images *4.2.03* and *4.2.07* from the USC-SIPI miscellaneous image dataset[4].

Details about these test images and results with analysis and comparisons are presented in next sections.

5.2. Simulation results

Fig. 10 shows the simulation results of the proposed image scrambling method using the 2D Sudoku associated bijections with peer algorithms [4,25]. As can be seen from these results, the proposed method outperforms the two recently proposed algorithms, in the sense that its scrambled image is random-like and pixels more evenly scrambled, while results of Van De Ville et al.'s method [20] contain distinguishable patterns of foreground from background; results of Abu Dalhoum et al.'s method [4] contain line-like patterns and results of Ye's method [25] are of mesh-like patterns. Furthermore, it is noticeable that Abu Dalhoum et al.'s method might require extra space to deal with the edge effect in cellular automata (see results on the 2nd column), while the chaos map used Ye's method might be not that dynamic and generates weak results (see Ye's result on test image #239096).

Besides the 8-bit grayscale images listed in Fig. 10, we also test the proposed method on various image types including binary images *CCITT-3* and *CCITT-7* from the CCITT fax image compression dataset[4], 16-bit grayscale knee MRI images, and color images *4.2.03* and *4.2.07* from the USC-SIPI miscellaneous image dataset[5]. Scrambling results are given in Fig. 11.

In regard of the detail information about these test images including processing time and speed, we list these relevant information in Table 4. Roughly speaking, the speed of the proposed scrambler is about 150 KB/s, or 0.15 MB/s equivalently under

[2] This dataset is free for non-commercial research and educational purposes, available under http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/
[3] The CCITT database can be found under page: http://cdb.paradice-insight.us
[4] The USC-SIPI image database can be found on http://sipi.usc.edu/database/

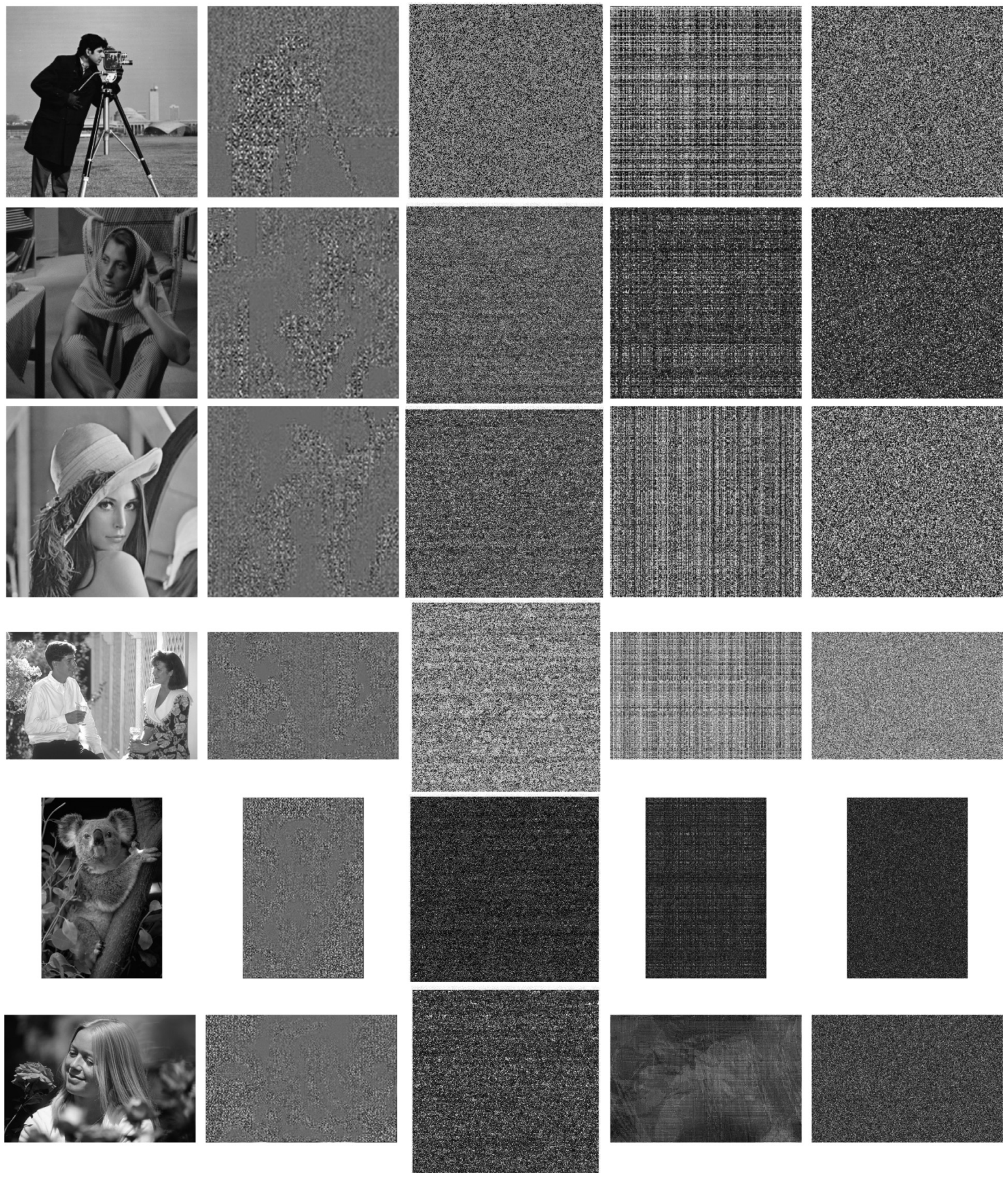


Fig. 10. Image scrambling results with visual comparisons; 1st column: test images (from 1st row to the last are *Cameraman*, *Barbara*, *Lenna*, and #157055, #69015 and #239096); 2nd column: simulation results of Van De Ville et al.'s method [20]; 3rd column: simulation results of Abu Dalhoum et al.'s method [4]; 4th column: simulation results of Ye's method [25]; and 5th column: simulation results of ours.

MATLAB environment. It is worthwhile to note that this speed can be largely enhanced by using parallel computations, because the scrambling process of each bit-plane is independent of each other and image blocks that not overlapped are also independent of each other, both of which implying these works can be done in parallel to save time. Meanwhile, it is well-known that MATLAB is very slow for *for-loop* execution, and thus implementing the proposed scrambler in other languages might further enhance the processing speed. Therefore, the proposed scrambler with a proper implement could meet the demand for real time processings.

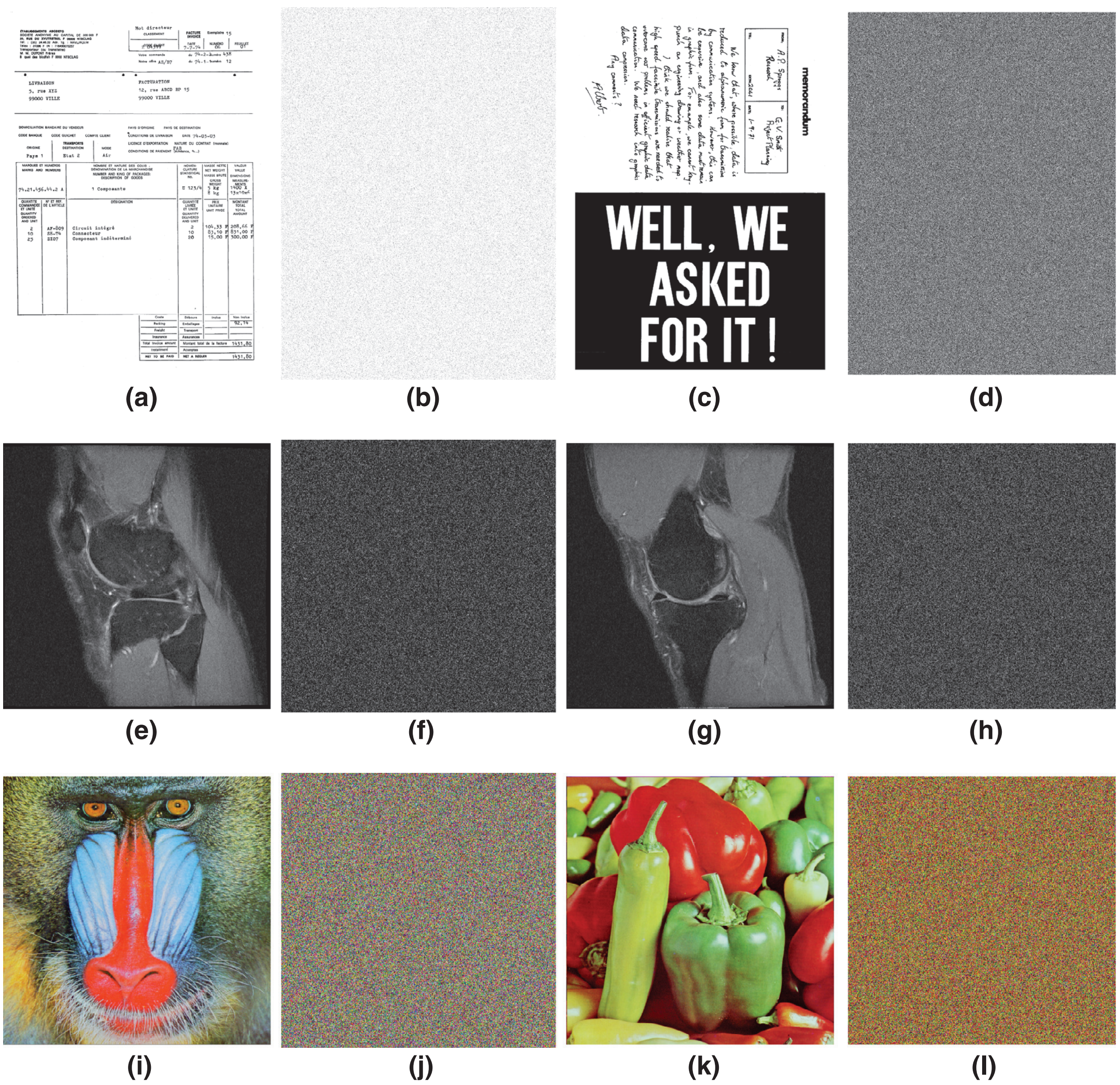


Fig. 11. Image scrambling results on various image types using Sudoku associated image scrambler; (a) and (b) *CCITT-3* before and after scrambling; (c) and (d) *CCITT-7* before and after scrambling; (e) and (f) *knee MRI sample A* before and after scrambling; (g) and (h) *knee MRI sample B* before and after scrambling; (i) and (j) *4.2.03* before and after scrambling, and (k) and (l) *4.2.07* before and after scrambling.

5.3. *Gray difference and gray degree of scrambling*

Gray difference and gray degree of scrambling (GDD) [28] are two measures used for quality evaluation of scrambled images. The gray difference for a $W \times H$ image X is defined as follows,

$$GD^X_{i,j} = \frac{1}{4} \sum_{p,q \in \{-1,+1\}} \left(X_{i,j} - X_{i+p,j+q}\right)^2 \tag{7}$$

Further, the mean gray difference of image X is computed by averaging all pixels except those on image edges as shown in Eq. (9).

$$EGD^X_{i,j} = \frac{\sum_{i=2}^{W-1} \sum_{j=2}^{H-1} GD^X_{i,j}}{(W-2)(H-2)} \tag{8}$$

Table 4
Test images information with execution speed.

	Test image information			Execution information	
Image file	Width	Height	Bit depth	Time(s)	Speed(KB/s)
Cameraman	256	256	8	2.8280	181.0467
Barbara	256	256	8	2.9266	174.9470
Lenna	256	256	8	3.0336	168.7764
#157055	321	481	8	5.0176	240.4053
#69015	481	321	8	4.6216	261.0044
#239096	321	481	8	4.6537	259.2040
CCITT-3	2339	1728	1	31.4804	125.3816
CCITT-7	2339	1728	1	32.8107	120.2980
Knee MRI Sample A	448	448	16	21.1253	148.4476
Knee MRI Sample B	448	448	16	21.1838	148.0377
4.2.03	512	512	24	31.1277	197.3805
4.2.07	512	512	24	32.4009	189.6244

Table 5
Gray scrambling degree of scrambled images.

Method	Test Images					
	Cameraman	*Barbara*	*Lenna*	#157055	#69015	#239096
Van De Ville et al.'s [20], 2004	0.1315	0.1930	0.1353	0.2752	0.5397	0.4502
Ye et al.'s [26], 2007	0.8832	N\A	0.9010	0.8780	0.8646	0.8976
Ye et al.'s [28], 2008	0.8926	0.8740	0.9311	0.8731	0.8789	0.9134
Abu Dalhoum et al.'s [4], 2012	0.8971	0.8749	0.9320	0.8821	0.8827	0.9388
Ye's [25], 2010	0.9118	0.9176	0.9618	0.9218	0.9406	0.9670
Ours	0.9165	0.9200	0.9663	0.9359	0.9439	0.9672

Finally, the gray degree of scrambling is defines as the ratio of the difference of the mean gray differences before and after scrambling image X to their sum, namely

$$GGD_{i,j}^{X,X'} = \frac{|EGD_{i,j}^{X} - EGD_{i,j}^{X'}|}{EGD_{i,j}^{X} + EGD_{i,j}^{X'}} \tag{9}$$

Table 5 compares the gray degree of scrambling of the proposed method with peer algorithms [4,20,25,26,28][5]. As can be seen, the proposed method outperforms the listed peer algorithm by achieving a higher gray degree of scrambling.

5.4. Adjacent pixel autocorrelation

A digital image is commonly of high information redundancy in the sense that adjacent pixels are strongly correlated. In contrast, a well scrambled image should break such correlations between neighbor pixels, so that the scrambled image is unrecognizable. For example, one can simply plot a pixel correlation figure as shown in Fig. 12, where each subfigure illustrating the adjacent pixel correlations before and after scrambling image *Lenna* by plotting the intensity values of 1024 random select pixels as x coordinates, intensity values of their horizontal neighbors as y coordinates, and intensity values of their vertical neighbors as z coordinates. It is obvious that before scrambling adjacent pixels in image *Lenna* are close correlated and thus the plots concentrates on the diagonal direction, namely the line $x = y = z$. In contrast, after scrambling the plot of the same set of adjacent pixels speared almost everywhere.

Adjacent pixel autocorrelation coefficient (APCC) is a common measure used in signal process and image encryption [22–24,27,30]. Mathematically, we can define this autocorrelation coefficient as follows:

$$\rho = \mathrm{E}[(X_t - \mu)(X_{t+1} - \mu)]/\sigma^2 \tag{10}$$

where X_t denotes the tth element in signal X, $\mathrm{E}(.)$ denotes the mathematica expectation as shown in Eq. (11), μ is the expectation namely the mean value defined in Eq. (12), and σ denotes the standard deviation defined in Eq. (13).

$$\mathrm{E}[Y] = \sum_{i=1}^{N} Y_i/N \tag{11}$$

$$\mu = \mathrm{E}[X] \tag{12}$$

[5] Results of Ye et al.'s [26], 2007, Ye et al.'s [28], 2008 and Abu Dalhoum et al.'s [4] are directly taken from listed results in [4].

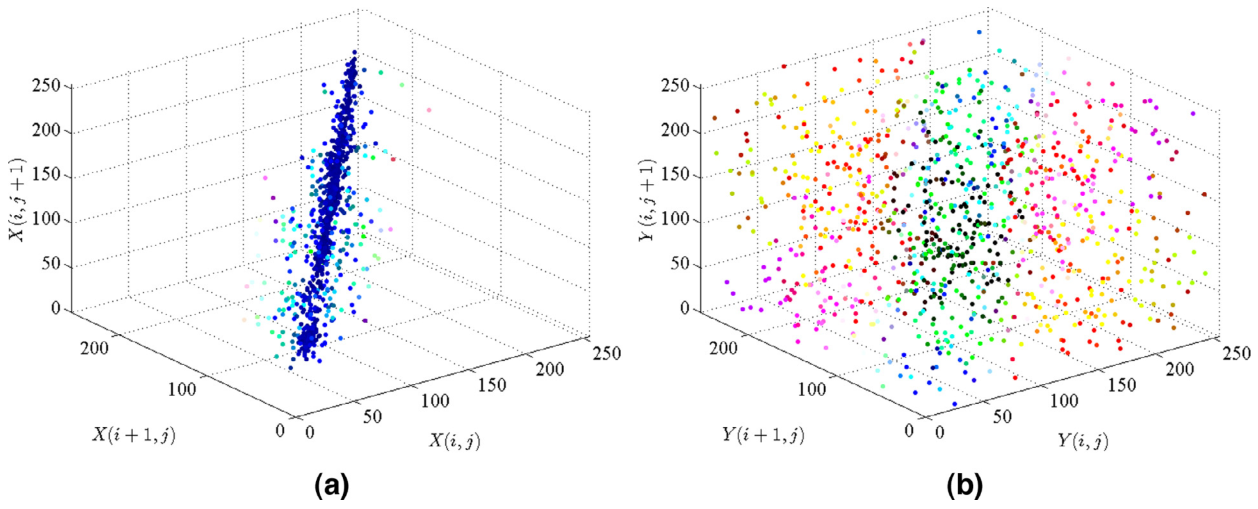


Fig. 12. Adjacent pixel correlations before and after Sudoku associated image scrambling for image *Lenna*; (a) before scrambling; and (b) after scrambling. Point colors reflect the distance between a point and the line $x = y = z$. The closer a point to this line, the bluer the color is.

Table 6
Horizontal adjacent pixel correlation coefficients of scrambled images.

Horizontal APCC	Test images					
method	*Cameraman*	*Barbara*	*Lenna*	#157055	#69015	#239096
Before scrambling	0.9565	0.9238	0.9709	0.9522	0.9613	0.9796
Van De Ville et al.'s [20], 2004	0.6967	0.6974	0.7156	0.6963	0.7099	0.7005
Abu Dalhoum et al.'s [4], 2012	0.0184	−0.0629	−0.0204	−0.0254	0.1564	0.1771
Ye's [25], 2010	0.0827	0.0398	0.2130	0.2270	0.0359	−0.0372
Ours	−0.0020	−0.0015	−0.0021	0.0024	−0.0025	−0.0017

Table 7
Vertical adjacent pixel correlation coefficients of scrambled images.

Vertical APCC	Test images					
method	*Cameraman*	*Barbara*	*Lenna*	#157055	#69015	#239096
Before scrambling	0.9333	0.8797	0.9400	0.9475	0.9570	0.9739
Van De Ville et al.'s [20], 2004	0.6613	0.6699	0.6741	0.6690	0.6785	0.6865
Abu Dalhoum et al.'s [4], 2012	0.0543	0.0172	0.0476	0.0376	0.2162	0.2314
Ye's [25], 2010	0.2808	0.0598	−0.0955	0.1015	0.0860	−0.0329
Ours	−0.0033	−0.0031	0.0016	−0.0004	0.0012	0.0040

$$\sigma = \sqrt{\mathrm{E}[(X-\mu)^2]} \tag{13}$$

Consequently, less correlated adjacent pixels within image X are, the closer ρ approaches zero. In contrast, if adjacent pixels within image X are completely dependent, then $|\rho| = 1$.

Commonly the adjacent pixels in image can be defined along the horizontal direction and the vertical direction, respectively. Equivalently, we extract scrambled image pixels along rows and along columns respectively and compare the autocorrelation coefficients for these two pixel sequences. Results are given in Tables 6 and 7. It is not difficult to see that the proposed method outperforms the three peer algorithms again.

It is also well-known that whether an observed autocorrelation coefficient ρ is significantly different from zero can be tested by *Student's t-distribution* [5,14,15]. Specifically in [14], it states that "if the true correlation between X and Y within the general population is zero, and if the size of sample T, on which an observed value ρ is based is equal to or greater than 6, then the quantity t defined by Eq. (14)

$$t = \rho\sqrt{\frac{T-2}{1-\rho^2}} \tag{14}$$

is distributed approximately as a *Student's t-distribution* with $T-2$ degrees of freedom", where *Student's t-distribution* has the probability density function given by Eq. (15), where v is the number of degrees of freedom and $\Gamma(.)$ is the Gamma function.

$$g(t)=\frac{\Gamma\left(\frac{v+1}{2}\right)}{\sqrt{v\pi}\,\Gamma\left(\frac{v}{2}\right)}\left(1+\frac{t^2}{v}\right)^{-\frac{v+1}{2}} \tag{15}$$

Therefore, given an autocorrelation coefficient of an $W\times H$ image X, it is not difficult to verify whether its adjacent pixels are correlated or not by taking the test statistic t for two-sided hypothesis tests. Specifically, one can calculate the P-value of a sample correlation coefficient ρ by finding its P-value under the null hypothesis that the derived statistic t from ρ follows a *Student's t-distribution*, where the P-value of a given test statistic t is computed as follows

$$\mathrm{P}-\mathrm{value}(t)=2\int_{-\infty}^{-|t|}g(\tau)d\tau \tag{16}$$

Statistically speaking, a P-value is a measure of how much evidence we have under the null hypothesis. In other words, the smaller P-value the more evidence we have against the hypothesis. The range of a P-value is [0, 1].

In our situation, the sample size T is the number of pixels within image X, which is $T=WH$. Consequently, test statistic t can be derived by the autocorrelation coefficient ρ and T using Eq. (14). As a result, corresponding P-value results can be calculated for the autocorrelation coefficients of the scrambled image using the proposed method. These results are shown in Table 8. It is clear that these P-values are much greater than 5%, which is a common critical value used in statistics implying to reject the

Table 8
P-values of adjacent pixel correlation coefficients.

	Test images					
Statistics	*Cameraman*	*Barbara*	*Lenna*	#157055	#69015	#239096
size	256×256	256×256	256×256	481×321	321×481	481×321
Degree of freedom v	65534	65534	65534	154399	154399	154399
APCC $\rho_{horizontal}$	−0.0020	−0.0015	−0.0021	0.0024	−0.0025	−0.0017
Statistic $t_{horizontal}$	−0.5120	−0.3840	−0.5376	0.9431	−0.9823	−0.6680
P-value $P_{horizontal}$	60.84%	70.10%	58.08%	34.56%	32.60%	50.42%
APCC $\rho_{vertical}$	−0.0033	−0.0031	0.0016	−0.0004	0.0012	0.0040
Statistic $t_{vertical}$	−0.8448	−0.7936	0.4096	0.1572	0.4715	−1.5718
P-value$P_{vertical}$	37.62%	42.74%	68.22%	87.50%	63.62%	11.60%

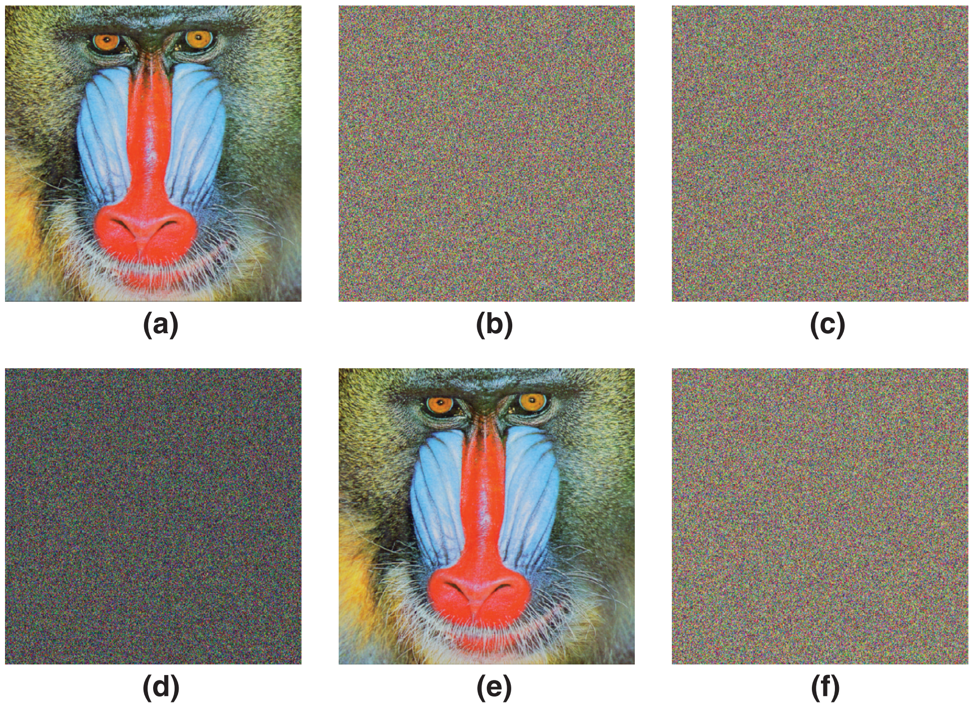


Fig. 13. Key sensitivities of *Sudoku Associated Image Scrambler*; (a) image *4.02.03*; (b) scrambled image Y_a using K_a; (c) scrambled image Y_b using K_b; (d) difference image of $|Y_a-Y_b|$; (e) descrambled image of Y_a using K_a; and (f) descrambled image of Y_a using K_b.

null hypothesis if a P-value is less than 5%, while accept the null hypothesis otherwise. Therefore, these results indicate that the correlation coefficient between adjacent pixels in the scrambled image using the proposed method are indeed zeros. In other words, the proposed method successfully breaks the strong correlations between adjacent pixels after scrambling.

5.5. Key space and key sensitivity

A good image scrambler should have a sufficiently large key space to resist *brute-force* attacks. *Sudoku Associated Image Scrambler* is designed to be of 192-bit length, which is considered sufficiently large to be immune to this type of attacks with respect to the current computer capacities. on the other hand, this key space can be easily extended because the number of possible Sudoku matrices are extremely huge. A lower bound of the number of 256 $\times$ 256 Sudoku matrices is $256! \approx 2^{1684}$. Considering that majority of digital image files are larger than 256 $\times$ 256, the number of Sudoku matrices at larger sizes are even huge.

In addition, a good scrambler should have high sensitivity to scrambler key. This has a two-folded meanings: a slight key change should lead to significant change in scrambled images during scrambling process, and also significant change in descrambled images during descrambling process. Fig. 13 shows the key sensitivity of *Sudoku Associated Image Scrambler*, with two keys K_a and K_b differentiate from each other only for one bit. As can be seen, one bit change in scrambling key leads to two different scrambled images, whose difference image is also random-like. And descrambling using an incorrect key which is just one bit different from the correct key leads to random-like image.

$$K_a = \text{B697F2703EA4347A85D997FB18A1FC3CE7E6901B6A9AE5EA}$$
$$K_b = \text{A697F2703EA4347A85D997FB18A1FC3CE7E6901B6A9AE5EA}$$

6. Conclusion

In this paper, we mainly discussed the Sudoku associated 2D bijections and applications for image scrambling. We showed that these bijections are can be defined by Sudoku associated matrix element representations, which provide additional and parametric means to denote matrix elements besides the conventional way of using the row-column pair. Specifically, these new Sudoku associated matrix element representations are row-digit pair, digit-row pair, column-digit pair, digit-column pair, block-digit pair, and digit-block pair.

Since all these Sudoku associated matrix element representations are parametric with respect to a reference Sudoku matrix, it then allows us to denote matrix elements in secret ways and further provides Sudoku matrix dependent 2D bijections constructed by mapping from one representation to the other. We showed that many of these Sudoku associated 2D bijections have deterministic scrambling effect when we use them for image scrambling. For example, the bijection mapping from row–column pair to row-digit pair is equivalent to scramble pixels within a row to different positions that none two pixels that originally lies in the same column is still in the same column after scrambling; the bijection mapping from block-grid pair to block-digit pair is equivalent to scramble pixels within each block and cause a mosaic-like effect.

Furthermore, we proposed *Sudoku Associated Image Scrambler*, a simple but effect digital image scrambler of using these Sudoku associated 2D bijections, by using a scrambling key to control these bijections in a parametric way. Because a multiround scrambler is mathematically equivalent to a new bijection composed of a series of bijections in each scrambler round, we showed that these fundamental Sudoku associated 2D bijections can be cascaded together to scramble image pixels in a deterministic way. Simulation results of the proposed image scrambler and comparisons to recent peer algorithms indicate that *Sudoku Associated Image Scrambler* outperforms or atleast reaches state-of-the-art suggested by these peer algorithms [4,25,26,28] with respect to a number of evaluation and analysis methods, including human visual inspections, gray degree of scrambling, and autocorrelation coefficient of adjacent pixels. Moreover, statistical tests also support that *Sudoku Associated Image Scrambler* does break the strong correlations between adjacent pixels to zero correlations after scrambling.

Acknowledgment

This work was supported in part by the Macau Science and Technology Development Fund under Grant FDCT/017/2012/A1 and by the Research Committee at University of Macau under Grants MYRG2014-00003-FST, MRG017/ZYC/2014/FST, MYRG113(Y1-L3)-FST12-ZYC and MRG001/ZYC/2013/FST.

References

[1] L. Aaronson, Sudoku science, IEEE Spectr. 43 (2) (2006) 16–17.
[2] R. Bailey, P.J. Cameron, R. Connelly, Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and hamming codes, Am. Math. Mon. 115 (5) (2008) 383–404.
[3] G. Chen, Y. Mao, C.K. Chui, A symmetric image encryption scheme based on 3D chaotic cat maps, Chaos, Solitons Fract. 21 (3) (2004) 749–761.
[4] A.L.A. Dalhoum, B.A. Mahafzah, A.A. Awwad, I. Aldhamari, A. Ortega, M. Alfonseca, Digital image scrambling method based on two dimensional cellular automata: a test of the lambda value, IEEE Multimed. 19 (4) (2011) 28–36.
[5] A. Donner, B. Rosner, On inferences concerning a common correlation coefficient, Appl. Stat. (1980) 69–76.
[6] L. Fang, W. YuKai, Restoring of the watermarking image in arnold scrambling, in: The 2nd International Conference on Signal Processing Systems (ICSPS), 2010, pp. 771–774.
[7] C. Fu, B. bin Lin, Y. sheng Miao, X. Liu, J. jie Chen, A novel chaos-based bit-level permutation scheme for digital image encryption, Opt. Commun. 284 (23) (2011) 5415–5423.

[8] Z. Hua, Y. Zhou, C.-M. Pun, C.L.P. Chen, 2D sine logistic modulation map for image encryption, Inf. Sci. 297 (0) (2015) 80–94.
[9] H. fen Huang, Perceptual image watermarking algorithm based on magic squares scrambling in DWT, in: Proceedings of the Fifth International Joint Conference on INC, IMS and IDC (NCM '09), 2009, pp. 1819–1822.
[10] G. Kendall, A.J. Parkes, K. Spoerer, A survey of np-complete puzzles., ICGA J. 31 (1) (2008) 13–34.
[11] Y. Li, X. Hao, A blind watermarking algorithm based on image scrambling and error correct coding preprocessing, in: Proceedings of the 2011 International Conference on Electrical and Control Engineering (ICECE), 2011, pp. 4231–4233.
[12] K. Lin, Hybrid encoding method by assembling the magic-matrix scrambling method and the binary encoding method in image hiding, Opt. Commun. 284 (7) (2011) 1178–1784.
[13] J. Lorch, Mutually orthogonal families of linear sudoku solutions, J. Aust. Math. Soc. 87 (03) (2009) 409–420.
[14] R. Lowry, Concepts and applications of inferential statistics, 2005.
[15] J. Macklin, An investigation of the properties of double radio sources using the spearman partial rank correlation coefficient, Mon. Not. R. Astron. Soc. 199 (1982) 1119–1136.
[16] Y. Mao, G. Chen, S. Lian, A novel fast image encryption scheme based on 3D chaotic baker maps, Int. J. Bifurc. Chaos 14 (10) (2004) 3613–3624.
[17] D. Martin, C. Fowlkes, D. Tal, J. Malik, A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics, in: Proceedings of the 8th International Conference on Computer Vision, 2, 2001, pp. 416–423.
[18] V. Patidar, N.K. Pareek, G. Purohit, K.K. Sud, A robust and secure chaotic standard map based pseudorandom permutation-substitution scheme for image encryption, Opt. Commun. 284 (19) (2011) 4331–4339.
[19] W.C. rong, L.J. Jing, L.G. Ying, A DCT-SVD domain watermarking algorithm for digital image based on moore-model cellular automata scrambling, in: Proceedings of the 2010 IEEE International Conference on Intelligent Computing and Integrated Systems (ICISS), 2010, pp. 104–108.
[20] D. Van De Ville, W. Philips, R. Van de Walle, I. Lemahieu, Image scrambling without bandwidth expansion, IEEE Trans. Circuits Syst. Video Technol. 14 (6) (2004) 892–897.
[21] Y. Wu, The Sudoku Array and Its Applications in Information Security, Tufts University, 2012 Ph.D. thesis.
[22] Y. Wu, J. Noonan, S. Agaian, Binary data encryption using the sudoku block cipher, in: Proceedings of the 2010 IEEE International Conference on Systems Man and Cybernetics (SMC), 2010b, pp. 3915–3921.
[23] Y. Wu, Y. Zhou, J.P. Noonan, S. Agaian, Design of image cipher using latin squares, Inf. Sci. 264 (0) (2014) 317–339.
[24] Y. Wu, Y. Zhou, J.P. Noonan, K. Panetta, S. Agaian, Image encryption using the sudoku matrix, 7708, SPIE, 2010, p. 77080P.
[25] G. Ye, Image scrambling encryption algorithm of pixel bit based on chaos map, Pattern Recogn. Lett. 31 (5) (2010) 347–354.
[26] G. Ye, X. Huang, C. Zhu, Image encryption algorithm of double scrambling based on ASCII code of matrix element, in: Proceedings of the 2007 IEEE International Conference on Computational Intelligence and Security, 2007, pp. 843–847.
[27] R. Ye, A novel chaos-based image encryption scheme with an efficient permutation-diffusion mechanism, Opt. Commun. 284 (22) (2011) 5290–5298.
[28] R. Ye, H. Li, A novel image scrambling and watermarking scheme based on cellular automata, in: Proceedings of the 2008 IEEE International Symposium on Electronic Commerce and Security, 2008, pp. 938–941.
[29] M. Zanin, A.N. Pisarchik, Gray code permutation algorithm for high-dimensional data encryption, Inf. Sci. 270 (0) (2014) 288–297.
[30] G. Zhang, Q. Liu, A novel image encryption method based on total shuffling scheme, Opt. Commun. 284 (12) (2011) 2775–2780.
[31] Y.-Q. Zhang, X.-Y. Wang, A symmetric image encryption algorithm based on mixed-linear nonlinear coupled map lattice, Inf. Sci. 273 (0) (2014) 329–351.
[32] Y. Zhou, K. Panetta, S. Agaian, C.L.P. Chen, Image encryption using P-Fibonacci transform and decomposition, Opt. Commun. 285 (5) (2012) 594–608.
[33] Z. Zhu, W. Zhang, K. Wong, H. Yu, A chaos-based symmetric image encryption scheme using a bit-level permutation, Inf. Sci. 181 (6) (2011) 1171–1186.