Journal of Information Hiding and Multimedia Signal Processing ©2011 ISSN 2073-4212
Ubiquitous International Volume **2**, Number **1**, January **2011**

A Pseudo-Random Pixel Rearrangement Algorithm Based on Gaussian Integers for Image Watermarking

Aleksey Koval, Frank Y. Shih, Boris S. Verkhovsky

Department of Computer Science,
New Jersey Institute of Technology, Newark, NJ 07102

Received May 2010; revised August 2010

Abstract. *This paper presents a pseudo-random pixel rearrangement algorithm to improve the security of most image watermarking techniques. Many published watermarking algorithms rely on methods of rearranging pixels. They often use chaotic maps as a part of the watermarking procedure. In this paper, we propose a new method of rearranging image pixels based on the properties of Gaussian integers. It results in a more random-looking image transformation that, in turn, significantly improves the security of the embedded watermark. The computation time is much better than the computation time of Arnold cat map chaotic transformation algorithm, used in methods previously published.*

Keywords: Gaussian integers, watermarking, steganography, Arnold Cat map

1. Introduction.

Steganography is a process of hiding information in a medium in such a manner that no one except the anticipated recipient knows of its existence ([1]). The history of steganography can be traced back to around 440 B.C.E, where the Greek historian Herodotus described in his writings about two events: one used wax to cover secret messages, and the other used shaved heads. With the explosion of internet as a carrier for various digital media, many new directions of this state-of-the-art emerged.

A notable application of steganography is watermarking of digital images, which is a useful tool for identifying the source, creator, owner, distributor, or authorized consumer of a document or an image. It has become very easy nowadays to copy or distribute digital images (whether copyrighted or not). A watermark is a pattern of bits inserted into a digital media for copyright protection ([2]). There are two kinds of watermarks: visible and hidden. A good visible watermark must be difficult for an unauthorized person to remove and can resist falsification. Since it is relatively easy to embed a pattern or a logo into a host image, we must make sure the visible watermark was indeed the one inserted by the author. In contrast, a hidden watermark is embedded into a host image by sophisticated algorithms and is invisible to the naked eye. It could, however, be extracted by a computer.

There are many innovating watermarking algorithms and many more get published every day (such as recently published [3, 4, 5, 6]). In many image watermarking algorithms, for example in [7, 8, 9, 10], it is required to rearrange the pixels as a part of watermarking process. Randomness is desired during this step. Modular arithmetic and, specifically, the integer exponentiation modulo prime numbers are widely used in modern cryptographic algorithms. One important property of integer exponentiation modulo prime is that it generates a sequence of integers that looks very much like a sequence of random numbers.

This is a property that is desirable for pixel rearrangement algorithms. In this paper we revisit the rearrangement step of watermarking algorithms and propose a different universal method for doing it. It is easy to replace the rearrangement step in [7, 8, 9, 10] with the method we propose in this paper. Moreover, the proposed method can be used with most image watermarking algorithms to enhance them.

One can look at Gaussian integers as an extension of real integers into two dimensions. They exhibit similar properties as regular integers but have some notable differences, that could be exploited in various fields, such as cryptography [11, 12, 13, 14]. One important difference is that they have a larger order for the same prime size, which provides the increased security.

In [8, 9], Arnold's cat map [15] was used to rearrange pixels for improving the performance of watermarking techniques. In this paper, we propose a replacement, namely, a novel pixel rearrangement algorithm based on Gaussian integers, to rearrange pixels in an image. We show that the new algorithm is superior to Arnold's cat map in both time complexity and security. The proposed technique is not a watermarking algorithm by itself but rather a universal enhancement to any existing watermarking algorithms. The technique tends to increase robustness to noise by uniformly distributing noise throughout the image. The increase in robustness depends on the watermarking algorithm enhanced by the proposed technique.

This paper is organized as follows. Section 2 introduces Gaussian integers. Section 3 describes the proposed pixel rearrangement algorithm. Section 4 discusses the cryptoimmunity of the rearrangement algorithm. Section 5 presents the comparison to Arnold's Cat map chaos transformation. Section 6 provides the proof of algorithm validity. Section 7 gives an example in image watermarking. Conclusion is drawn in Section 8.

2. Gaussian Integers.

A Gaussian integer is a complex number: $Z[i] = \{a + bi : a, b \in \mathbb{Z}\}$, where both a and b are integers. Gaussian integers, with ordinary addition and multiplication of complex numbers, form an integral domain. The norm of a Gaussian integer is a natural number, defined as $|a + bi| = a^2 + b^2$.

The prime elements of $Z[i]$ are also known as Gaussian primes, and can be divided into two subgroups. One subgroup consists of primes $P = (p, 0)$, where p is a real prime and $(p \bmod 4) = 3$, referred to as Blum primes. The second subgroup consists of primes $P = (a, b)$, where $|P|$ is a real prime and $(|P| \bmod 4) = 1$, referred to as non-Blum Gaussian primes.

There is one-to-one relationship between groups modulo real non-Blum primes and non-Blum Gaussian primes. Because of this fact, the algorithms based on such Gaussian integers do not offer any advantages over real integers. Hence, we should consider a subset of $Z[i]$: real primes p : $(p \bmod 4) = 3$ or Blum primes. This allows the following definition of modulo (mod) operation for Gaussian primes to be

$$A \bmod p = (a \bmod p) + (b \bmod p)i. \tag{1}$$

Equivalently, we can look at $A = a + bi$ as a vector $A = (a, b)$. Throughout this paper, the vector notation is used. Moreover, upper-case letters are used to denote Gaussian integers and lower-case letters to denote real integers.

The concept of the order for Gaussian integers is described in [16]. We call k an *order* of a Gaussian integer H if $H^k = (1, 0) \pmod{p}$. This is equivalent to stating that

$$\operatorname{ord}(H) \bmod p = k \tag{2}$$

We can apply the ideas in [17] to speed up multiplication of Gaussian integers. It takes three multiplications to multiply two different Gaussian integers. Below is the algorithm inspired by [17].

Algorithm 2.1. *Gaussian integer multiplication*
Given: Gaussian integers (a,b) and (c,d).
Output: Gaussian integer (x,y)=(a,b)(c,d).

$$v_1 := (a+b)(c+d); v_2 := ac; v_3 := bd; \tag{3}$$

$$x = v_2 - v_3; y = v_1 - v_2 - v_3; \tag{4}$$

It takes two multiplications to square a Gaussian integer:

$$(a,b)^2 = ((a+b)(a-b), ab+ab) \tag{5}$$

To find a generator for the Gaussian integer group modulo prime p, the following algorithm is sufficient (improvement is possible for certain cases [18]):

Algorithm 2.2. *Simple algorithm for finding Gaussian generators*
1. Factor $p^2 - 2$*.*

$$p^2 - 1 = (f_1)^{e_1}(f_2)^{e_2}...(f_k)^{e_k} \tag{6}$$

2. Select a $G = (a,b)$*, such that* $a \neq 0, b \neq 0$*, and* $a^2 \neq b^2 (\text{mod } p)$*.*
3. For each factor f_i *of* $p^2 - 1$*, compute*

$$B_i = G^{\frac{p^2-1}{f_i}} \text{ mod } p \tag{7}$$

If any of $B_i = (1,0)$ *mod p, then G is not a generator and go to Step 2. Otherwise, G is a generator.*

3. The Proposed Pixel Rearrangement Algorithm.

Algorithm 3.1. *Pixel rearrangement based on Gaussian integers*
Given: Image $I = (x,y)$ *of size* $m \times n$*;*
Output: Image $I' = (x', y')$ *of size* $m \times n$*;*
1. Generate a prime $p > \max(m,n)$ *and (p mod 4)=3.*
2. Find a Gaussian integer generator $G = (a,b)$ *mod p using Algorithm 2.2.*
3. Generate a random number s, such that $0 < s < p-1$*. Compute*

$$S = (s_x, s_y) = G^s \text{ mod } p \tag{8}$$

$$If \ s_x \le m \ or \ s_y \le n \tag{9}$$

is not satisfied, then compute

$$S = SG \text{ mod } p \tag{10}$$

until condition (9) is satisfied. Let $S = \{s_x, s_y\}$ *be the starting pixel.*
4. For each pixel $\{x_i, y_i\}$ *of image I, start with pixel*

$$C = (c_1, c_2) := S \tag{11}$$

```
for $i = 1$ to $m$
    for $j = 1$ to $n$
            $I'\{c_1, c_2\} := I\{i, j\}$        (12)
            $C := CG \text{ mod } p$        (13)
        while $c_1 < m$ or $c_2 > m$
            $C := CG \text{ mod } p$        (14)
        end-while
    end-for
end-for
```

Note that we need to save the last value of $C = (c_1, c_2)$ in order to rearrange back the pixels. Without the value of C, pixels could be rearranged back; however, it would require additional computation.

Algorithm 3.2. *Reverse of Algorithm 3.1*

```
$C_r := C$                                              (15)
for $i = m$ downto 1
    for $j = n$ downto 1
        $I\{i,j\} := I'\{c_1, c_2\}$                    (16)
        $C_r := C_r G^{-1} \bmod p$                     (17)
        while ($c_1 > m$ or $c_2 > m$)
            $C_r := C_r G^{-1} \bmod p$                 (18)
        end-while
    end-for
end-for
```

The time complexity of Algorithms 3.1 and 3.2 can be defined in terms of p. The most computationally expensive operations of the algorithm are (8), (13), and (14) inside the loop of Step 4 of Algorithm 3.1. Suppose that u is the time spent to multiply two integers of size p. The square-multiply algorithm is used for exponentiation, and Algorithm 2.1 is used to multiply two Gaussian integers. Therefore, the time complexity of (8) is:

$$O(3.5u \log_2 p) \tag{19}$$

Because the order of Gaussian integers is $p^2 - 1$, in Step 4 of Algorithm 3.1, $p^2 - 1$ multiplications are performed. Therefore, the number of multiplications required is:

$$O(3u(p^2-1)) = O(3up^2) \tag{20}$$

The total time complexity of Algorithm 3.1 is:

$$O(3u(p^2-1) + 3.5u \log_2 p) = O(up^2) \tag{21}$$

The complexity of integer multiplication u depends on the size of p. For small p, the most efficient algorithm is the naïve multiplication with time complexity of $O(l^2)$, where $l = \log_2 p$ is the size of p in bits. For a larger p, the multiplication algorithm in [17] is faster than the naïve method. The time complexity of Karatsuba multiplication is $O(3l^{1.585})$. For an even larger p, Toom-Cook (or Toom-3) algorithm is more efficient with a time complexity of $O(n^{1.465})$ [19]. The thresholds for the size of p vary widely with implementation details. However, it is reasonable to assume that most images would not be sufficiently large for Toom-Cook or Karatsuba multiplication. Therefore, we can assume that the naive multiplication method can be used and (21) becomes:

$$O(up^2) = O\left[(p \log_2 p)^2\right] \tag{22}$$

which is the time complexity of Algorithm 3.1. The time complexity for the Algorithm 3.2 is the same.

To minimize the time complexity, it is reasonable to select p close to $max(m.n)$. If p is selected in such a way, then the time complexity in terms of image size is

$$O\left\{[\max(m,n) \log_2 (\max(m,n))]^2\right\}. \tag{23}$$

The rearrangement algorithm described above is universal and can be used for many purposes. It can be applied for image watermarking as follows:

Algorithm 3.3. *Watermarking with pixel rearrangement based on Gaussian integers.*
1. Rearrange the image using Algorithm 3.1;
2. Apply the desired watermarking technique to the resulting rearranged image from Step 1;
3. Apply Algorithm 3.2 to the resulting image from Step 2.

Algorithm 3.4. *Extraction of the watermark applied by Algorithm 3.3.*
1. Rearrange the image using Algorithm 3.1.
2. Extract the watermark using the watermarking extraction technique in Algorithm 3.2. Note that in Algorithm 3.2, depending on watermarking technique, it may be possible to extract watermark and perform rearrangement on the watermark rather than on the image.

4. **Cryptoimmunity of the Rearrangement Algorithm.** From the properties of Gaussian integer group, we can estimate how hard it is for an adversary to obtain the original image from the rearranged one. The less an adversary knows about the algorithm and parameters, the harder it is to determine the original arrangement. It is reasonable to look at the following three cases:

Case 1. The adversary knows nothing about the rearrangement algorithm used, but he/she suspects that some kind of algorithm has been used. In this case, it is extremely hard for the adversary to figure out the original arrangement because there are too many possibilities. That is, there are $n!$ possible permutations, where n is the number of pixels in the image.

Case 2. The adversary knows that Algorithm 3.1 was used, but he/she does not know the parameters such as prime p, generator G, or private key s. In this case, the number of possible permutations for an image I of size $m \times n$ is:

$$\left(p^2-1\right)\left[\phi\left(p^2-1\right)\right], \tag{24}$$

where ϕ is the Euler's totient function ([20]).

The formula (24) does not include the complexity of guessing p. The reason for this is that it is too computationally expensive to use a large p (refer to (22)). For efficiency, p should be close to the image size. The prime p in (24) can be selected in such a way that $\phi(p^2-1)$ is maximized. To do this, we can select a prime with large prime divisors of $p+$ and $p-1$. For example,

$$p+1 = s_1 q_1 \tag{25}$$

and

$$p-1 = s_2 q_1 \tag{26}$$

where s_1 and s_2 are small integers, and q_1 and q_2 are primes close to p in size. In this case:

$$\phi(p^2-1) = \phi((p-1)(p+1)) = \phi(s_1 s_2)(q_1-1)(q_2-1) \tag{27}$$

and

$$o(\phi(p^2-1)) = o((q_1-1)(q_2-1)) = o(q_1 q_2) = o(p^2)) \tag{28}$$

Consequently, the approximate computational complexity of (24) is:

$$o((p^2-1)[\phi(p^2-1)]) = o(p^4) = o(\max(m,n)^4) \tag{29}$$

Case 3. The adversary knows Algorithm 3.1 used, prime p, and a generator G. In this case, the number of possible permutations is limited to

$$p^2-1. \tag{30}$$

While it may be unreasonable to assume that the adversary would not know Algorithm 3.1, there is no reason to make a prime p and a generator G known. Therefore, case 2 may be the most reasonable security estimate.

If increased protection is desired, Algorithm 3.1 could be applied several times on the same image. Suppose that Algorithm 3.1 was applied t times on image I of size $m \times n$. In this case, the number of possible permutations is:

$$o(\max(m,n)^{4t}) \tag{31}$$

while the time to compute the rearranged image would still be reasonable and be on the same order in terms of image size:

$$O\{t[\max(m,n)\log_2\max(m,n)]^2\} = O\{[\max(m,n)\log_2\max(m,n)]^2\}. \tag{32}$$

Therefore, one can achieve the desired level of security by increasing the time it takes to rearrange the image somewhat. Multiple rearrangements could provide a desirable and practical tradeoff.

5. Comparison to Arnold's Cat Map Chaos Transformation.

The Arnold's cat map transformation variation used in [8] is defined as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ l & l+1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \bmod N, \tag{33}$$

where N is the width of the square image. The possible values of l in (33) are $l : 1 < l < N - 2$. Therefore, the number of the transformations required is $O(N)$. It is reasonable to assume that N is small enough to call for naive multiplication algorithms. Thus, the multiplication time complexity is

$$O(\log_2^2 N), \tag{34}$$

and we have to perform it for every pixel (i.e. N^2 times). Therefore, the time complexity of Arnold's Cat Map is:

$$O(N^3\log_2^2 N) \tag{35}$$

Formula (35) should be compared with formula (23), assuming $N \approx \max(m,n)$. It is obvious that the computational complexity of Algorithm 3.1 described by (23) is much better than the that of Arnold's Cat map described by (35).

As far as security, it is obvious that there are only $o(N)$ possible permutations because $l : 1 < l < N - 2$. It is much smaller than $o(\max(m,n)^4)$ for Algorithm 3.1.

Another important advantage of Algorithm 3.1 is that the transformed image does not have any visible patterns. After rearrangement with this algorithm, the resulting image looks like random noise. The trasformation with Arnold's Cat map, on the other hand, preserves visible patterns. Figure 1 clearly illustrates this point. At every step of Arnold's Cat map transformation, C1-C7 patterns are clearly visible. The image B, on the othe hand, looks like random noise. Consequently, Algorithm 3.1, when used for watermaking, is far superior to Arnold's Cat map in terms of security and computational time.

6. Proof of Algorithm Validity.

The validity of the algorithms arises from the properties of Gaussian integer group. In this section we are going to describe and prove these properties. For any two complex numbers A and B, it is true that $|AB| = |A||B|$. Gaussian integer is a special kind of complex number, so it is true for Gaussian integers too. When we multiply Gaussian integer C by itself modulo p, we in turn multiply the norm of C mod p. This means that $|C^i|$ mod p ($i = 1, 2, ...$) will cycle with a period of ord($|C|$) mod p as illustrated in the examples below.

In addition, we can see that $C^{\text{ord}(|C|)}$ is a Gaussian integer with norm equal to 1 modulo p. In fact, the Gaussian integers $U : |U| = (1 \bmod p)$ form a cyclic subgroup with an

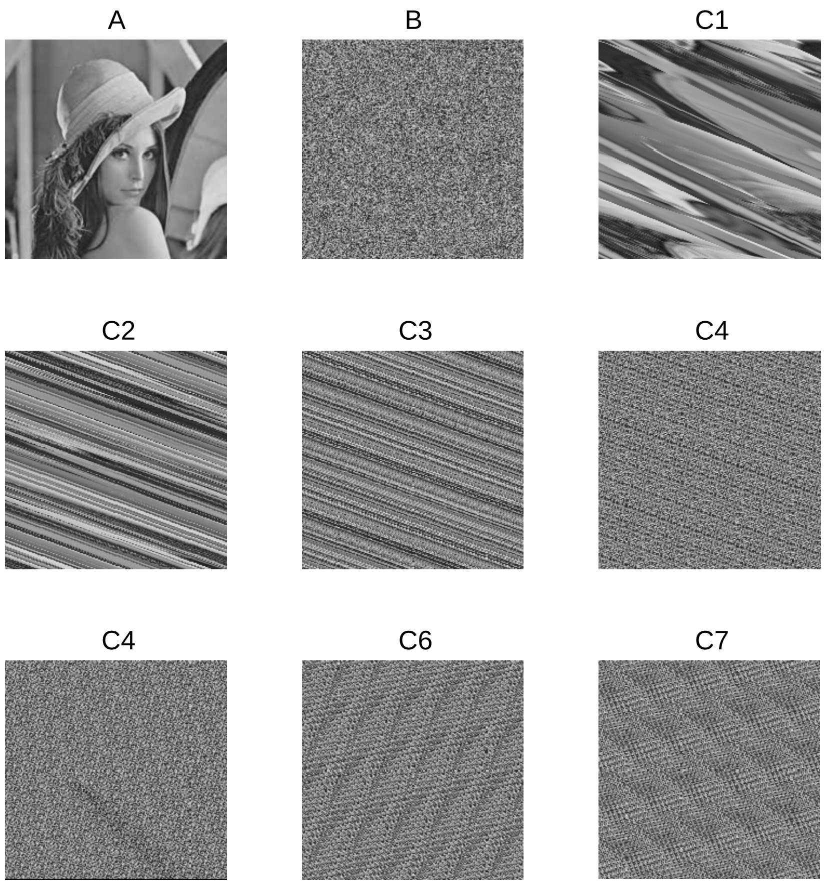

FIGURE 1. Image rearranged by Algorithm 3.1 and Arnold's Cat map side-by-side. A is the original image, B is the rearranged image by Agorithm 3.1, and C1-C7 are the steps of Arnold's Cat map rearrangement.

Example 1. *Repeating norm examples for prime p = 7*

Power:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Norm:	2	4	**1**	2	4	**1**	2	4	**1**	2	4	**1**	2	4	**1**	
	(1,6) [2]	(0,5) [4]	**(5,5) [1]**	(3,0) [2]	(3,4) [4]	**(0,1) [1]**	(1,1) [2]	(2,0) [4]	**(2,5) [1]**	(0,3) [2]	(3,3) [4]	**(6,0) [1]**	(6,1) [2]	(0,2) [4]	**(2,2) [1]**	(4,0) [2]
	(1,1) [2]	(0,2) [4]	**(5,2) [1]**	(3,0) [2]	(3,3) [4]	**(0,6) [1]**	(1,6) [2]	(2,0) [4]	**(2,2) [1]**	(0,4) [2]	(3,4) [4]	**(6,0) [1]**	(6,6) [2]	(0,5) [4]	**(2,5) [1]**	(4,0) [2]
Norm:	3	2	6	4	5	**1**	3	2	6	4	5	**1**	3	2	6	4
	(3,1) [3]	(1,6) [2]	(4,5) [6]	(0,5) [4]	(2,1) [5]	**(5,5) [1]**	(3,6) [3]	(3,0) [2]	(2,3) [6]	(3,4) [4]	(5,1) [5]	**(0,1) [1]**	(6,3) [3]	(1,1) [2]	(2,4) [6]	(2,0) [4]
	(4,6) [3]	(1,6) [2]	(3,2) [6]	(0,5) [4]	(5,6) [5]	**(5,5) [1]**	(4,1) [3]	(3,0) [2]	(5,4) [6]	(3,4) [4]	(2,6) [5]	**(0,1) [1]**	(1,4) [3]	(1,1) [2]	(5,3) [6]	(2,0) [4]

order $(p+1)$. We will refer to this subgroup as a $Norm$ 1 subgroup. Moreover, the order of any Gaussian integer C is a product of $\text{ord}(|C|)$ and $\text{ord}(|U|)$, where $U = C^{\text{ord}(|C|)}$ mod p. From this, we derive the algorithms for finding Gaussian generators to use for discrete logarithm based cryptography.

Lemma 6.1. *If C is a complex number and p is a prime, then*

$$|C^n| = |C|^n \bmod p \tag{36}$$

Example 2. Repeating norm examples for prime p=11

Power:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Norm:	2	4	8	5	10	9	7	3	6	**1**	2	4	8	5	10
	(3,2) [2]	(5,1) [4]	(2,2) [8]	(2,10) [5]	(8,1) [10]	(0,8) [9]	(6,2) [7]	(3,7) [3]	(6,5) [6]	**(8,5) [1]**	(3,9) [2]	(2,0) [4]	(6,4) [8]	(10,2) [5]	(4,4) [10]
	(10,1) [2]	(0,9) [4]	(2,2) [8]	(7,0) [5]	(4,7) [10]	(0,8) [9]	(3,3) [7]	(5,0) [3]	(6,5) [6]	**(0,1) [1]**	(10,10) [2]	(2,0) [4]	(9,2) [8]	(0,7) [5]	(4,4) [10]
Norm:	3	9	5	4	**1**	3	9	5	4	**1**	3	9	5	4	**1**
	(3,4) [3]	(4,2) [9]	(4,0) [5]	(1,5) [4]	**(5,8) [1]**	(5,0) [3]	(4,9) [9]	(9,10) [5]	(9,0) [4]	**(5,3) [1]**	(3,7) [3]	(3,0) [9]	(9,1) [5]	(1,6) [4]	**(1,0) [1]**
	(7,8) [3]	(7,2) [9]	(0,4) [5]	(1,6) [4]	**(3,6) [1]**	(6,0) [3]	(9,4) [9]	(9,1) [5]	(0,2) [4]	**(6,3) [1]**	(7,3) [3]	(3,0) [9]	(10,2) [5]	(10,6) [4]	**(0,1) [1]**

Proof: For any complex number it is true that $|C^n| = |C|^n$ so it must be true that $|C^n| = |C|^n \bmod p$

Lemma 6.2. *If C is a Gaussian integer and p is a Blum prime, then*
1) ord(C) mod p is divisible by ord(|C|) mod p
2) if $C^{ord(|C|)} = U$ mod p, then $|U| = 1$ mod p
3) if $U = C^{ord(|C|)}$ mod p, then ord(C) mod p is divisible by ord(U) mod p

Proof:
1) Suppose that ord(C) mod p is not divisible by $ord(|C|)$ mod p. This means that $|C^{ord(C)}|$ mod p is not equal to 1, but $C^{ord(C)} = (1, 0)$. This is a contradiction.
2) $|U|$ must equal to 1 mod p because $|C^n| = |C|^n$ mod p and, in this case, $n = ord(|C|)$.
3) If ord(C) mod p is not divisible by ord(U), then $C^{ord(C)}$ would not equal to (1,0), so ord(C) must be divisible by ord(U).

Lemma 6.3. *If U is a Gaussian Integer, p is a Blum prime, and $|U| = 1$ mod p, then*
1) the maximum order of U is (p+1), and
2) ord(U) mod p must divide (p+1).

Proof:
1) Any Gaussian integer A taken to the power (p+1) mod p is in the form (c, 0).
In our case U^{p+1} mod p could be one of either (1, 0) or $(-1, 0)$ because $|U| = 1$ mod p. Since $p + 1$ is divisible by 4 for all Blum primes, $U^{(p+1)/4}$ is a Gaussian integer of norm 1 and is a root of degree of U^{p+1}. For $(-1, 0)$, all roots of degree 4 have a norm equal to -1 mod p. This means that $U^{(p+1)}$ must equal to (1, 0) mod p.

Lemma 6.4. *If C is a Gaussian Integer and p is a Blum prime, then ord(C) = ord(—C—)ord(U) mod p, where*

$$U = C^{ord(|C|)} mod p \tag{37}$$

Proof: ord(C) must be divisible by ord($|C|$) and ord(U), so ord(C)=n*ord($|C|$)ord(U), where n is an integer. In addition, $C^{ord(|C|)ord(U)} = U^{ord(U)} = (1, 0)$. Consequently, n must equal to 1.

7. **An Example in Image Watermarking.**

Algorithm 3.1 can be used with general watermarking techniques. The following example illustrates its use of applying LSB substitution for watermark. Even though this technique does not provide a robust watermark, the use of rearrangement does improve the security by making the watermark virtually undetectable. When pixel rearrangement is used and the adversary looks at the last two bits of the watermarked image, all he/she sees is random noise. The only way to see the watermark is to rearrange the pixels.

Figure 2 illustrates the advantages of using the rearangement algorithm for image watermarking. In Figure 2, (a) is the original Cameraman image, (b) is the two most significant bits of the Lena image to used as the watermark, (c) is the rearranged image of Cameraman using Algorithm 3.1, (d) is the watermarked image of the rearranged image using LSB substitution, (e) is the rearranged back of the preceding watermarked image using Algorithm 3.2, (f) is the extracted 2 bits of LSB, and (g) is the rearranged back of the preceding extracted image using Algorithm 3.2. Note that image (g) is exactly the same as the original watermark (b).

If we perform watermarking without rearangement, then the hidden watermark is easily detectible. By using the proposed algorithms, it is impossible to see the original watermark in image (f), which is random noise just like images (c) and (d). It is fairly difficult for the adversary to extract the original watermark, even though her/she knows that the watermark is hidden there. With sequential applications of Algorithm 3.1, the security could be enhanced to an arbitrary level, making watermark practically impossible to reconstruct for the adversary.

8. **Conclusion.**

We propose a new method of rearranging image pixels for watermarking based on the properties of Gaussian integers. It results in such a more random-looking image transformation that significantly improves the security of the embedded watermark. Moreover, its speed is much faster as compared to the Arnold cat map. The proposed algorithm is an easy- to-implement practical technique that would enhance the security of any watermarking algorithm. It is flexible enough to offer variable levels of security.

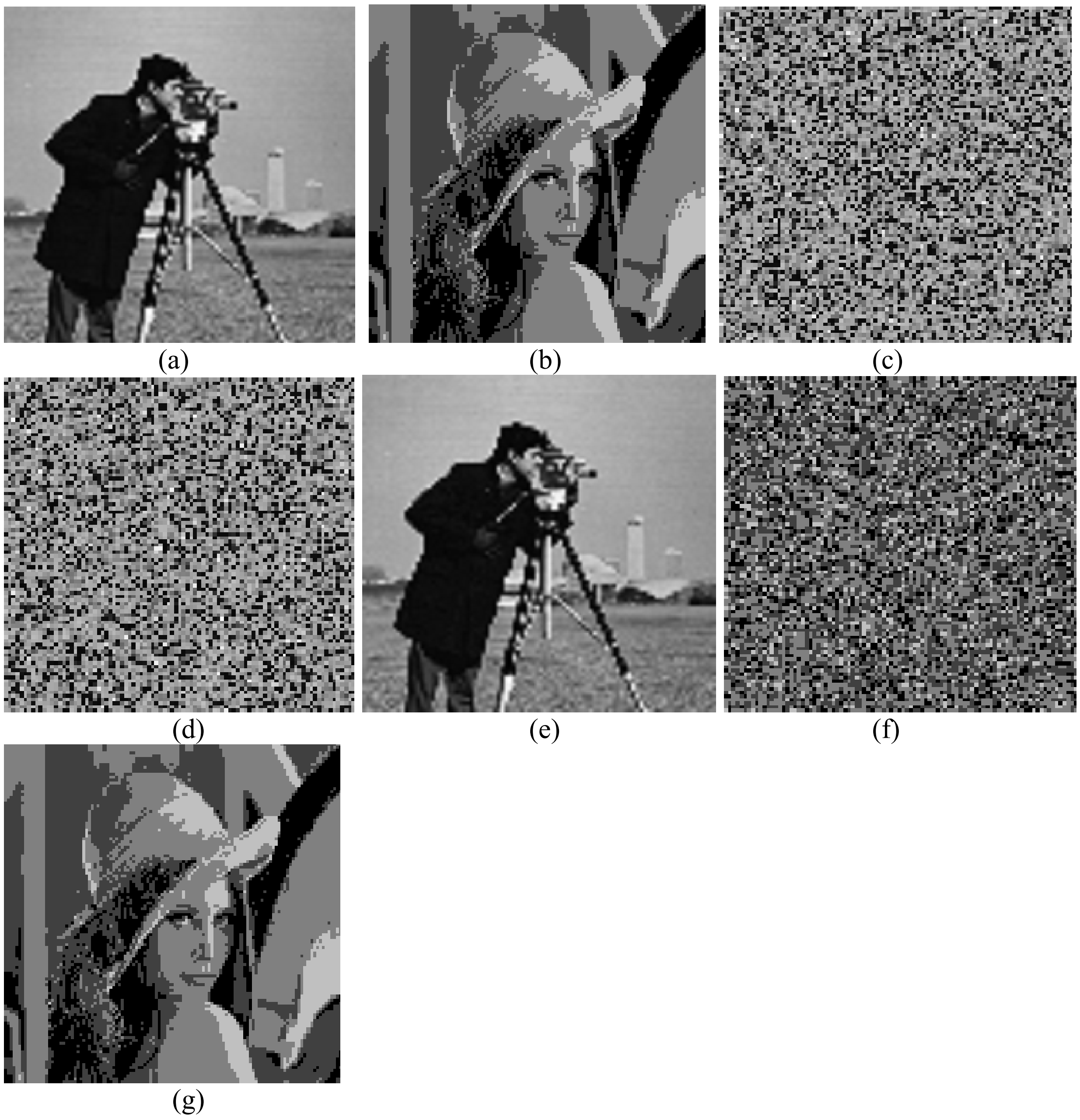

Figure 2. (a) The original Cameraman image, (b) the two most significant bits of Lena as the watermark, (c) the rearranged image of Cameraman using Algorithm 3.1, (d) the watermarked image of the rearranged image using LSB substitution, (e) the rearranged back of the preceding watermarked image using Algorithm 3.2, (f) the extracted 2 bits of LSB (g) the rearranged back of the preceding extracted image using Algorithm 3.2.

REFERENCES

[1] F. Y. Shih (eds.) , *Digital Watermarking and Steganography: Fundamentals and Techniques*, Taylor & Francis Group, CRC Press., Inc., Boca Raton, FL, USA, 2008.

[2] H. Berghel, and L. O'Gorman, Protecting ownership rights through digital watermarking, *Proc. of IEEE Comput. Mag.*, vol. 29, no. 7, pp. 101-103, 1996.

[3] H. Al-Qaheri, A. Mustafi, and S. Banerjee, Digital watermarking using ant colony optimization in fractional fourier domain, *Journal of Information Hiding and Multimedia Signal Processing*, vol. 1, no. 3, pp. 220-240, 2010.

[4] H. C. Huang, Y. H. Chen, and A. Abraham, Optimized watermarking using swarm- based bacterial foraging, *Journal of Information Hiding and Multimedia Signal Processing*, vol. 1, no. 1, pp. 51-58, 2010.

[5] C. C. Lin, and P. F. Shiu, Highcapacity data hiding scheme for dct-based images, *Journal of Information Hiding and Multimedia Signal Processing*, vol. 1, no. 3, pp. 220- 240, 2010.

[6] K. Yamamoto, and M. Iwakiri, Real-time audio watermarking based on characteristics of pcm in digital instrument, *Journal of Information Hiding and Multimedia Signal Processing*, vol. 1, no. 2, pp. 59-71, 2010.

[7] Z. Dawei, C. Guanrong, and L. Wenbo, A chaos-based robust wavelet-domain watermarking algorithm, *Chaos, Solitons and Fractals*, vol. 22, no. 1, pp. 47-54, 2004.

[8] Y. Wu, and F. Y. Shih, Digital watermarking based on chaotic map and reference register, *Pattern Recognition*, vol. 40, no. 12, pp. 3753-3763, 2007.

[9] Z. Yantao, M. Yunfei, and L. Zhiquan, A robust chaos-based dct-domain watermarking algorithm, *Proc. of 2008 International Conference on Computer Science and Software Engineering*, vol. 3, pp. 935-938, 2008.

[10] G. Ye, Image scrambling encryption algorithm of pixel bit based on chaos map, *Pattern Recognition Letters*, vol. 31, no. 5, pp. 347-354, 2010.

[11] H. Elkamchouchi, K. Elshenawy, and H. Shaban, Extended RSA cryptosystem and digital signature schemes in the domain of gaussian integers, *Proc. of the 8th International Conference on Communication Systems*, vol. 1, pp. 91-95, 2002.

[12] A. El-Kassar, M. Rizk, N. Mirza, and Y. Awad, El-gamal public-key cryptosystem in the domain of gaussian integers, *Int J Appl Math*, vol. 7, no. 4, pp. 405-412, 2001.

[13] A. N. El-Kassar, R. A. Haraty, Y. A. Awad, and N. C. Debnath, Modified RSA in the domains of gaussian integers and polynomials over finite fields, *Proc. of the 18th International Conference on Computer Applications in Industry and Engineering*, Sheraton Moana Surfrider, Honolulu, Hawaii, USA, pp. 298-303, 2005.

[14] B. Verkhovsky, and A. Koval, Cryptosystem based on extraction of square roots of complex integers, *Proc. of the 5th International Conference on Information Technology: New Generations (ITNG 2008)*, Las Vegas, Nevada, USA, vo. l0, pp. 1190-1191, 2008.

[15] V. I. Arnold, and A. Avez, *Ergodic Problems in Classical Mechanics*, Benjamin, New York, 1968.

[16] J. T. Cross, The Euler's Φ-function in the gaussian integers, *Amer. Math.*, vol. 55, pp. 518-528, 1983.

[17] A. Karatsuba, and Y. Ofman, Multiplication of many-digital numbers by automatic computers, *Proc. of USSR Academy of Sciences*, vol. 145, pp. 293-294, 1962.

[18] B. S. Verkhovsky, and S. Sadik, Accelerated search for gaussian generator based on triple prime integers, *J. of Computer Science*, vol. 5, no. 9, pp. 614-618, 2009.

[19] D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley, 1998.

[20] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, Dover, New York, 1964.