《混沌映射与比特重组的图像加密》一文的研究和算法重现

作者姓名：吴彬　作者学号：2017061033

目录

摘要　当前很多图像加密都采用基于比特的加密算法，针对这种比较流行的加密算法所存在的安全缺陷问题，《混沌映射与比特重组的图像加密》[1]一文提出了一种能够解决比特面 0 比特和 1 比特置乱时的位置限制的图像加密算法。

具体做法是，首先利用 Tent 混沌映射生成一个伪随机序列，然后利用生成的伪随机序列对比特明文图像进行整行以及整列的置乱，将置乱后的比特像素矩阵分块分别进行 Henon 映射的置乱，最后经过扩散操作得到最后的密文图像。

本文首先对上面的算法成果进行了勘误，主要是指出原文的加密算法只适用于大小为 $N \times N$ 的明文图像，应用到高不等于宽的图像时可能会出错，在此基础上往原算法中添加了基于辗转相除法的滑动窗口操作，使得算法可以应用到任意有限大小的图像上。然后对原文算法背后的一些理论和算法的一些步骤作了分析。

此外本文还仿照原论文[1]中的操作进行了一系列仿真实验，结果表明该加密算法得到的密文图像的像素值分布由不均匀变成了均匀分布，明文图像的各灰度级之间的相关性被打破，使得原图没有了统计特性，能够有效抵御统计分析的攻击。而解密算法得到的解密图像与原明文图像完全一致，即该加密算法和解密算法是无损的。

而本文仿照原论文[1]中的操作进行的一系列性能分析也表明加密算法对明文具有很强的敏感性，加密算法对密钥具有很强的敏感性，解密算法对部分密钥具有很强的敏感性，算法能够抵抗差分攻击，相邻像素相关系数，信息熵指标都接近理想值。

关键词：图像加密，图像解密，Tent 映射，比特置乱，Henon 映射

§1 引言

互联网技术的迅猛发展极大地促进了数字图像的传输量，这些数字图像很多都涉及个人，企业，军事等各方面的安全隐私，因此，图像的安全性已经成为了一个各界广泛关注的重要问题。图像加密是解决各种图像安全问题的一种有效的办法。在过去的十几年，许多经典的图像加密算法已经被提出。这些算法主要有两种，分别是基于像素的图像加密算法，以及基于比特的图像加密算法。

对于基于像素的图像加密算法，根据它们的体系结构可以分为 3 种主要的算法类型，分别是只进行像素的置乱，只进行灰度扩散，以及置乱和扩散都进行的 3 种算法类型。只进行像素位置的置乱由于算法的计算复杂性比较低，算法的效率相对较高，但是这种算法只是改变了像素的位置而没有改变像素的值，置乱后图像的直方图不变，算法很容易受到统计分析的攻击。而基于置乱-扩散的图像加密算法就可以解决这种问题，因为算法中增加了对像素值的改变，增强了加密系统的安全性。在基于像素的图像加密算法的这类论文中，文献[2]提出一种典型的基于置乱-扩散的图像加密算法，它是利用基于多种混沌映射的自适应模型实现对图像的像素的置乱以及扩散。文献[3]则是将 Conway 提出的生命游戏用于像素置乱，但是由于算法计算量大，效率会受到很大的影响。

而基于比特的图像加密算法能在改变像素位置的同时改变像素值。在这类算法的论文中，文献[4]提出了一种新颖的比特级图像加密算法，基本思想是通过两个阶段置乱操作对图像进行加密。第 1 阶段的置乱时利用 Chebyshev 映射所产生的的混沌序列，第 2 阶段的置乱是利用 Arnold Cat 映射。然而该算法存在以下问题：①第 1 阶段置乱中利用 Chebyshev 映射所生成的混沌序列与明文图像无关，当密钥初始值不变时，加密过程使用相同的密钥流序列；②置乱所利用的混沌序列是唯一的，即图像所有列的像素值都是按照同一个混沌序列进行置乱的；③算法只有置乱过程没有扩散过程。在文献[5]中，明文图像首先被转换成一维的比特序列，然后利用 Logistic 映射生成一个与明文序列等长的一个伪随机序列，通过伪随机序列来对比特序列进行排序置乱，这类算法的优点是容易设计和实现，但缺点是比较耗时，算法执行效率比较低。文献[6]中，算法利用像素转化成 8 位比特后，高 4 位包含原图信息的 94.125%这一特点，只对高 4 位的比特面进行单独置乱，低 4 位的比特面进行合并置乱，然后组合成一幅完整的置乱后的密文图像，最后对置乱后的密文图像进行扩散操作，得到最终的密文图像。但是在这个算法中，对于高 4 位的每个比特面，它们的统计信息在置乱后没有改变，每个比特面的 0 比特和 1 比特的比重没有发生改变。文献[7]所提出的算法解决了文献[6]算法的问题，由于相邻比特面之间存在着很强的相关性，基于比特级的置乱就会限制经过置乱后每个比特的目标位置，所以每个比特只能在同一个比特平面内移动位置。文献[7]提出了一种“胀缩”的方法来部分消除这种限制，通过让原来属于一个偶数比特面的比特移动到另一个偶数比特面的目标位置，奇数比特面也是同样的移动方式，从而实现了改变比特面的 0 比特和 1 比特的比重。

基于上述讨论，考虑到：

① 学习文献[2]，令加密算法既有置乱环节又有扩散环节，从而增强算法对统计分析攻击的抵抗性，避免出现如文献[4]中只有置乱环节没有扩散环节；

② 学习文献[2]，在置乱过程中使用多种混沌序列，避免出现如文献[4]中置乱用混沌序列单一的问题；

③ 避免设计出过于简单粗糙的置乱规则（如文献[5]）或过于复杂的置乱规则（如文献[3]）使得算法计算量大，效率低下；

④ 将置乱用混沌序列与明文图像产生关联，从而增强明文敏感性，避免如文献[4]一样混沌

序列与明文图像无关；

⑤ 完全解除比特置乱的限制，避免出现如文献[6]一样比特置乱被完全限制在同一比特面，或者如文献[7]一样比特置乱限制的解除只在高 4 位的比特面，

论文[1]提出了一种新型的基于混沌映射与比特重组的图像加密算法。算法的核心思想如下：利用 Tent 混沌映射产生两组混沌序列，利用这两组混沌序列对转成比特的明文图像分别进行整行与整列置乱，然后将置乱后的密文图像分块，利用 Henon 混沌映射分别对 8 个比特面进行置乱，最后经过扩散操作得到最终密文图像。

§ 2 加解密算法的研究

§ 2.1 原文算法步骤与勘误

原文只给出加密算法的步骤，这里引述如下（附有一些本人的注记）。

第 1 阶段置乱

（1）选取一幅大小为 $M \times N$ 的灰度级数字图像，计算图像中像素值的总和记作 s，然后利用式(2.1.1), (2.1.2)分别计算出 Tent 混沌系统的控制参数 μ 和 Tent 混沌系统初始迭代次数 k，即

$$\mu = 2^{\frac{s}{M \times N \times 255}}, \tag{2.1.1}$$

$$k = s \mod 10^3 + 10^3. \tag{2.1.2}$$

式中 $s/(M \times N \times 255)$ 的结果不取整。Tent 混沌系统的系统方程定义为

$$x(n+1) = \begin{cases} \mu \bullet x(n), & 0 < x(n) \le 0.5, \\ \mu \bullet (1 - x(n)), & 0.5 < x < 1. \end{cases} \tag{2.1.3}$$

其等价形式是

$$x(n+1) = \mu \bullet \min\{x(n), 1 - x(n)\}. \tag{2.1.4}$$

要求 $x(n) \in (0,1)$，$\mu \in (0,2)$. 当 $\mu > 1$ 时，系统处于混沌状态。

（2）将像素矩阵中的每个像素转换成 8 位二进制数，例如将 123 转换成 8 位二进制是 01111011.

（3）输入初始密钥 x_0，并根据步骤（1）求出的控制参数 μ，对 Tent 混沌系统进行 k 次迭代，消除初态效应的影响。

（4）Tent 混沌系统继续迭代 M 次，由此产生长度为 M 的混沌序列 $E = \{e_1, e_2, \cdots, e_M\}$，产生的混沌序列的值均在 0 到 1 之间。

（5）将步骤（4）生成的序列 E 按升序排序，从而得到一个位置向量 $P^E=\left\{p_1{}^E, p_2{}^E, \cdots, p_M{}^E\right\}$，利用生成的位置向量 P^E，对已经转成比特的数字图像矩阵进行整行置乱。比如任意序列 $\{0.3,\ 0.7,\ 0.5,\ 0.4,\ 0.8,\ 0.2\}$，将该序列按升序进行排序后得到有序序列是 $\{0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.7,\ 0.8\}$，则相应的位置序列就是 $\{6,\ 1,\ 4,\ 3,\ 2,\ 5\}$。下图为采用该位置序列的一个整行置乱示意图。

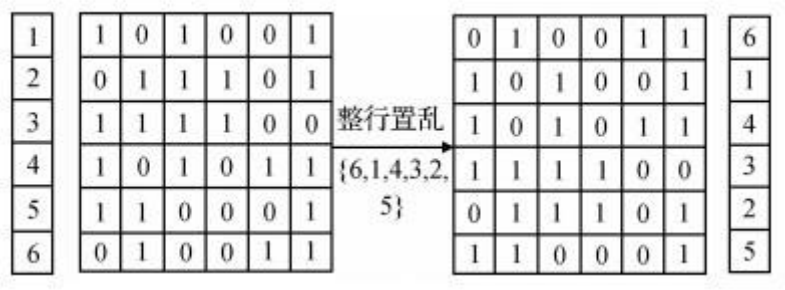


Fig 2.1.1 整行置乱示意图.

（6）Tent 混沌系统继续迭代 $8\times N$ 次，由此产生的长度为 $8\times N$ 的混沌序列 $F=\left\{f_1, f_2, \cdots, f_M\right\}$，将序列 F 按升序排序之后得到相应的位置向量 $P^F=\left\{p_1{}^F, p_2{}^F, \cdots, p_M{}^F\right\}$，利用 P^F 对数字图像矩阵进行整列置乱。

第 2 阶段置乱

（1）Tent 混沌系统继续迭代 $M\times N$ 次，由此产生的长度为 $M\times N$ 的混沌序列 $R=\left\{r_1, r_2, \cdots, r_{M\times N}\right\}$。

（2）将第 1 阶段得到的置乱矩阵从左到右分成 8 个 $M\times N$ 的比特矩阵，对 8 个矩阵分别使用 Henon 映射

$$\begin{cases} x(n+1)=1-ax^2(n)+y(n) \mod N, \\ y(n+1)=bx(n)+c \mod N. \end{cases} \tag{2.1.5}$$

进行置乱，式(2.1.5)中控制参数 a_i，$c_i\left(i=1,2,\cdots,8\right)$ 为

$$a_i=Ceiling\left(f_{N/2+i}\times 10^{14}\right) \mod 2^8, \tag{2.1.6}$$

$$c_i=Ceiling\left(f_{N/2+i}{}^2\times 10^{14}\right) \mod 2^8. \tag{2.1.7}$$

式(2.1.5)中控制参数 b 的取值为 1（为了使混沌系统是可逆的），$Ceiling(\ )$ 函数表示向上取整。

（3）对于每个比特矩阵中的比特位 (x, y)，根据式(2.1.5)计算出比特位新的位置 (x', y')，

然后将比特位(x,y)移动到(x',y')。

（4）确定每个比特矩阵进行 Henon 映射迭代的次数$n_i\ (i=1,2,\cdots,8)$

$$n_i = Ceiling\left(f_{N/2+i}\times 10^{14}\right) \bmod 5+1. \tag{2.1.8}$$

每个比特矩阵根据迭代的次数进行迭代，最后将 8 个比特矩阵合并，将比特转化为十进制像素值，即中间密文图像C'。

第 3 阶段扩散

得到中间密文图像后，再对C'进行加密得到最终的密文图像C，即

$$D_i = Ceiling\left(r_i\times 2^{48}\right) \bmod 2^8, \tag{2.1.9}$$

$$C_i = \left[\left(D_i + C_i'\right) \bmod 2^8\right]\oplus C_{i-1}. \tag{2.1.10}$$

当$i=1$时为

$$C_0 = S \bmod 2^8. \tag{2.1.11}$$

参数S作为密钥，取值为正整数。

不过原文给出的加密算法存在原理或表述上的一些小瑕疵。以下简单给出几点。

（1）第 1 阶段置乱的步骤（1）中，选取图像的大小为$M\times N$，而参考式(2.1.5)，原文给出的可逆 Henon 映射表达式中第一式和第二式中模数均为N（而不是M和N），分别对应图像的高和宽。因此基于原文给出的资料和描述，只能确定该算法适用于大小为$N\times N$的方形图像。而事实上，考虑在一般大小的图像上进行 Henon 映射时，自然地将 Henon 映射表达式改为

$$\begin{cases} x(n+1) = 1-ax^2(n)+y(n) \bmod M, \\ y(n+1) = x(n)+c \bmod N. \end{cases} \tag{2.1.12}$$

当$M\neq N$时可能会出问题。事实上，当$M<N$时，将图像像素横坐标$x(n)=0,1,\cdots,M-1$代入第二式，得到的$y(n+1)$无法覆盖到所有的纵坐标$0,1,\cdots,N-1$，这样会导致后续第 2 阶段置乱的步骤（2）中经 Henon 映射得到的图像的某些像素是“空的”。综上论述，整个算法流程应该严格要求图像尺寸为$N\times N$，相应地，μ的计算式应改为

$$\mu = 2^{\frac{s}{N^2\times 255}}.$$

后面的序列$E=\{e_1,e_2,\cdots,e_M\}$改为$E=\{e_1,e_2,\cdots,e_N\}$，向量$P^E=\{p_1{}^E,p_2{}^E,\cdots,p_M{}^E\}$改为$P^E=\{p_1{}^E,p_2{}^E,\cdots,p_N{}^E\}$，序列$R=\{r_1,r_2,\cdots,r_{M\times N}\}$改为$R=\{r_1,r_2,\cdots,r_{N\times N}\}$。

（2）第 1 阶段置乱步骤（6）中表述出现失误，应该将$F=\{f_1,f_2,\cdots,f_M\}$改为

$F=\{f_1,f_2,\cdots,f_{8\times N}\}$，将 $P^F=\{p_1{}^F,p_2{}^F,\cdots,p_M{}^F\}$ 改为 $P^F=\{p_1{}^F,p_2{}^F,\cdots,p_{8\times N}{}^F\}$。

（3）第 2 阶段置乱步骤（2）中，a_i, c_i 的表达式中 $f_{N/2+i}$ 的下标 $N/2+i$ 不妥当，因为 $\frac{N}{2}$ 不一定是整数。这里可将 $f_{N/2+i}$ 调整为 $f_{N\bullet(i-1)+1}$。第 2 阶段置乱步骤（4）中的 n_i 计算式中的 $f_{N/2+i}$ 作出同样的调整。

（4）第 3 阶段扩散中，式(2.1.10)中的符号 $\oplus$ 意义不明，查阅了整篇原文也没有任何说明。因此这里将 $\oplus$ 理解为普通加法 $+$，为了保证 $0\le C_i<256$，在 C_{i-1} 后再加上一个 $\bmod 2^8$，此时式(2.1.10)变成

$$C_i=\left[\left(D_i+C_i'\right)\ \bmod 2^8\right]+C_{i-1}\bmod 2^8. \tag{2.1.13}$$

利用同余式的性质，可将式(2.1.13)化简为

$$C_i=D_i+C_i'+C_{i-1}\ \bmod 2^8. \tag{2.1.14}$$

§ 2.2 勘误和整理后的加解密算法步骤

加密算法步骤（需要两个密钥 x_0, S）

（1）选取一幅大小为 $N\times N$ 的灰度级数字图像 I，规定图像 I 最左上角的像素坐标为 $(1,1)$。根据式(2.2.1)计算图像中像素值的总和 s。

$$s=\sum_{j=1}^{N}\sum_{i=1}^{N}I(i,j). \tag{2.2.1}$$

然后根据式(2.2.2), (2.2.3)分别计算出 Tent 混沌系统的控制参数 μ 和 Tent 混沌系统初始迭代次数 k。

$$\mu=2^{\frac{s}{N\times N\times 255}}, \tag{2.2.2}$$

$$k=10^3+\left(s\ \bmod 10^3\right). \tag{2.2.3}$$

（2）输入初始密钥 x_0，根据步骤（1）中求出的控制参数 μ，对 Tent 混沌系统进行 k 次迭代，消除初态效应的影响。

（3）Tent 混沌系统继续迭代 N 次，产生长度为 N 的混沌序列 $E=\{e_1,e_2,\cdots,e_N\}$。

（4）Tent 混沌系统继续迭代 $8\times N$ 次，产生长度为 $8\times N$ 的混沌序列 $F=\{f_1,f_2,\cdots,f_{8\times N}\}$。

（5）Tent 混沌系统继续迭代 $N\times N$ 次，产生长度为 $N\times N$ 的混沌序列 $R=\{r_1,r_2,\cdots,r_{N\times N}\}$。

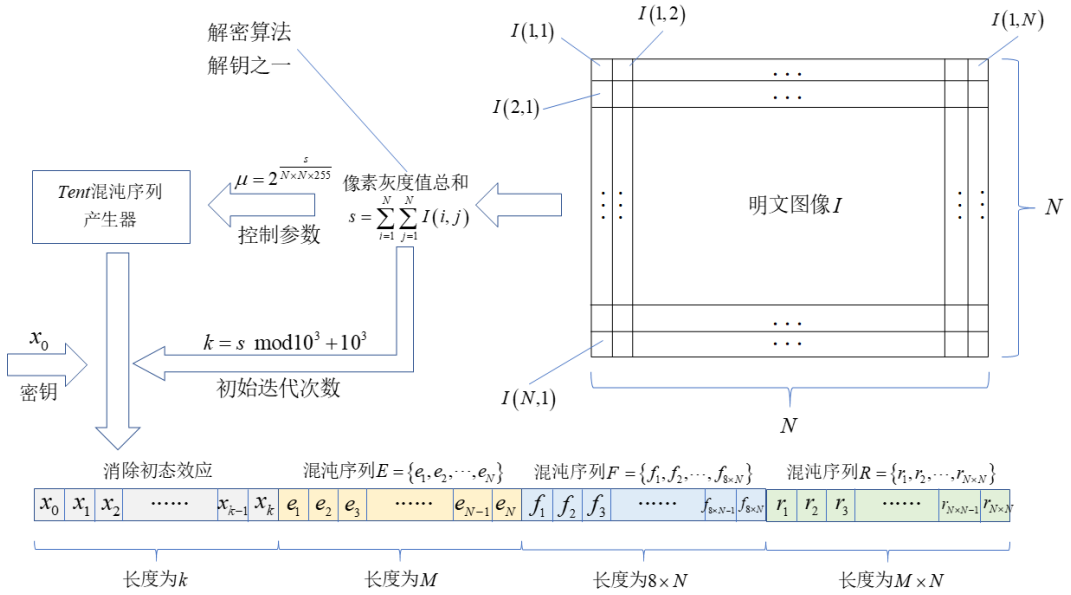
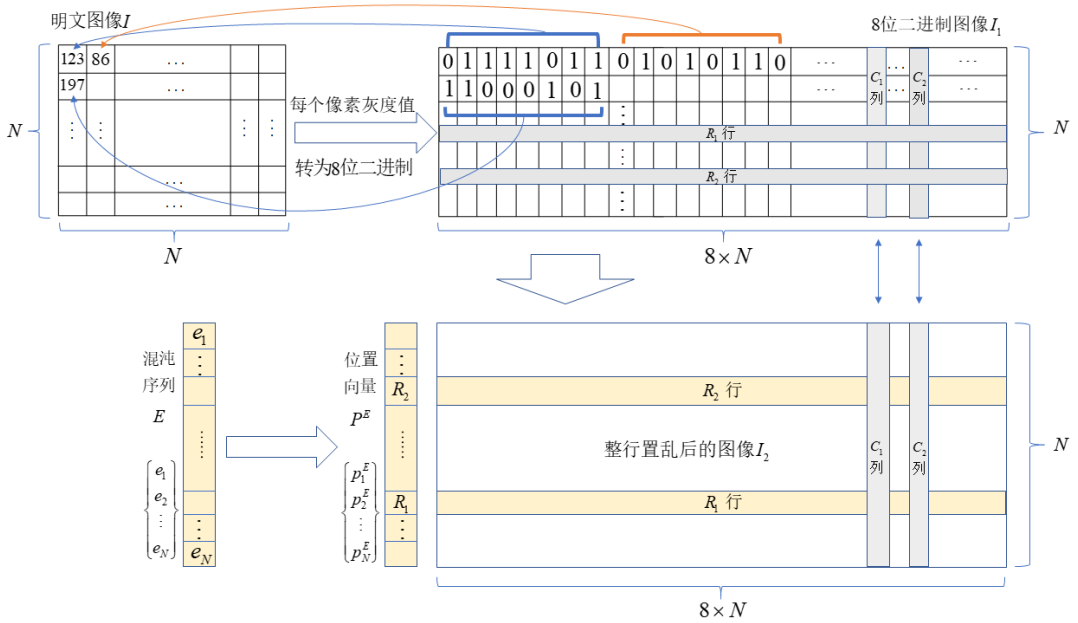


Fig 2.2.1 Tent 系统生成三个混沌序列的示意图.

（6）将像素矩阵中每个像素转换成 8 位二进制数。

（7）将序列 E 按升序排序，得到位置向量 $P^E=\left\{p_1{}^E,p_2{}^E,\cdots,p_N{}^E\right\}$。利用 P^E 对已经转换成比特的数字图像矩阵进行整行置乱。

（8）将序列 F 按升序排序，得到位置向量 $P^F=\left\{p_1{}^F,p_2{}^F,\cdots,p_{8N}{}^F\right\}$。利用 P^F 对整行置乱后的矩阵再进行整列置乱。



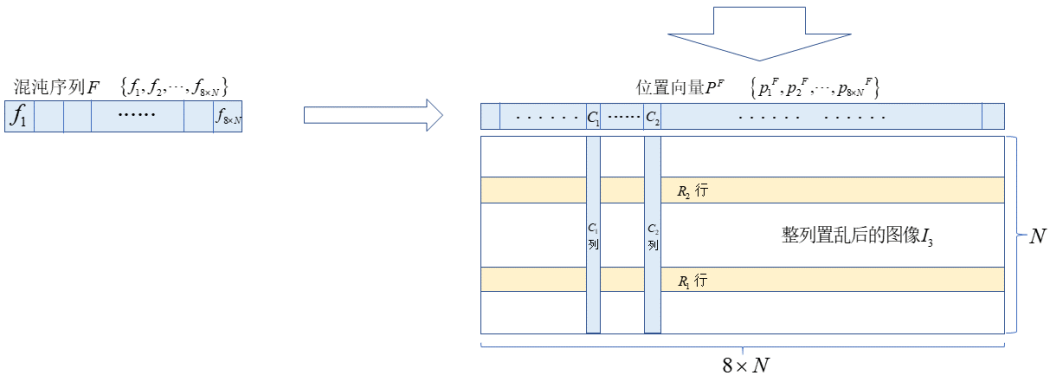


Fig 2.2.2 明文图像矩阵转为比特矩阵，再进行整行，整列置乱的示意图.

（9）将整列置乱后的矩阵从左到右分成 8 个 $N\times N$ 的比特矩阵，对 8 个矩阵分别使用可逆的 Henon 映射

$$\begin{cases} x(n+1)=1-ax^2(n)+y(n) \mod N, \\ y(n+1)=x(n)+c \mod N. \end{cases} \tag{2.2.4}$$

进行置乱，式(2.2.4)中控制参数 a_i，$c_i\,(i=1,2,\cdots,8)$ 的值为

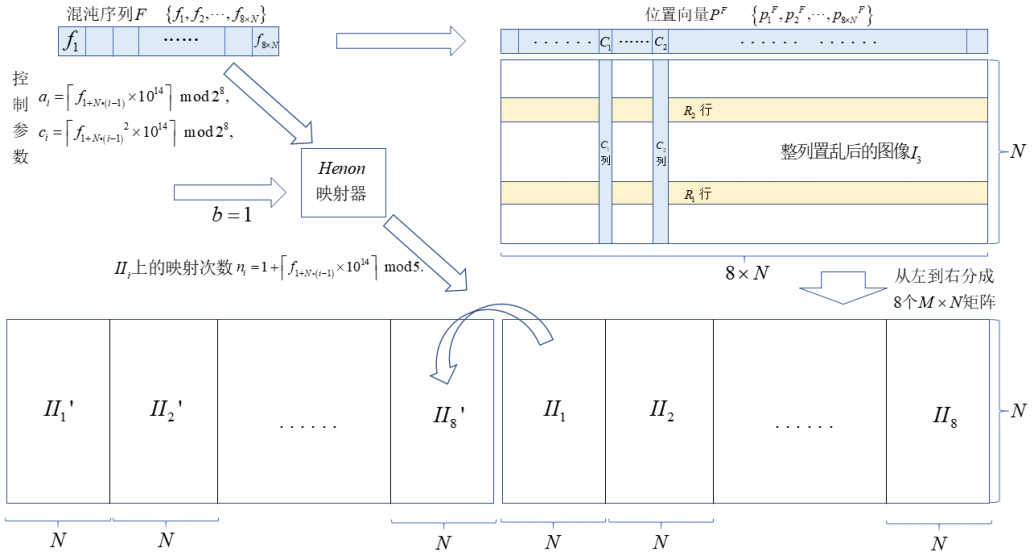
$$a_i=\left\lfloor f_{N\cdot(i-1)+1}\times 10^{14}\right\rfloor \mod 2^8, \tag{2.2.5}$$

$$c_i=\left\lfloor f_{N\cdot(i-1)+1}{}^2\times 10^{14}\right\rfloor \mod 2^8. \tag{2.2.6}$$

其中 $\lfloor\bullet\rfloor$ 表示取整（上，下取整均可，下同）。每个比特矩阵的置乱次数 $n_i\,(i=1,2,\cdots,8)$ 为

$$n_i=1+\left(\left\lfloor f_{N\cdot(i-1)+1}\times 10^{14}\right\rfloor \mod 5\right). \tag{2.2.7}$$

（10）每个比特矩阵完成相应次数的 Henon 映射迭代后，将这 8 个比特矩阵按顺序，按照垂直于比特矩阵平面的方向合并，合并过程中同时将比特转化为十进制像素值，得到中间密文图像 C_0。



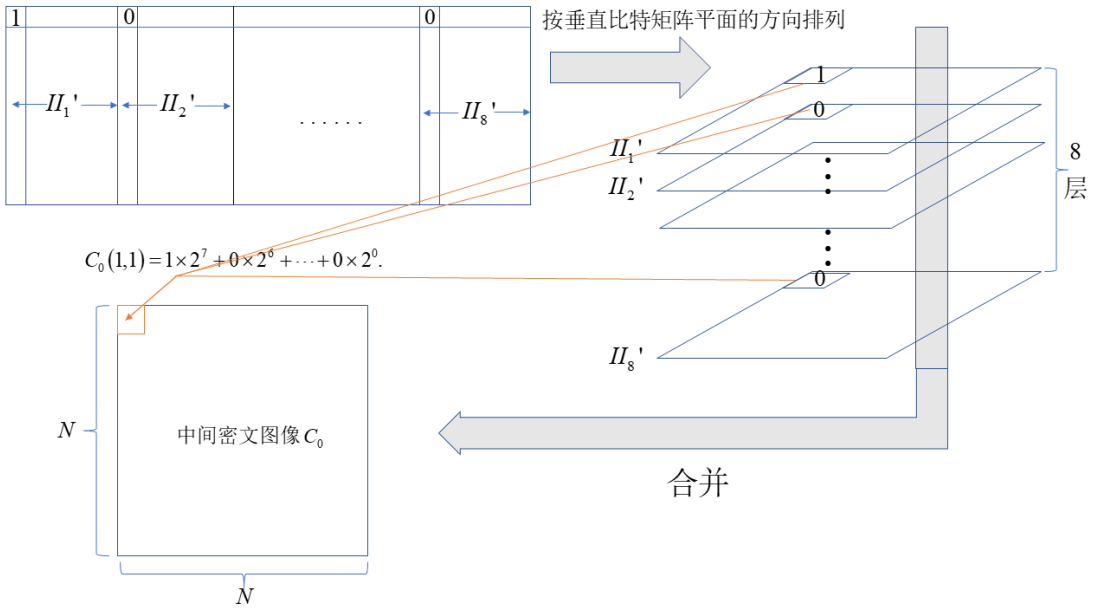


Fig 2.2.3 执行 Henon 映射置乱示意图（上）以及中间密文图像生成过程的示意图（下）.

（11）按照式(2.2.8)对中间密文图像C_0进行最后的扩散加密得到最终密文图像C，

$$C_t = D_t + (C_0)_t + T_t \mod 2^8,\ t=1,2,\cdots,N\times N, \tag{2.2.8}$$

其中$D_t = \left\lceil r_t \times 2^{48} \right\rceil \mod 2^8$，$T_t = \begin{cases} S, & t=1, \\ C_{t-1}, & t>1. \end{cases}$。而$C_t$，$(C_0)_t$与$C(i,j)$，$C_0(i,j)$的对应关系为$C(i,j)=C_t$，$C_0(i,j)=(C_0)_t$，其中$t=(N-1)\times i+j,\ i,j=1,2,\cdots,N$。

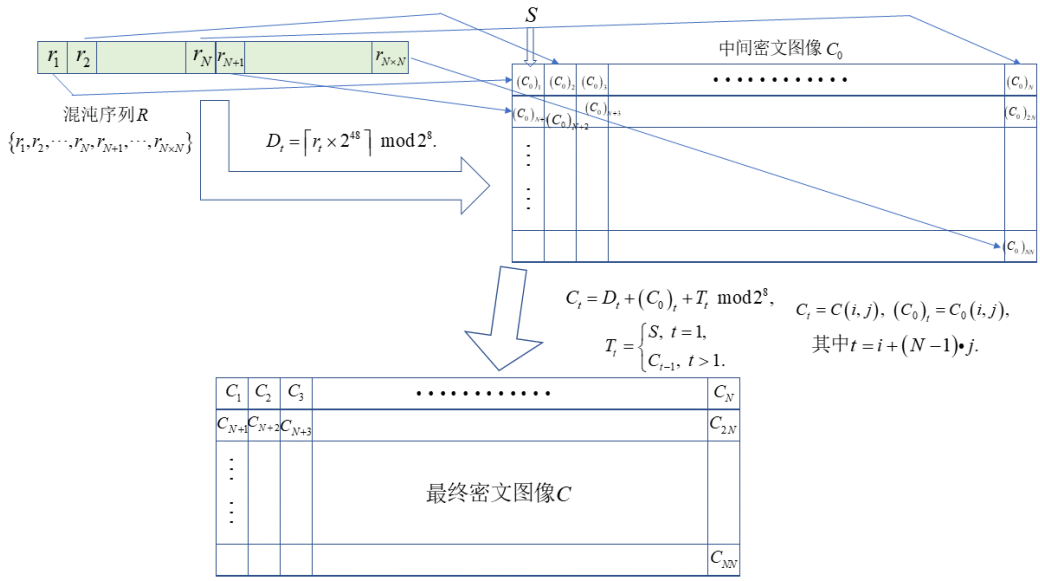


Fig 2.2.4 最终密文图像的产生过程示意图.

解密算法步骤（需要三个解钥 x_0, S, s ）

（1）接收加密算法得到的明文图像像素值总和 s，由式(2.2.2), (2.2.3)计算出 Tent 混沌系统的控制参数 μ 和 Tent 混沌系统初始迭代的次数 k。

（2）根据步骤（1）中求出的控制参数 μ，对 Tent 混沌系统进行 k 次迭代，消除初态效应的影响。继而依次迭代 N 次，$8\times N$ 次，$N\times N$ 次，产生长度分别为 N，$8\times N$，$N\times N$ 的混沌序列 $E=\{e_1,e_2,\cdots,e_N\}$，$F=\{f_1,f_2,\cdots,f_{8\times N}\}$，$R=\{r_1,r_2,\cdots,r_{N\times N}\}$。

（3）接收加密算法得到的最终密文图像 C，按照式(2.2.9)对 C 进行扩散反加密得到原中间密文图像 C_0。

$$\left(C_0\right)_t=C_t-D_t-T_t \mod 2^8,\ t=1,2,\cdots,N\times N. \tag{2.2.9}$$

其中 $\forall t,\ 0\le\left(C_0\right)_t<256$，$D_t=\left\lfloor r_t\times 2^{48}\right\rfloor \mod 2^8$，$T_t=\begin{cases}S,\ t=1,\\ C_{t-1},\ t>1.\end{cases}$。$[\bullet]$ 表示取整（此时要与加密算法中的取整函数 $[\bullet]$ 相同），而 C_t，$\left(C_0\right)_t$ 与 $C(i,j)$，$C_0(i,j)$ 的对应关系为 $C(i,j)=C_t$，$C_0(i,j)=\left(C_0\right)_t$，其中 $t=(N-1)\times i+j,\ i,j=1,2,\cdots,N$。

（4）将中间密文图像 C_0 分解成由上至下的 8 个 $N\times N$ 比特平面，对每个比特矩阵分别使用 Henon 逆映射

$$\begin{cases}x(n)=y(n+1)-c \mod N,\\ y(n)=x(n+1)-1+ax^2(n) \mod N.\end{cases} \tag{2.2.10}$$

进行反置乱，控制参数 a_i，$c_i\ (i=1,2,\cdots,8)$ 的值按式(2.2.5), (2.2.6)计算。每个比特矩阵的反置乱次数 $n_i\ (i=1,2,\cdots,8)$ 按式(2.2.7)计算。

（5）每个比特矩阵完成相应次数的 Henon 逆映射迭代后，将 8 个比特矩阵按顺序从左到右排列，并成一个 $N\times 8N$ 的矩阵，利用由序列 F 得到的位置向量 $P^F=\left\{{p_1}^F,{p_2}^F,\cdots,{p_{N\times N}}^F\right\}$ 对该矩阵先进行整列反置乱，然后再利用由序列 E 得到的位置向量 $P^E=\left\{{p_1}^E,\cdots,{p_N}^E\right\}$ 对完成整列反置乱的矩阵再进行整行反置乱。

（6）最后对完成整行反置乱的 $N\times 8N$ 矩阵中的每 8 个水平方向相邻元素视为一个二进制数，并转化成十进制像素值，得到一个 $N\times N$ 的矩阵，即得到解密图像 I。

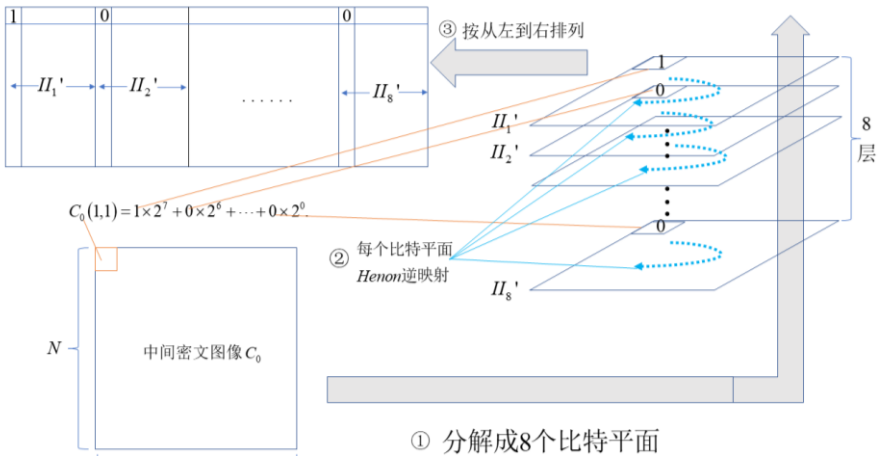


Fig 2.2.5 由中间密文图像反推出 8 个比特矩阵之并的示意图.

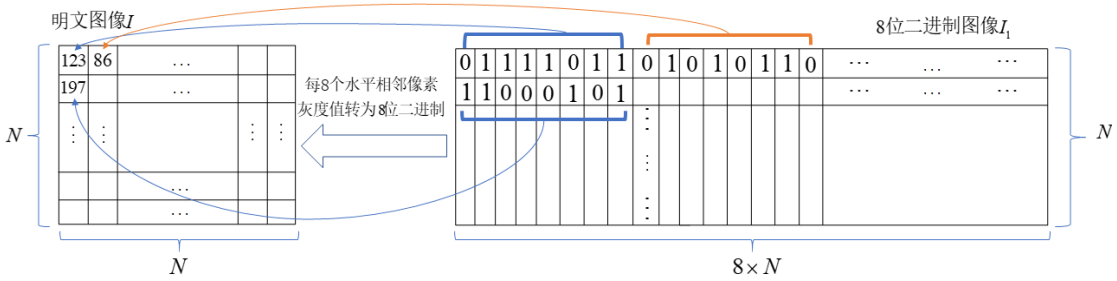


Fig 2.2.6 解密图像生成过程示意图.

§ 2.3 算法改进

§ 2.1 中提到了基于论文[1]所给的资料，只能确定原文的加密算法适用于大小为 $N\times N$ 的图像。这个限制的形成来源于可逆 Henon 映射的表达式

$$\begin{cases} x(n+1)=1-ax^2(n)+y(n) \mod N, \\ y(n+1)=x(n)+c \mod N. \end{cases}$$

中第一，二式的模数都要求是 N ，而原文加密算法涉及可逆 Henon 映射的步骤只有§ 2.2 中加密算法的步骤（9）：应用可逆 Henon 映射对 8 个 $N\times N$ 比特矩阵进行置乱。事实上，当明文图像大小为 $M\times N$ 时，该步骤中的 8 个比特矩阵的大小也都为 $M\times N$ 。我们指出，通过添加一些处理，也可以在 $M\times N$ 矩阵上应用同上的可逆 Henon 映射来置乱。

任取 1 个 $M\times N$ 的比特矩阵 $B_i\,(i=1,2,\cdots,8)$ 为例来说明操作方法。当 $M=N$ 时就是矩阵为方阵的情况，此时不用再添加其他处理。当 $M\neq N$ 时，不妨令 $N>M$ 。我们欲使用滑动窗口法。

滑动窗口法算法步骤

初始创建一个大小为$\delta \times \delta = M \times M$的滑动窗口，设置其滑动步长为$\delta = M$，令窗口的初始位置在矩阵$B_i$的最左端，规定窗口滑动方向为向右。

Step 1

对窗口内的矩阵进行n_i次 Henon 映射置乱，n_i是对应于该比特矩阵B_i的置乱次数。

Step 2

① 若窗口的滑动方向为向右，判断窗口右侧矩阵的列数是否$\geq \delta$。若是，将窗口向右滑动δ格，然后转到 Step 1；否则转到 Step 3.

② 若窗口的滑动方向为向下，判断窗口下侧矩阵的行数是否$\geq \delta$。若是，将窗口向下滑动δ格，然后转到 Step 1；否则转到 Step 3.

Step 3

① 若窗口的滑动方向为向右，当窗口右侧刚好没有矩阵区域，即右侧矩阵列数$=0$时，则该比特矩阵B_i的置乱完全结束，退出算法；当窗口右侧有$a \times b$的矩阵区域（$b > 0$）时，置窗口的大小为$\delta \times \delta = b \times b$，置窗口的滑动步长为$\delta = b$，将窗口移动到剩余矩阵区域的最上方，并规定窗口滑动方向为向下，然后转到 Step 1.

② 若窗口的滑动方向为向下，当窗口下方刚好没有矩阵区域，即下方矩阵行数$=0$时，则该比特矩阵B_i的置乱完全结束，退出算法；当窗口下方有$a \times b$的矩阵区域（$a > 0$）时，置窗口的大小为$\delta \times \delta = a \times a$，置窗口的滑动步长为$\delta = a$，将窗口移动到剩余矩阵区域的最左端，并规定窗口滑动方向为向右，然后转到 Step 1.

对任意有限大小的矩阵，由辗转相除法的思想，上述算法执行的步骤总是有限的。

Note NC : Not Completed, C-ing : Completing, C : Completed.

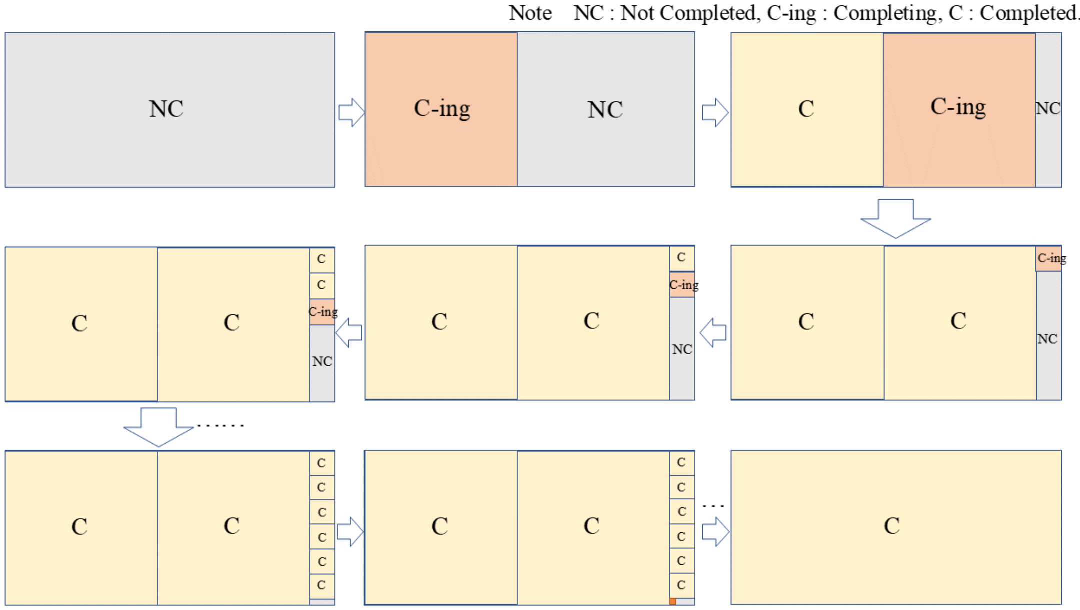


Fig 2.3.1 滑动窗口法执行过程的示意图.

§ 2.4 算法展望

以下针对§ 2.2 中加密算法的一些步骤给出几点拓展。默认已经将滑动窗口法加入到步骤（9）中，使得§ 2.2 中的加密算法已经适用于任意有限大小的图像。

（1）步骤（6）中将像素矩阵中每个像素转换成 8 位二进制数。

① 这 8 个数字默认是按横向排列的，因此得到的矩阵是 $M\times(8\times N)$ 的大小。作为拓展，也可以将这 8 个数字按照纵向排列，得到一个 $(8\times M)\times N$ 的矩阵。

② 原文[1]默认这 8 个数字是紧密排列的，作为拓展，也可以将这 8 个数字两两之间隔开 N 个（横向排列的情况下）或者 M 个（纵向排列的情况下）排列，或者设计其他更加复杂的排列方式。

③ 此外，该步骤中得到大小为 $M\times(8\times N)$ 或者 $(8\times M)\times N$ 的矩阵后，可以考虑再将里面的 8 个 $M\times N$ 的比特矩阵旋转合适的角度（不过要保证旋转后的 8 个小矩阵能够拼接在一起），如大矩阵为 $M\times(8\times N)$ 的大小，从左到右每个大小为 $M\times N$ 的小矩阵可以逆时针旋转 90 度或者顺时针旋转 90 度，每个矩阵旋转的方向可以不一样。又或者当 $M=N$ 时，8 个 $M\times N=N\times N$ 的小矩阵各自可以自由地选择逆时针旋转 90 度，180 度或者 270 度。

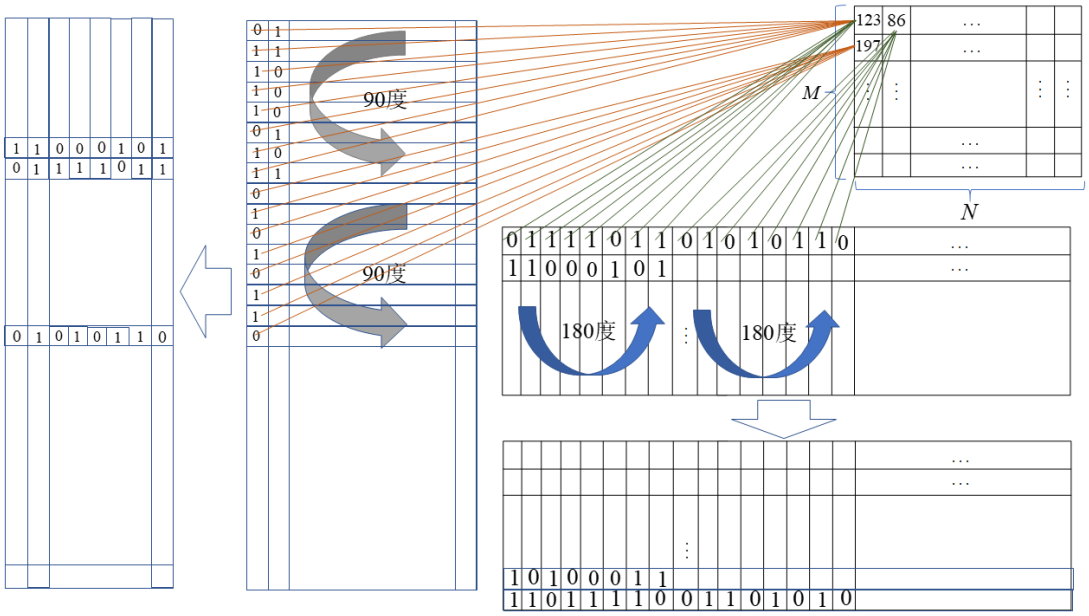


Fig 2.4.1 将像素矩阵各像素转成 8 位二进制数的拓展方法.

（2）步骤（7）中默认将序列 E 按升序排序得到位置向量 P^E，作为拓展，也可以将 E 按降序排序得到位置向量 P^E，或者将 E 中的奇数索引的数据单独按升序（降序）排列得到 P_1^E，偶数索引的数据单独按升序（降序）排列得到 P_2^E，再令 $P^E=\left[P_1^E,P_2^E\right]$ 或 $P^E=\left[P_2^E,P_1^E\right]$

得到位置向量 P^E。或者可以设计更加复杂的排序方式来得到不同的位置向量 P^E。

（3）步骤（9）中对第 i 个比特矩阵默认按下式计算控制参数 a_i, c_i 和迭代次数 n_i

$$a_i = \left\lfloor f_{N\bullet(i-1)+1} \times 10^{14} \right\rfloor \bmod 2^8,$$

$$c_i = \left\lfloor f_{N\bullet(i-1)+1}{}^2 \times 10^{14} \right\rfloor \bmod 2^8,$$

$$n_i = 1 + \left(\left\lfloor f_{N\bullet(i-1)+1} \times 10^{14} \right\rfloor \bmod 5 \right).$$

三个式子均使用了混沌序列 F 里面的第 $N\bullet(i-1)+1$ 个值 $f_{N\bullet(i-1)+1}$。作为拓展，可以让上面三个式子分别使用混沌序列 F 中的其他的值，比如将上面三个式子改写成

$$a_i = \left\lfloor f_{N\bullet(i-1)+1} \times 10^{14} \right\rfloor \bmod 2^8,$$

$$c_i = \left\lfloor f^2_{N\bullet(i-1)+\left\lfloor \frac{N}{2} \right\rfloor} \times 10^{14} \right\rfloor \bmod 2^8,$$

$$n_i = 1 + \left(\left\lfloor f_{N\bullet i-1} \times 10^{14} \right\rfloor \bmod 5 \right).$$

此外，若希望对第 i 个比特平面迭代 Henon 映射的最大次数不是 5 次，例如改成 10 次，20 次，则可以将 n_i 计算式中的 $\bmod 5$ 改成 $\bmod 10,\ \bmod 20$。

（4）步骤（1）中计算 Tent 混沌系统初始迭代次数 k 的公式是

$$k = 10^3 + \left(s \bmod 10^3 \right).$$

若希望初始迭代的最小次数不是 10^3，例如改成 500，10^4 次，则可以将上式改写成

$$k = 500 + \left(s \bmod 500 \right),\ k = 10^4 + \left(s \bmod 10^4 \right).$$

初始迭代的最小次数其实也是可以作为另外的密钥输入到加密程序中。

（5）步骤（10）中默认将各自完成相应次数 Henon 映射迭代的 8 个比特矩阵按垂直于比特矩阵平面的方向合并，作为拓展，也可以在 8 个比特矩阵垂直排列的同时，将其中部分或全部矩阵旋转合适的角度后再合并，比如由上至下将第 1, 3, 5, 7 个矩阵逆时针旋转 90 度，将第 2, 4, 6, 8 个矩阵顺时针旋转 90 度，然后再合并. 或者当大比特矩阵的大小是 $M \times (8 \times N)$ 时，直接将每 8 个横向相邻像素的灰度值合并，转化为十进制像素值(当大比特矩阵的大小是 $(8 \times M) \times N$ 时，则将每 8 个纵向相邻像素的灰度值合并，转化为十进制像素值). 或者设计其他更加复杂的合并方式.

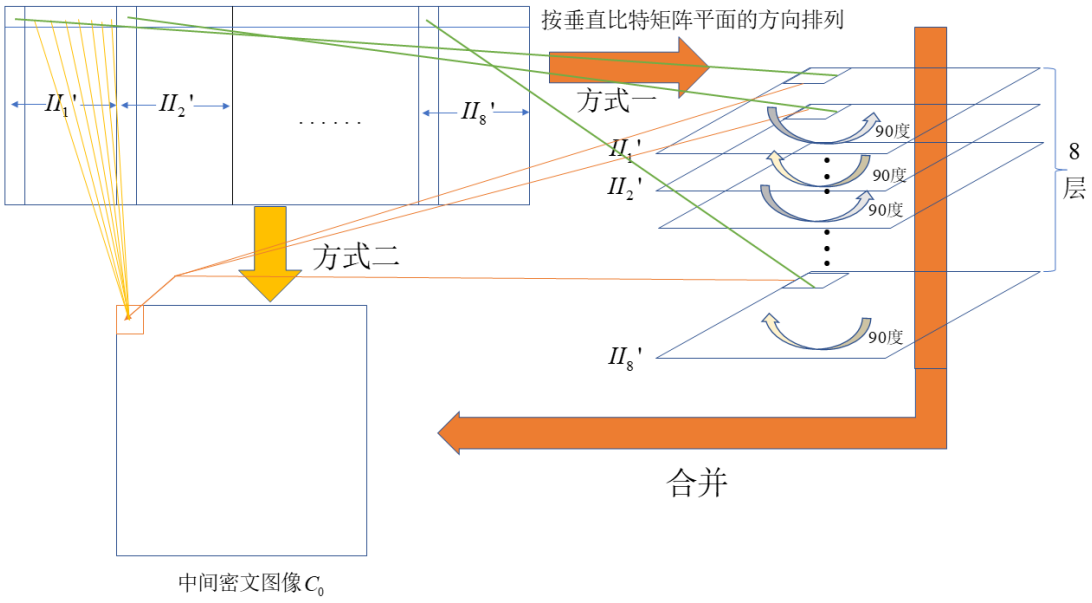


Fig 2.4.2 八个比特矩阵合并得到中间密文图像的其他思路.

（6）§2.3 算法改进关于滑动窗口的属性有滑动窗口大小，滑动窗口移动步长，滑动窗口每一阶段的初始位置 (这里一个阶段的末尾的标志是窗口再按照规定方向滑动会超出区域范围)，滑动窗口的滑动方向. 以刚开始的滑动窗口为例:

① 本文上面设置刚开始的滑动窗口大小为 $\min\{M,N\}\times\min\{M,N\}$，实际上在严格限制滑动窗口为方形窗口的前提下，可以调整滑动窗口的大小为 $\delta\times\delta$，其中 $\delta\in\mathbb{Z}_{>0}$, $2\le\delta\le\min\{M,N\}$. 需要注意的是窗口大小变小后，原来的移动策略也需要作出相应更加复杂的调整. 而滑动窗口越小，一般相应地需要的计算量也会变多.

② 对于滑动窗口移动步长，作为拓展，其实不一定要等于滑动窗口的尺寸. 例如滑动窗口的大小为 $\delta\times\delta$，则可以设置移动步长为 $\left[\frac{\delta}{2}\right]$，此时移动前窗口 I_1 和移动后的窗口 I_2 的交非空，但这并不会给算法带来破坏，只不过交集区域的像素会进行多起码一倍计算量的 Henon 映射置乱而已，某种意义上讲，可以使得算法结构更具复杂性，更稳定.

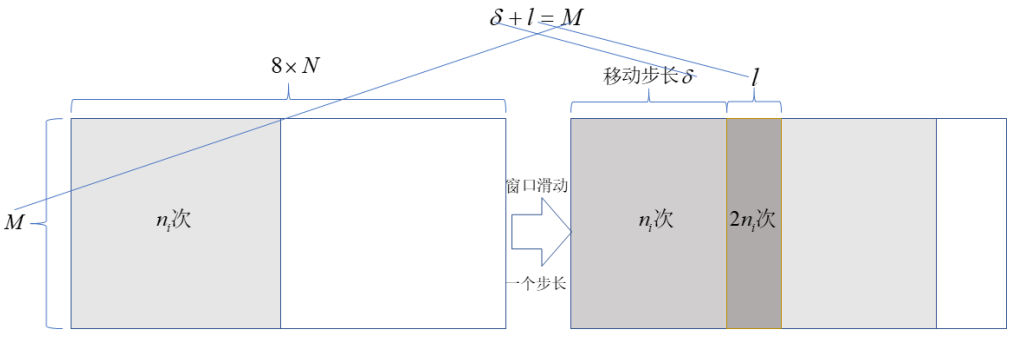


Fig 2.4.3 第 i 个比特矩阵上应用滑动窗口—Henon 映射置乱的示例一.

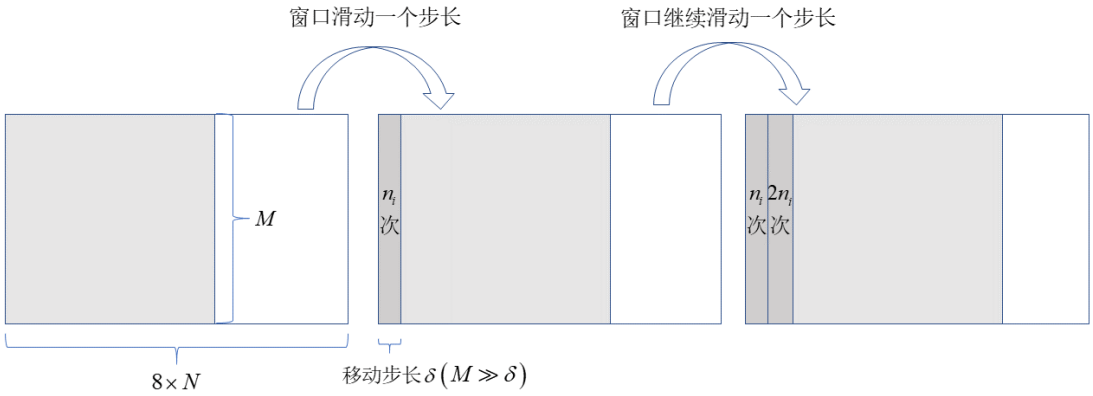


Fig 2.4.4 第 i 个比特矩阵上应用滑动窗口—Henon 映射置乱的示例二.

③ 对于滑动窗口每一阶段的初始位置，可以是最左端，最右端，最上方，最下方，或者是其他非平凡的位置，只不过初始位置一改，相应的窗口移动策略也需要作出相应调整.

④ 对于滑动窗口滑动方向，一般就是向左滑动，向右滑动，向上滑动，向下滑动. 或者可以设计其他更加复杂的滑动方式，只不过需要设计相应的滑动策略，以及验证这样的滑动方式和滑动策略能不能让起码绝大部分的像素发生了 Henon 映射置乱，确保这样的操作是有效的.

§ 2.5 算法分析

（1）完全离散的 Henon 映射[8]

$$\begin{cases} x(n+1)=1-ax^2(n)+y(n) \mod N, \\ y(n+1)=bx(n)+c \mod N. \end{cases} \tag{2.5.1}$$

当 $b=1$ 时是集合 $A=\left\{(n_1,n_2)\in\mathbb{Z}_{>0}^2 : 1\le n_{1,2}\le N\right\}$ 上的双射。

证明 当 $b=1$ 时，表达式更新为

$$\begin{cases} x(n+1)=1-ax^2(n)+y(n) \mod N, \\ y(n+1)=x(n)+c \mod N. \end{cases} \tag{2.5.2}$$

首先证明映射(2.5.2)是一个单射，即证明成立推理

$$\begin{aligned} &(x_1(n+1),y_1(n+1))=(x_2(n+1),y_2(n+1)). \\ \Rightarrow\ &(x_1(n),y_1(n))=(x_2(n),y_2(n)). \end{aligned} \tag{2.5.3}$$

$$\begin{aligned} &(x_1(n+1),y_1(n+1))=(x_2(n+1),y_2(n+1)). \\ \Rightarrow\ &y_1(n+1)=y_2(n+1). \\ \Rightarrow\ &x_1(n)+c\equiv x_2(n)+c \mod N. \\ \Rightarrow\ &x_1(n)\equiv x_2(n) \mod N. \end{aligned}$$

由于$1 \le x_{1,2}(n) \le N$，因此成立

$$x_1(n) = x_2(n). \tag{2.5.4}$$

基于式(2.5.4)，有以下推理

$$\begin{aligned}
&\left(x_1(n+1), y_1(n+1)\right) = \left(x_2(n+1), y_2(n+1)\right).\\
\Rightarrow\ & x_1(n+1) = x_2(n+1).\\
\Rightarrow\ & 1 - a x_1^{\,2}(n) + y_1(n) \equiv 1 - a x_2^{\,2}(n) + y_2(n) \mod N.\\
\Rightarrow\ & y_1(n) \equiv y_2(n) \mod N.
\end{aligned}$$

由于$1 \le y_{1,2}(n) \le N$，因此成立

$$y_1(n) = y_2(n). \tag{2.5.5}$$

由式(2.5.4), (2.5.5)，得到推理(2.5.3)成立，即映射(2.5.2)是一个单射。

再证明映射(2.5.2)是一个满射。根据同余式的性质，成立

$$\begin{aligned}
&x(n+1) \equiv 1 - a x^2(n) + y(n) \mod N. && y(n+1) \equiv x(n) + c \mod N.\\
\Leftrightarrow\ & y(n) \equiv x(n+1) - 1 + a x^2(n) \mod N. && \Leftrightarrow x(n) \equiv y(n+1) - c \mod N.
\end{aligned}$$

得到逆推式

$$\begin{cases}
x(n) = y(n+1) - c \mod N,\\
y(n) = x(n+1) - 1 + a x^2(n) \mod N.
\end{cases} \tag{2.5.6}$$

式(2.5.6)说明了$\forall\left(x(n+1), y(n+1)\right) \in A$，总可以由式(2.5.6)找到对应的$\left(x(n), y(n)\right)$，使得二者成立(2.5.2)的映射关系，因此映射(2.5.2)是一个满射。

综上，映射(2.5.2)是A上的一个双射。 □

上面的证明过程也给出了式(2.5.6)就是映射(2.5.2)的逆映射。又因为**双射的复合还是一个双射**，这为算法中在$N \times N$方阵上能够实行若干次完全离散 Henon 映射的可行性提供了理论依据。

（2）Tent 系统的系统方程为

$$\begin{cases}
x(n+1) = \mu \bullet x(n),\ 0 < x(n) \le 0.5,\\
x(n+1) = \mu \bullet \left(1 - x(n)\right),\ 0.5 < x(n) < 1.
\end{cases}$$

上式要求$x(n) \in (0,1)$，为了让$x(n+1)$也满足$x(n+1) \in (0,1)$，规定$\mu \in (0,2)$。

① $0 < \mu < 1$时，序列$\{x(n)\} = \{x(1), x(2), \mu x(2), \mu^2 x(2), \cdots\}$，其中

$$x(2) = \min\{x(1), 1 - x(1)\} < 0.5.$$

该序列除去第 1 个数之后显然是一个单调递减的收敛序列，收敛点为 0.

② $\mu=1$ 时，序列 $\{x(n)\}=\{x(1),x(2),x(2),x(2),\cdots\}$，其中

$$x(2)=\min\{x(1),1-x(1)\}<0.5.$$

③ $1<\mu<2$ 时，系统处于混沌状态。

需要特别注意的是，论文[1]基于上面的讨论只点明了算法应用 Tent 系统时应该控制参数 $\mu>1$ 使得该系统时混沌的，但是我们同时需要注意初值 x_0（加密算法的密钥）的选取。这里就只简单地讨论一点，考虑

$$\begin{cases} x(n)>0.5, \\ x(n)=\mu(1-x(n)). \end{cases} \tag{2.5.7}$$

当已经确定了 Tent 混沌系统参数 μ 的值之后，应该避免初值 $x(1)$ 即密钥 x_0 的取值满足上式(2.5.7)，否则由 Tent 系统得到的序列除去第一位后是一个常数列 $\{x(2)\}$。

例子 已经确定 $\mu=1.5$，令 $x_0=0.6$，则 μ 与 x_0 满足式(2.5.7)，此时 Tent 系统产生的序列为

$$0.6,\ 0.4,\ 0.4,\ 0.4,\ \cdots$$

已经不具备混沌性，该序列作为密钥流，安全性将大大折扣。类似地，当已经确定了初值 $x(1)$ 即密钥 x_0 而要选择参数 $\mu\in(1,2)$ 时，也要注意类似的问题。

（3）注意到 Tent 混沌序列的控制参数 μ 和初始迭代次数 k 的计算公式

$$\mu=2^{\frac{s}{M\times N\times 255}},$$

$$k=10^3+(s\ \bmod 10^3),$$

中都使用到明文图像的像素值总和 s，因此 Tent 混沌序列的生成与明文图像有关，当密钥 x_0 相同而明文图像不同时，Tent 系统生成的混沌序列也不一样，由此增强了算法的明文敏感性。

（4）注意到加密算法全过程中使用到的混沌序列不是唯一的，有长度为 M 的混沌序列 E，长度为 $8\times N$ 的混沌序列 F，以及长度为 $M\times N$ 的混沌序列 R.

由 E 产生的位置向量 P^E 用来对 $M\times(8\times N)$ 的大比特矩阵进行整行置乱.

由 F 产生的位置向量 P^F 用来对 $M\times(8\times N)$ 的大比特矩阵进行整列置乱，且 F 本身

有 8 个分量分别参与到 8 个 $M \times N$ 小比特矩阵上 Henon 映射参数 a_i, c_i 和映射迭代次数 n_i 的生成.

混沌序列 R 在最后(第 3 阶段扩散解密)灰度加密过程中起到了作用.

由此可见, 算法过程中综合使用了 3 种混沌序列来进行置乱或扩散, 算法安全性增强了.

（5）注意到该算法突破了**置乱范围在同一比特面的限制**, 具体表现在原算法中的对 $M \times (8 \times N)$ 的大比特矩阵的**整列置乱**环节, 使得一个比特面里的元素 0 或 1 有机会移动到另一个比特面同一行的某个位置, 再配合上之前的**整行置乱**环节, 实现了一个比特面里的元素 0 或 1 有机会移动到另一个比特面里的任何一个位置. 置乱的规则由混沌序列 E 和 F 所产生的位置向量 P^E 和 P^F 给出.

（6）§2.2 中加密算法涉及到**置乱**的主要环节有: 步骤(7)中的整行置乱, 步骤(8)中的整列置乱, 步骤(9)中每个比特矩阵上各自进行的若干次 Henon 映射置乱. 值得指出的是, 上面的置乱均在比特平面上进行, 按照原文[1]的说法, 实质上既改变了像素的位置, 又改变了像素的值, 即也带有一些扩散的效果.

算法涉及到**扩散**的主要环节是: 步骤(11)中的扩散加密, 该步骤的增加增强了抵御统计分析攻击的能力.

（7）§2.2 步骤(9)中计算 a_i, c_i, n_i 的式子中均出现 10^{14}，它的意义分析如下。Tent 混沌系统输入的初值 x_0 是 0 到 1 之间的一个小数, 控制参数 μ 是介于 1 到 2 的一个小数. 两个小数相乘得到的结果的小数点后位数一般会更多. 由此可以看出, 在保证精度的前提下, 随着 Tent 系统不断产生新的值, 该新值的小数点后位数一般会不断变多. 以 a_i 的计算公式为例, $f_{N\bullet(i-1)+1} \times 10^{14}$ 就是将混沌序列 F 的分量 $f_{N\bullet(i-1)+1}$ 的小数点后 14 位移动到整数位上, 配合取整函数 $[\bullet]$, 将第 15 位之后的小数删去不作使用.

§3 加解密算法的仿真实验

图像大小是 $N \times N$ 的情况

论文[1]中仿真使用的图像是 256×256 的。由于找不到和原论文[1]配图相同的高清图，这里使用另一幅构图相似的图片素材进行仿真实验。该图片素材的大小为 323×323。

加密过程使用密钥 $x_0 = 0.234,\ S = 1280$。

明文图像的像素值总和 $s = 13122840$。

解密过程使用解钥 $x_0 = 0.234,\ S = 1280,\ s = 13122840$。

加密结果

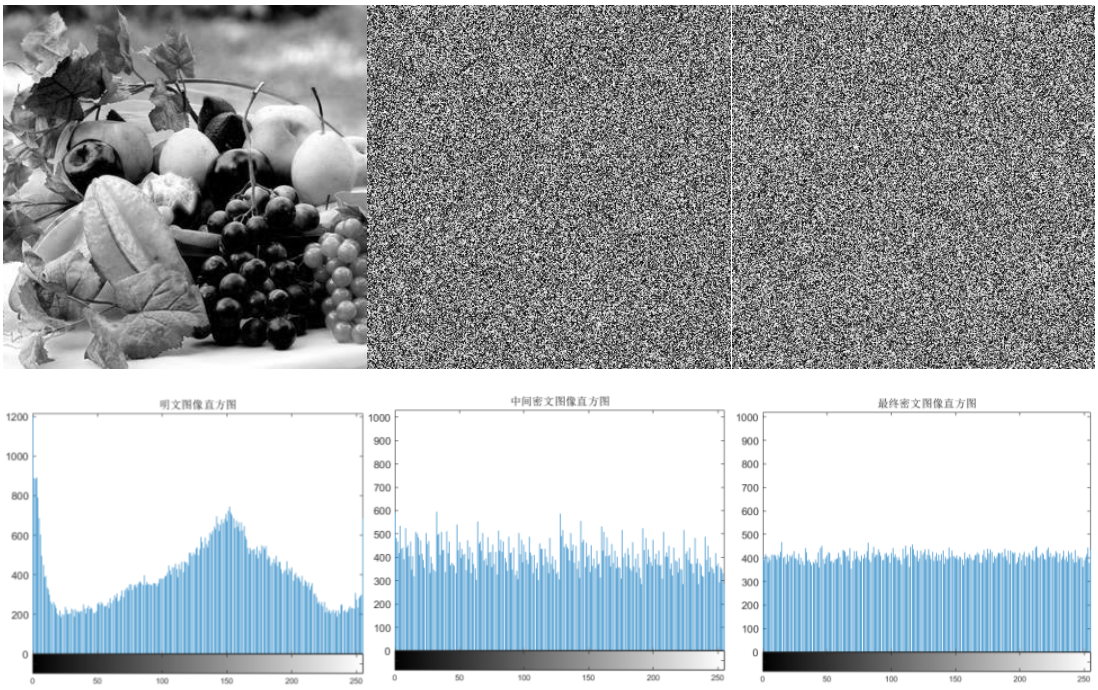


Fig 3.1 N*N 明文图像（左），中间密文图像（中），最终密文图像（右）和各自的直方图.

直观上看第一行的图像变化，表明该加密算法确实具有较好的加密性质。由第二行的直方图可知，加密后明文图像的像素值的分布由不均匀变成了近似均匀分布，明文图像各个灰度级之间的相关性被打破，使得原图没有了统计特性。

解密结果

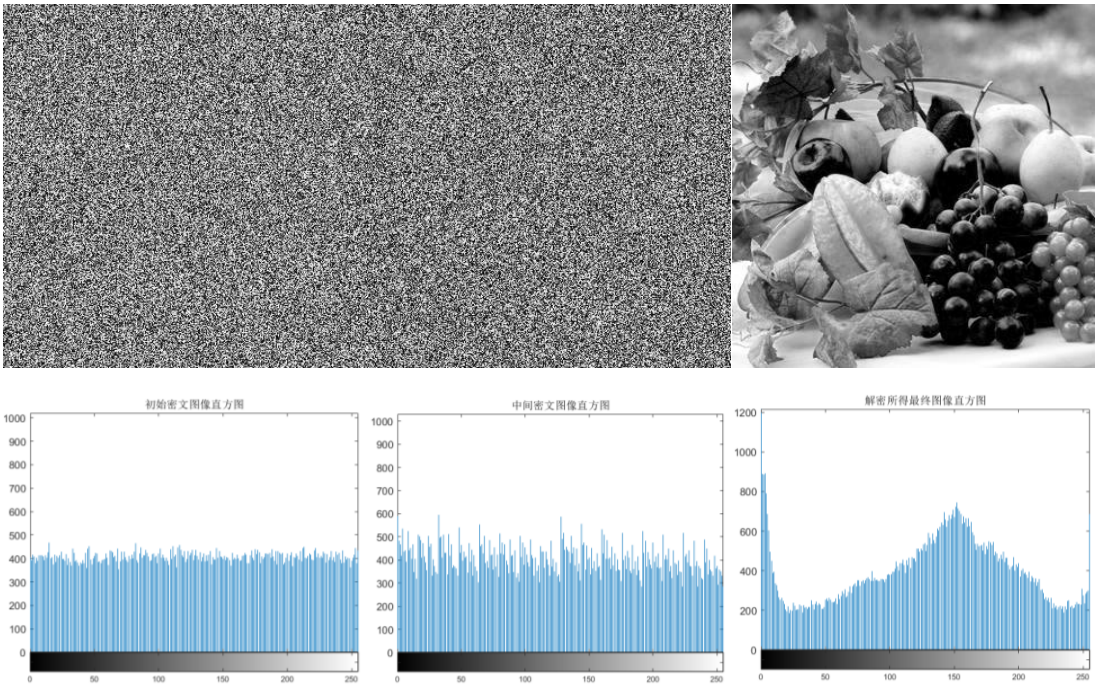


Fig 3.2 N*N 初始密文图像（左），中间密文图像（中），最终解密图像（右）和各自的直方图.

解密后所得图像与原明文图像完全一致，该算法对 $N \times N$ 大小图像的解密是无损的。

图像大小是$M \times N\left(M < N\right)$的情况（使用图片素材是$323 \times 500$的 fruit.png）

加密过程使用密钥$x_0 = 0.234,\ S = 1280$。

明文图像的像素值总和$s = 22112209$。

解密过程使用解钥$x_0 = 0.234,\ S = 1280,\ s = 22112209$。

加密结果

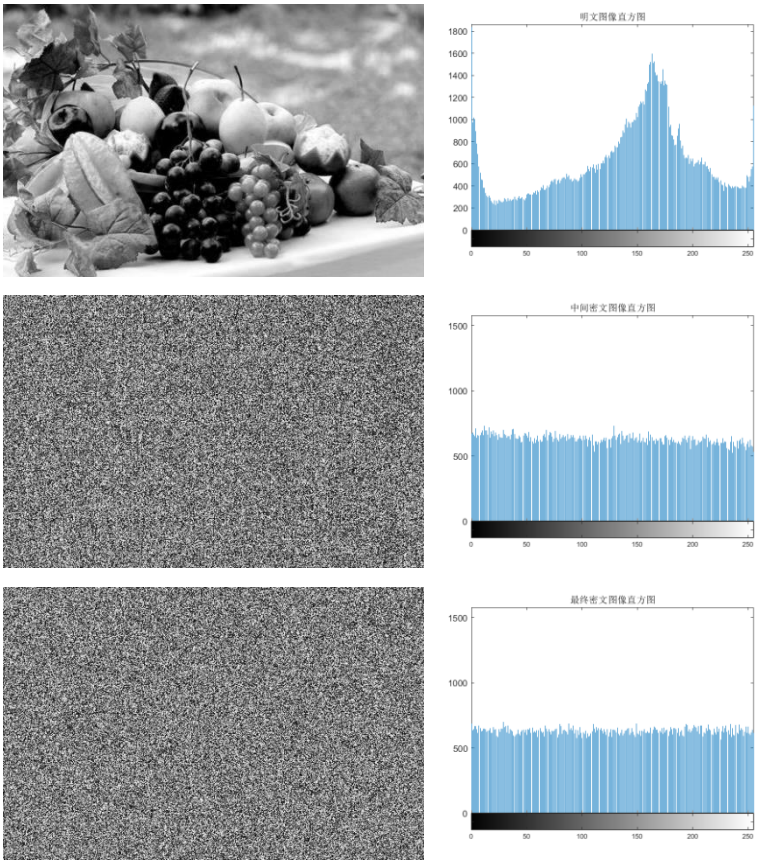
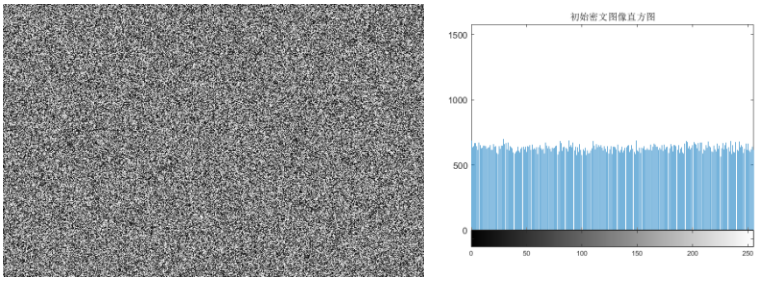


Fig 3.3 M*N (M<N)明文图像（上），中间密文图像（中），最终密文图像（下）和各自的直方图.

解密结果



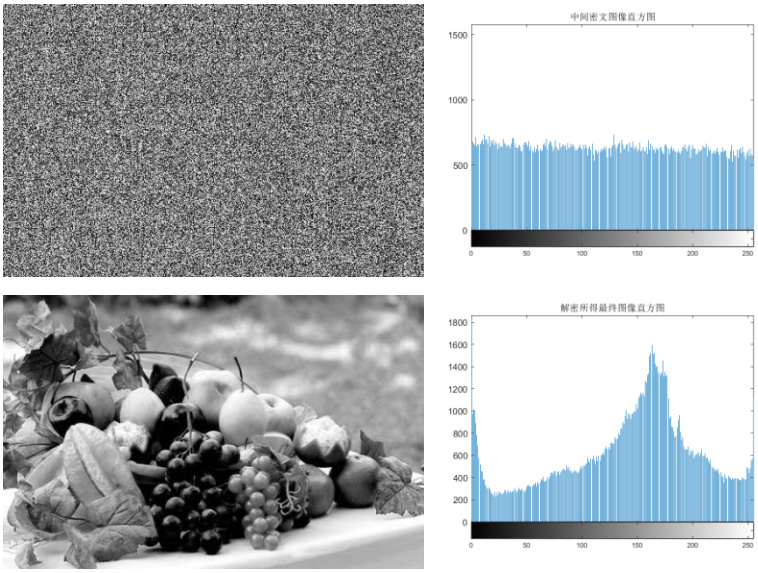


Fig 3.4 M*N (M<N)初始密文图像（上），中间密文图像（中），最终解密图像（下）和各自的直方图.

图像大小是 $M \times N\left(M > N\right)$ 的情况（使用图片素材是 651×500 的 **cat.png**）

加密过程使用密钥 $x_0 = 0.234,\ S = 1280$ 。

明文图像的像素值总和 $s = 45349338$ 。

解密过程使用解钥 $x_0 = 0.234,\ S = 1280,\ s = 45349338$ 。

加密结果

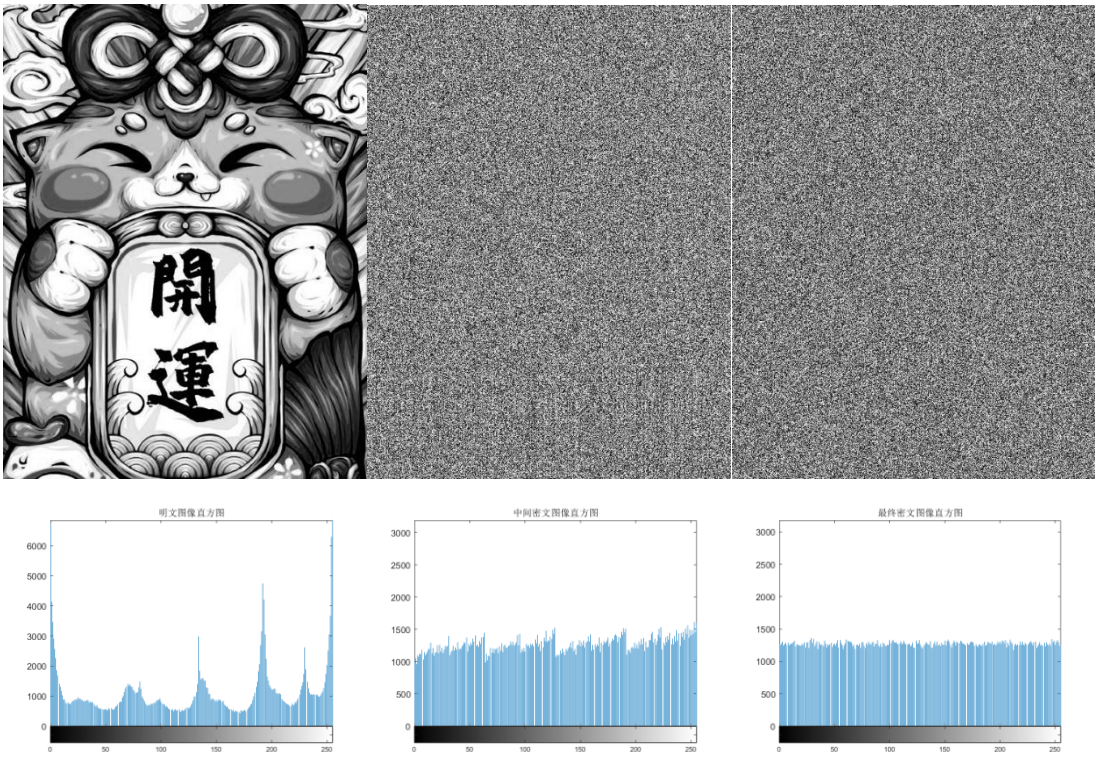


Fig 3.5 M*N (M>N)明文图像（左），中间密文图像（中），最终密文图像（右）和各自的直方图.

解密结果

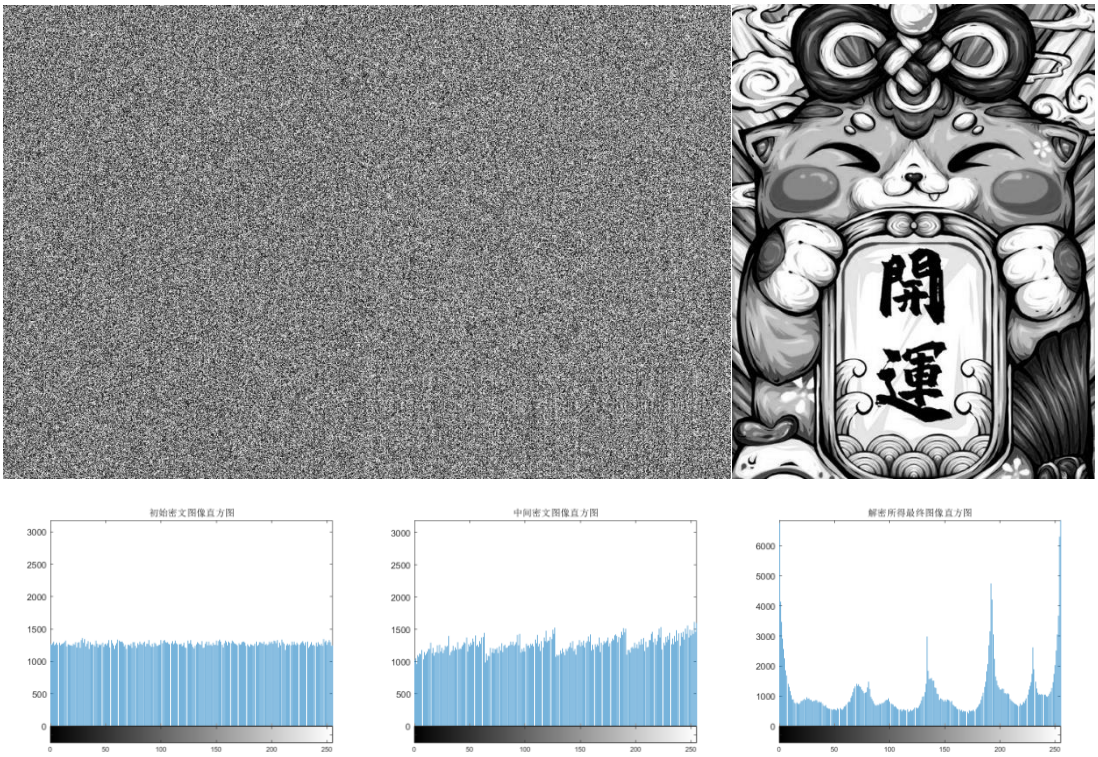
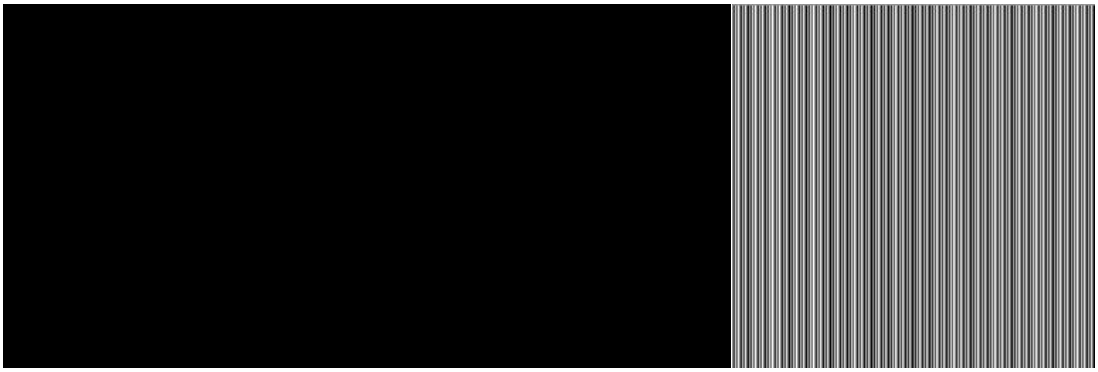


Fig 3.6 M*N (M>N)初始密文图像（左），中间密文图像（中），最终解密图像（右）和各自的直方图.

图像是全黑和全白的特殊情况

Tent 混沌系统 $x(n+1)=\mu \bullet \min\{x(n),1-x(n)\}$, $x(n)\in(0,1)$ 的控制参数 μ 要求满足 $\mu\in(1,2)$，而原文算法给出的 μ 的计算公式为 $\mu=2^{\frac{s}{M\times N\times 255}}$，其中 s 是明文图像像素值总和，$M\times N$ 是图像的大小。当明文图像是全黑时，$s=0$，由上式计算得 $\mu=2^0=1$；当明文图像是全白时，有 $s=M\times N\times 255$，此时计算得 $\mu=2^1=2$；而当明文图像是介于这两种情况之间的一般图像时，有 $\mu\in(1,2)$。因此我们置明文图像为全黑，全白这两种特殊情况，检验原文算法能否照常适用，附录提供的 m 文件 **cmbr_encryption.m** 能否照常运行。

加密结果



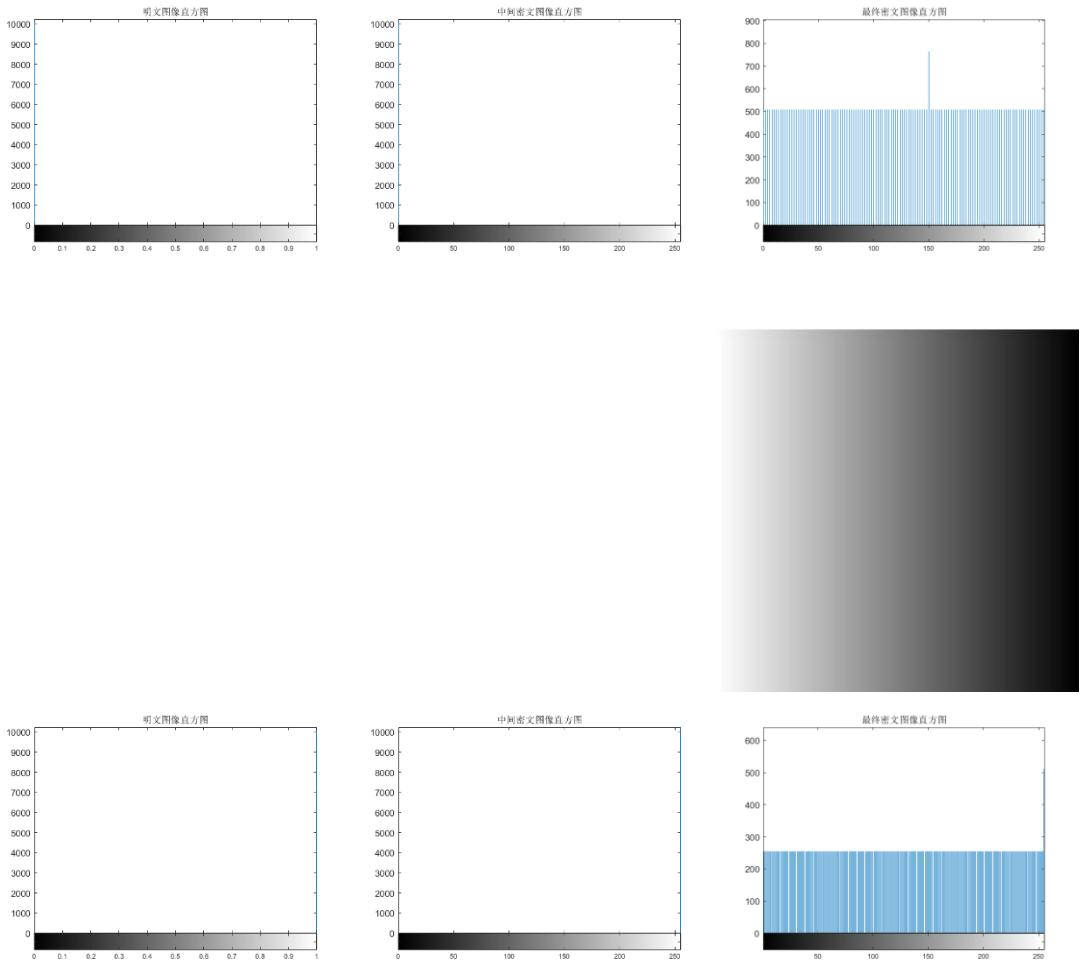


Fig 3.7 256*256 全黑图像加密过程（第一行）及其直方图（第二行），
全白图像加密过程（第三行）及其对应的直方图（第四行）.

结论是原加密算法照常适用。

§ 4 加解密算法的性能分析

§ 4.1 明文敏感性分析

评估指标

（1）像素变化率 R：表示当明文图像中某一个像素发生改变的时候，经过加密算法加密后得到密文图像的所有像素的改变率，计算公式为

$$\begin{cases} D(i,j)=\begin{cases}1, & C_1(i,j)\neq C_2(i,j),\\ 0, & C_1(i,j)=C_2(i,j),\end{cases} \\ R=\dfrac{\sum_{i,j} D(i,j)}{W\times H}\times 100\%. \end{cases}$$

其中W,H分别表示图像的宽和高，C_1,C_2分别表示加密后的密文图像以及明文图像的一个像素发生变化后再经过加密后的密文图像，$C_1(i,j),C_2(i,j)$表示密文图像中第(i,j)点的像素值。R越接近理想值$1-2^{-8}\approx 0.9961$[9]，说明密文对明文的敏感性越好。

（2）归一化平均变化强度U：表示原图与加密图像之间的平均加密强度，计算公式为

$$U=\frac{1}{W\times H}\left[\sum_{i,j}\frac{\left|C_1(i,j)-C_2(i,j)\right|}{255}\right]\times 100\%.$$

其中W,H分别表示图像的宽和高，C_1,C_2分别表示加密后的密文图像以及明文图像的一个像素值发生变化后再经过加密后的密文图像，$C_1(i,j),C_2(i,j)$表示密文图像中第(i,j)点的像素值。U越接近理想值0.3446[10]，说明算法越能够抵抗差分攻击。

分析方法（密，解钥$x_0=0.234,\ S=1280$）

① 原明文图像I_1正常加密得到密文图像C_1；

② 在I_1中随机选取一个像素，将该像素的值随机调整为0到255之间的另外一个不同的值得到另一个图像I_2，然后将I_2正常加密得到密文图像C_2；

③ 根据C_1,C_2计算指标R,U，观察两个指标距离各自理想值的情况；

④ 以上操作重复若干次（例如100次）。

实验结果

使用以下323×323的图片素材作为明文图像。

Fig 4.1.1 明文图像 fruit.png，大小为 323*323.

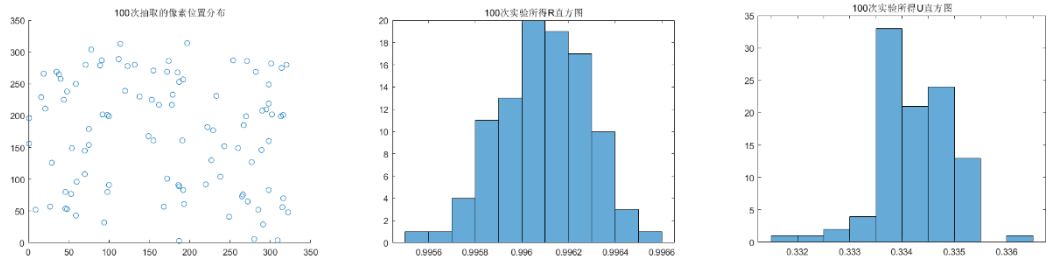


Fig 4.1.2 100 次重复实验得到的像素位置分布图（左），R 分布直方图（中），以及 U 分布直方图（右）.

100 次重复实验中 R 出现的最小值为 0.9956，最大值为 0.9965；U 出现的最小值为 0.3318，最大值为 0.3361。两个指标的表现均接近各自的理想值，由此得出结论：该加密算法得到的密文对明文的敏感性很强，且该算法能够有效抵御差分攻击。

明文敏感性来源

§ 2.2 一节中步骤（1）中 Tent 系统的控制参数 μ 和初始迭代次数 k 的计算公式含有明文图像像素和 s，即 Tent 系统本身就具有很强的明文敏感性。在此基础上，算法中由 Tent 系统产生的 3 个混沌序列 E,F,R 又参与到算法中的很多其他步骤，从而间接赋予算法其他环节一定的明文敏感性。具体地有，混沌序列 E 参与到步骤（7）中的整行置乱，混沌序列 F 参与到步骤（8）的整列置乱，步骤（9）中的比特面置乱，混沌序列 R 参与到步骤（11）的扩散加密。

§ 4.2 密钥，解钥敏感性分析

评估指标

同**明文敏感性分析**中的评估指标：像素变化率 R 和归一化平均变化强度 U，不同的地方在于，二者计算公式中的 C_1, C_2 分别指代明(密)文图像正常加(解)密得到的密(明)文图像，某一个密(解)钥发生微小改变而其他密(解)钥保持不变的情况下将明(密)文图像进行加(解)密得到的密(明)文图像.

加密算法需要的密钥有两个，分别为 Tent 混沌系统迭代初值 x_0 和扩散加密环节的 S；

解密算法需要的解钥有三个，分别为 Tent 混沌系统迭代初值 x_0，明文图像像素值总和 s，扩散加密环节的 S.

其中 $x_0 \in (0,1)$ 是小数，$s \in \mathbb{Z}_{>0}$，$S \in \mathbb{Z}$.

分析方法

明文图像还是采用 Fig 4.1.1 的 fruit.png，其像素值总和 $s=13122840$ 。

加密算法默认密钥 $x_0=0.234,\ S=1280$.

① 令 $x_0=0.234+10^{-10}$，S 保持不变，研究加密算法对密钥 x_0 的敏感性.

② 令 $S=1280+1,\ x_0$ 保持不变，研究加密算法对密钥 S 的敏感性.

解密算法默认解钥 $x_0=0.234,\ S=1280$，$s=13122840$，

①令 $x_0=0.234+10^{-10}$，$s,\ S$ 保持不变，研究解密算法对解钥 x_0 的敏感性.

②令 $S=1280+1,\ x_0,\ s$ 保持不变，研究解密算法对解钥 S 的敏感性.

③令 $s=13122840+1,\ x_0,\ S$ 保持不变，研究解密算法对解钥 s 的敏感性.

实验结果（针对密钥 x_0）

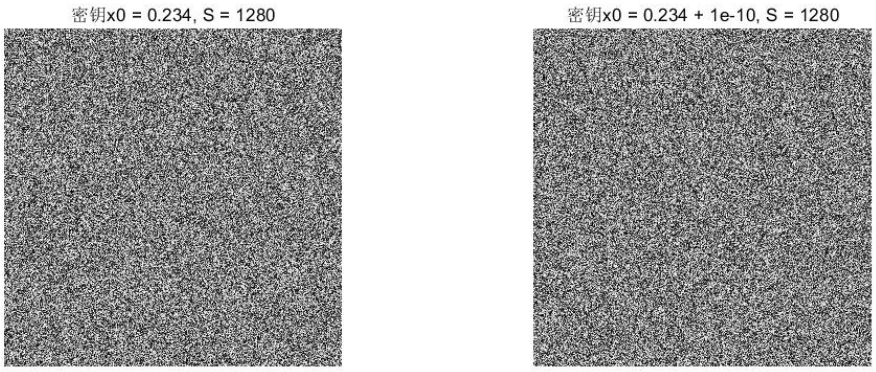


Fig 4.2.1 密钥 x0 发生微小变动后密文图像的变化.

1. 同一幅明文图像 `fruit.png` 在密钥 `S = 1280`，而 `x0` 分别为 `0.234`，`0.234 + 1e-10` 下加密得到的两幅密文图像的像素变化率 `R` 和归一化平均变化强度 `U` 分别为：
2. `R = 0.99602`
3. `U = 0.33424`

结果表明 R 值接近理想值 0.9961，U 值接近理想值 0.3446，说明加密算法对密钥 x_0 的敏感性强，该算法对于针对密钥 x_0 的差分攻击的抵御力较强。

实验结果（针对密钥 S ）

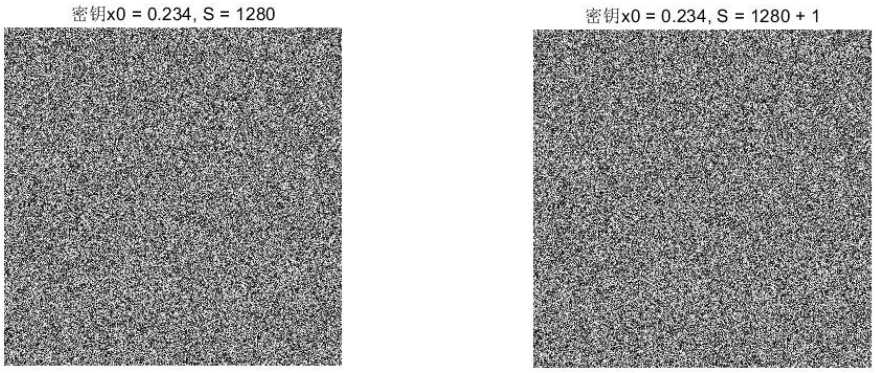


Fig 4.2.2 密钥 S 发生微小变动后密文图像的变化.

1. 同一幅明文图像 `fruit.png` 在密钥 `x0` 为 `0.234`，而 `S` 分别为 `1280`，`1281` 下加密得到的两幅密文图像的像素变化率 `R` 和归一化平均变化强度 `U` 分别为：
2. `R = 1`
3. `U = 0.0080652`

结果表明 R 值接近理想值 0.9961，U 值与理想值 0.3446 相差较大，说明该加密算法对密钥 S 的敏感性强，但是对于针对密钥 S 的差分攻击的抵御力较弱。

实验结果（针对解钥 x_0 ）

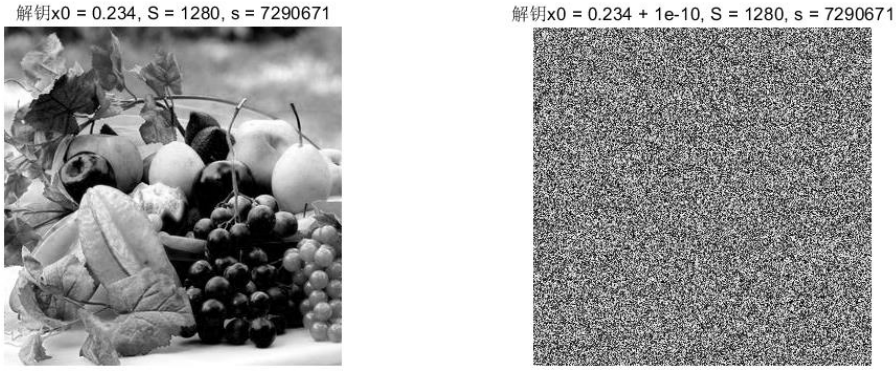


Fig 4.2.3 解钥 x0 发生微小变动后解密图像的变化.

1. 同一幅密文图像在解钥 `S = 1280，s = 13122840`，而 `x0` 分别为 `0.234，0.234 + 1e-10` 下解密得到的两幅明文图像的像素变化率 `R` 和归一化平均变化强度 `U` 分别为：
2. `R = 0.99598`
3. `U = 0.32742`

结果表明 R 值接近理想值 0.9961，U 值也接近理想值 0.3446，说明该解密算法对于解钥 x_0 的敏感性强，且对于针对解钥 x_0 的差分攻击抵御力较强。

实验结果（针对解钥 s ）

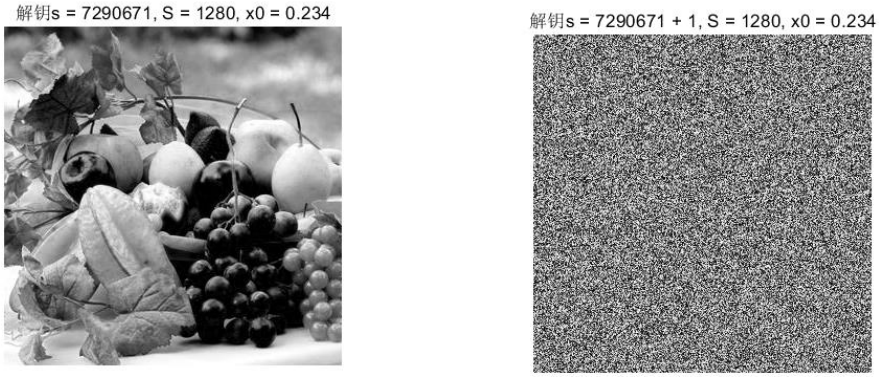


Fig 4.2.4 解钥 s 发生微小变动后解密图像的变化.

1. 同一幅密文图像在解钥 `x0 = 0.234, S = 1280,` 而 s 分别为 `13122840, 13122840 + 1` 下解密得到的两幅明文图像的像素变化率 R 和归一化平均变化强度 U 分别为:
2. `R = 0.99599`
3. `U = 0.32674`

结果表明 R 值接近理想值 0.9961，U 值也接近理想值 0.3446，说明该解密算法对于解钥 s 的敏感性强，且对于针对解钥 s 的差分攻击抵御力较强。

实验结果（针对解钥 S ）

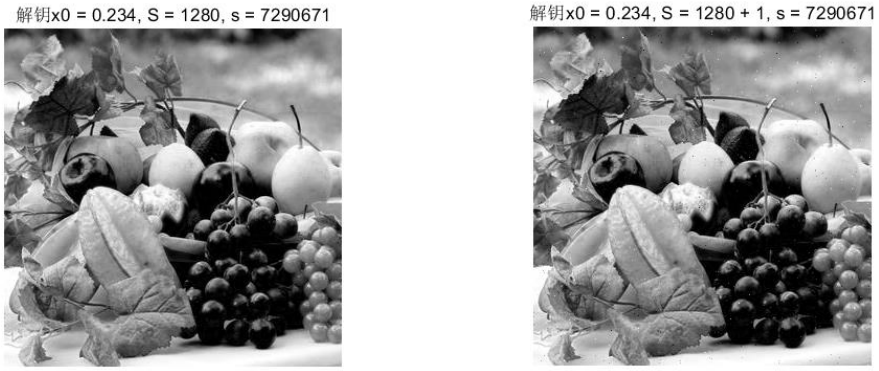


Fig 4.2.5 解钥 S 发生微小变动后解密图像的变化.

1. 同一幅密文图像在解钥 `x0 = 0.234, s = 13122840,` 而 S 分别为 `1280, 1280 + 1` 下解密得到的两幅明文图像的像素变化率 R 和归一化平均变化强度 U 分别为:
2. `R = 0.012748`
3. `U = 0.0011108`

结果表明 R 值与理想值 0.9961 相差甚大，U 值与其理想值 0.3446 偏差也很大，因此该解密算法对于解钥 S 的敏感性较弱，且对于针对解钥 S 的差分攻击的抵御力较弱。

§ 4.3 像素相关性分析

评估指标

相邻像素的相关系数 r，计算公式为

$$r_{xy}=\frac{Cov(x,y)}{Cov(x,x)\bullet Cov(y,y)},$$

$$Cov(x,y)=\frac{1}{n}\sum_{i=1}^{n}\left(x_i-E(x)\right)\left(y_i-E(y)\right),$$

$$E(x)=\frac{1}{n}\sum_{i=1}^{n}x_i.$$

其中 $Cov(x,y)$ 是向量 x 与向量 y 的协方差，x_i,y_i 是向量 x,y 的第 i 个分量，n 是向量 x,y 的长度，$E(x),E(y)$ 就是向量 x,y 各自所有分量的均值。r_{xy} 的值介于 -1 到 1 之间，其绝对值 $|r_{xy}|$ 越接近 1，表示向量 x,y 的相关程度越大。

已知一般的明文图像像素在水平，垂直，或对角方向上高度相关，在密文图像中，相邻像素之间的相关性应该显著降低。

分析方法

采用 Fig 4.1.1 的 fruit.png 作为明文图像，然后：

① 将明文图像进行加密得到密文图像；

② 规定好相邻像素位于原像素的什么方向（左方，右方，上方，下方，左上方，左下方，右上方，右下方）；

③ 随机在明文图像中抽取 n 对相邻像素形成两个长度为 n 的向量 X_1,Y_1，算其相关系数 r_1；

④ 随机在密文图像中抽取 n 对相邻像素形成两个长度为 n 的向量 X_2,Y_2，算其相关系数 r_2。

以下令 $n=20000$，令密，解钥 $x_0=0.234$，$S=1280$。

实验结果

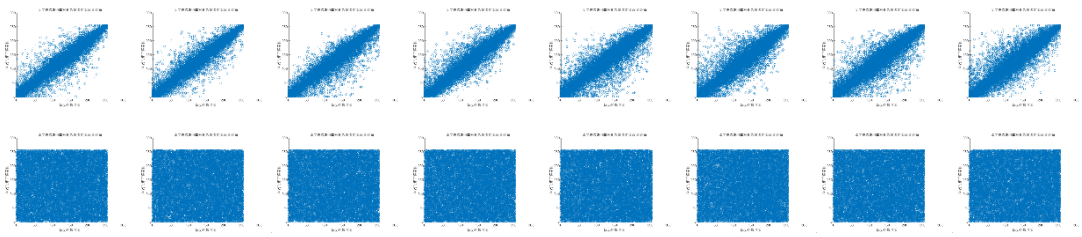


Fig 4.3.1 20000 次抽取的相邻像素的灰度值二维分布图，
从左到右对应的相邻像素方向依次为左，右，上，下，左上，左下，右上，右下.

```
'Left'
明文图像随机取 20000 对相邻像素计算得相关系数: 0.97101
密文图像随机取 20000 对相邻像素计算得相关系数: 0.0045093

'Right'
明文图像随机取 20000 对相邻像素计算得相关系数: 0.97189
密文图像随机取 20000 对相邻像素计算得相关系数: -0.0059508

'Up'
明文图像随机取 20000 对相邻像素计算得相关系数: 0.96464
密文图像随机取 20000 对相邻像素计算得相关系数: -0.0044817

'Down'
明文图像随机取 20000 对相邻像素计算得相关系数: 0.96428
密文图像随机取 20000 对相邻像素计算得相关系数: 0.012069

'Left_Up'
明文图像随机取 20000 对相邻像素计算得相关系数: 0.94745
密文图像随机取 20000 对相邻像素计算得相关系数: -0.0022895

'Left_Down'
明文图像随机取 20000 对相邻像素计算得相关系数: 0.94837
密文图像随机取 20000 对相邻像素计算得相关系数: -0.0056973

'Right_Up'
明文图像随机取 20000 对相邻像素计算得相关系数: 0.94772
密文图像随机取 20000 对相邻像素计算得相关系数: 0.014249

'Right_Down'
明文图像随机取 20000 对相邻像素计算得相关系数: 0.94375
密文图像随机取 20000 对相邻像素计算得相关系数: -0.0023908
```

实验结果表明无论规定相邻像素位于原像素的哪个方向，明文图像的相邻像素与原像素灰度值的二维分布呈现近似线性相关关系，相关系数接近 1；而密文图像的相邻像素与原像素灰度值的二维分布非常均匀，布满平面，相关系数接近 0. 因此加密算法满足零相关，加密效果较好，密文图像具有更低的像素相关性。

§ 4.4 图像安全性分析

评估指标

信息熵，一种用来检测加密系统安全性强度的参数，计算公式为

$$H(m)=\sum_{i=0}^{2N-1}p(m_i)\log_2\frac{1}{p(m_i)}.$$

式中，m_i 表示第 i 位灰度级，$m_i=0,1,\cdots,255$，$p(m_i)$ 表示像素值为 m_i 的概率，一般当图像像素个数足够多的时候，可以用频率代替，N 表示在密文图像中所有像素的个数。

对于一幅密文图像，它的理想的信息熵的值为 8，在这种情况下，密文图像就不会向那些试图获得没有授权访问的任何人泄露任何有用的信息。

实验结果

计算了由 Fig 4.1.1 加密（密钥 $x_0=0.234,\ S=1280$）得到的密文图像的信息熵，为 7.9982，非常接近 8，表明加密算法在进行对图像加密的过程中可以避免发生信息泄露，该图像加密算法具有良好的抗熵值分析攻击。

§ 4.5 计算量增长分析

这里我们就简单研究在同一台计算机上，保持密钥和其他条件不变，加（解）密算法用时与输入的明文图像大小的关系。为了方便，我们规定输入的明文图像大小是 $N\times N$ 的。

令 $N=50,\ 100,\ 150,\ \cdots,\ 1000$，密，解钥 $x_0=0.234,\ S=1280$，计算加（解）密算法用时得到的结果如下。

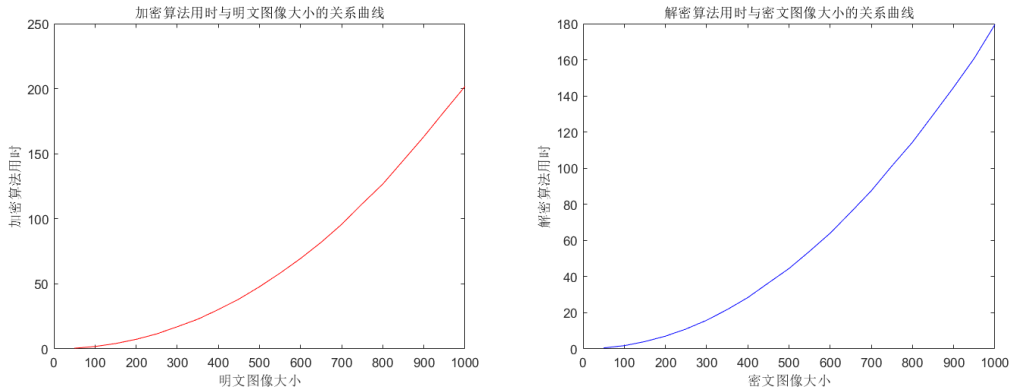


Fig 4.5.1 加，解密算法与明文图像大小的关系曲线.

```
历时 0.549801 秒。
历时 0.547340 秒。
图像大小为: 50 * 50 的加解密已完成
历时 1.837155 秒。
```

```
历时 1.809797 秒。
图像大小为: 100 * 100 的加解密已完成
历时 4.154024 秒。
历时 4.064169 秒。
图像大小为: 150 * 150 的加解密已完成
历时 7.362790 秒。
历时 7.058512 秒。
图像大小为: 200 * 200 的加解密已完成
历时 11.549046 秒。
历时 11.026304 秒。
图像大小为: 250 * 250 的加解密已完成
历时 17.058409 秒。
历时 15.857187 秒。
图像大小为: 300 * 300 的加解密已完成
历时 22.856086 秒。
历时 21.770861 秒。
图像大小为: 350 * 350 的加解密已完成
历时 30.244508 秒。
历时 28.422448 秒。
图像大小为: 400 * 400 的加解密已完成
历时 38.321001 秒。
历时 36.493705 秒。
图像大小为: 450 * 450 的加解密已完成
历时 47.782701 秒。
历时 44.477456 秒。
图像大小为: 500 * 500 的加解密已完成
历时 58.232867 秒。
历时 54.052163 秒。
图像大小为: 550 * 550 的加解密已完成
历时 69.501967 秒。
历时 64.052201 秒。
图像大小为: 600 * 600 的加解密已完成
历时 81.940872 秒。
历时 75.634364 秒。
图像大小为: 650 * 650 的加解密已完成
历时 95.774720 秒。
历时 87.555124 秒。
图像大小为: 700 * 700 的加解密已完成
历时 111.549318 秒。
历时 101.265168 秒。
图像大小为: 750 * 750 的加解密已完成
历时 126.735671 秒。
历时 114.441286 秒。
图像大小为: 800 * 800 的加解密已完成
```

```
历时 144.922362 秒。
历时 129.592058 秒。
图像大小为: 850 * 850 的加解密已完成
历时 163.219478 秒。
历时 144.857971 秒。
图像大小为: 900 * 900 的加解密已完成
历时 182.688934 秒。
历时 160.887248 秒。
图像大小为: 950 * 950 的加解密已完成
历时 201.769145 秒。
历时 179.507409 秒。
图像大小为: 1000 * 1000 的加解密已完成
```

结果与原文[1]分析得到的复杂度

$$O\left(8\times N^2+\left(8\times N\right)^2+N^2+8\times N^2\right)=O\left(81N^2\right)=O\left(N^2\right)$$

是很吻合的。

参考文献

[1] Ping P, Li J H, Mao Y C, Qi R Z. Image encryption algorithm based on chaotic maps and bit reconstruction [J]. Journal of Image and Graphics, 2017, 22(10): 1348-1355. [DOI: 10.11834/jig.170049].

[2] Huang X L, Ye G D. An efficient self-adaptive model for chaotic image encryption algorithm [J]. Communications in Nonlinear Science and Numerical Simulation, 2014, 19(12): 4094-4104. [DOI: 10. 1016/j. cnsns. 2014. 04. 012].

[3] Ding W, Yan W Q, Qi D X. Digital image scrambling and digital watermarking technologu based on Conway's game [J]. Journal of North China University of Technology, 2000, 12(1): 1-5.

[4] Fu C, Lin B B, Miao Y S, et al. A novel chaos-based bit-level permutation scheme for digital image encryption [J]. Optics Communications, 2011, 284(23): 5415-5423. [DOI: 10. 1016/j. optcom. 2011. 08. 013].

[5] Zhou Y C, Cao W J, Chen C L P. Image encryption using binary bitplane [J]. Signal Processing, 2014, 100: 197-207. [DOI: 10.1109/FSKD.2015.7382351].

[6] Teng L, Wang X Y. A bit-level image encryption algorithm based on spatiotemporal chaotic system and self-adaptive [J]. Optics Communications, 2012, 285(20): 4048-4054. [DOI: 10. 1016/j. optcom. 2012. 06. 004].

[7] Zhang W, Wong K W, Yu H, et al. A symmetric color image encryption algorithm using the intrinsic features of bit distributions [J]. Communications in Nonlinear Science and Numerical Simulation, 2013, 18(3): 584-600. [DOI: 10. 1016/j. cnsns. 2012. 08. 010].

[8] Ping P, Mao Y C, Lv X, et al. An image scrambling algorithm using discrete Henon map [C] // Proceedings of 2015 IEEE International Conference on Information and Automation. Lijiang: IEEE, 2015: 429-432. [DOI: 10.1109/ICInfA. 2015. 7279326].

[9] Zhou Q, Liao X F. Collision-based flexible image encryption algorithm [J]. Journal of Systems and Software, 2012, 85(2): 400-407. [DOI: 10. 1016/j. jss. 2011. 08. 032].

[10] Hussain I, Shah T, Gondal M A. Application of S-box and chaotic map for image encryption [J]. Mathematical and Computer Modelling, 2013, 57(9-10): 2576-2579. [DOI: 10. 1016/j. mcm. 2013. 01. 009].

附录—加解密算法实现与函数说明

m 文件名称	m 文件功能
tent_mapping_once.m	对输入的初值 x0 按照输入的系统参数 mju 执行一次 Tent 映射得到 x1
henon_mapping_once.m	对输入的二元组(x0, y0)按照输入的系统参数 a, c, N 执行一次可逆 Henon 映射得到(x1, y1)
henon_mappinginv_once.m	对输入的二元组(x1, y1)按照输入的系统参数 a, c, N 执行一次 Henon 逆映射得到(x0, y0)
cmbr_encryption.m	对输入的明文图像按输入的密钥进行加密
cmbr_decryption.m	对输入的密文图像按输入的解钥进行解密
r_xy_near.m	按照输入的数量 n 和规定的相邻像素方向 direction 随机抽取输入的明文图像中的 n 对相邻像素并计算相关系数
simulation_experiment.m	对原文加解密算法（的改进形式）的仿真实验
performance_analysis.m	对原文加密算法的一系列性能分析

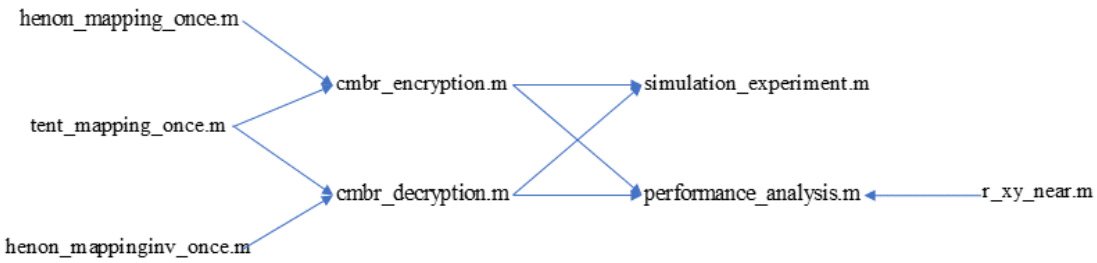


Fig 各 m 文件调用逻辑图.

tent_mapping_once.m

```
function y = tent_mapping_once( x, mju )
%% 说明
%   功能
%   该函数对输入的 x 作一次 Tent 映射，得到 y，映射的表达式为
%   if 0 < x <= 0.5, y = mju * x
%   if 0.5 < x < 1, y = mju * (1 - x)
%   即 y = mju * min{x, (1-x)}

%   参数
%   x: 输入映射系统的值，要求是大于 0 且小于 1 的实数
%   y: 映射系统输出的值，一定是一个大于 0 小于 1 的实数
```

```
%   mju: Tent 映射系统本身的参数值

%% 执行映射
if x <= 0.5
    y = mju * x;
else
    y = mju * (1 - x);
end

end
```

henon_mapping_once.m

```
function [ x1, y1 ] = henon_mapping_once( x0, y0, M, N, a, c )
%% 说明
%   功能
%   该函数将输入的 x0, y0 作一次可逆的 Henon 映射, 映射的表达式为
%   x1 = 1 - a * (x0 ^ 2) + y0 mod M
%   y1 = x0 + c mod N

%   注: Henon 映射当 M == N 时是双射, 是可逆的; 而当 M ~= N 时不一定是双射.
%       事实上, 当 M < N 时, 显然地, Henon 映射不是满射, 更不是双射.

%   参数
%   x0, y0: 输入的一个二元组, 代表映射前的原(图像 / 比特)矩阵的某一像素的二维坐标
%   x1, y1: 输出的一个二元组, 代表原坐标(x0, y0)经映射后在新(图像 / 比特)矩阵中的位置
%   M, N: 分别代表图像(比特)矩阵的行数和列数
%   a, c: Henon 映射系统的参数. 要求 a 是大于 0 的实数
%% 执行映射
b = 1;
x1 = mod( 1 - a * (x0 ^ 2) + y0, M );
y1 = mod( b * x0 + c, N );

end
```

henon_mappinginv_once.m

```
function [x0, y0] = henon_mappinginv_once( x1, y1, M, N, a, c)
%% 说明
%   功能
%   对输入的二元组 x1, y1 作一次 Henon 逆映射得到 x0, y0, 映射的表达式为
```

```
%   x0 = y1 - c mod N
%   y0 = x1 - 1 + a * (x0 ^ 2) mod M

%   注：这里的"Henon 逆映射"并不是严格数学意义上的逆映射，应该理解为逆操作
%       举个例子，显然地，当 M < N 时，Henon 映射不是一个满射，自然就不是一个双射，不可逆

%   参数
%   x1, y1: 输入的一个二元组，代表逆映射前的(图像 / 比特)矩阵的某一像素的二维坐标
%   x0, y0: 输出的一个二元组，代表原坐标(x1, y1)经逆向映射后在新(图像 / 比特)矩阵中的位置
%   M, N: 分别代表图像(比特)矩阵的行数和列数
%   a, c: Henon 逆映射系统的参数. 要求 a 是大于 0 的实数

%% 执行逆映射
x0 = mod( y1 - c, N );
y0 = mod( x1 - 1 + a * (x0 ^ 2), M );

end
```

cmbr_encryption.m

```
function [ C, C0, s ] = cmbr_encryption( I, x0, S )
%% 说明
%   cmbr 是 chaotic mapping & bit recombination 的缩写
%   功能
%   1. 计算原图像 I 的像素灰度值总和 s，作为解密程序所用的解钥之一
%   2. 先利用混沌映射和比特重组对明文图像 I 进行两阶段的置乱加密得到中间密文图像 C0
%       再对中间密文图像 C0 进行灰度(扩散)加密得到最终密文图像 C
%   注意程序编写过程中默认图像是 256 个灰度级的

%   参数
%   x0: 输入 Tent 系统进行迭代的初值，相当于加密系统的密钥
%   I: 输入的明文图像
%   S: 输入的密钥之一，供第三阶段的灰度加密用
%   s: 输出的明文图像像素灰度值总和，作为解密系统的解钥之一
%   C0: 经过两阶段置乱加密得到的中间密文图像
%   C: C0 再经过扩散加密得到的最终密文图像

%% 输入参数的检验

% 要求初值 x0 是一个大于 0 且小于 1 的实数
```

```
if ~isreal(x0) || x0 <= 0 || x0 >= 1
    error('Tent 映射初值 x0 必须是一个大于 0 且小于 1 的实数');
end

%% 第一阶段: 全局置乱加密

% 初始化参数
I = double(I);  % 数据类型变为 double 类
[M, N] = size(I);  % 计算图像尺寸
s = sum( sum(I) );  % 计算图像 I 中像素灰度值的总和
mju = 2 ^ ( s / (M * N * 255) );  % 设置 Tent 混沌系统的控制参数 mju
times = 10 ^ 3 + mod(s, 10 ^ 3);  % 设置 Tent 混沌系统初始迭代的次数 times

% 对 Tent 混沌系统输入初始密钥 x0，并根据控制参数 mju 进行 k 次迭代消除初态效应的影响
x_iter = [];
for i = 1 : times
    x_iter = tent_mapping_once(x0, mju);
    x0 = x_iter;
end

% 将像素矩阵中每个像素灰度值转换成 8 位二进制数，
% 原矩阵中每个像素(二进制)灰度值的每一位都作为新矩阵 II 的一个元素
% 因此新矩阵 II 的规模是 M * (8 * N)
II = [];
for i = 1 : M
    II_row = [];
    for j = 1 : N
        II_row = [II_row, zeros(1, 8)];
        t = dec2bin( I(i, j) );  % 将灰度值 I(i,j)转化为二进制，格式是 string

        % 从 string 格式的 t 中以 double 格式提取每一位，赋到相应的位置
        for k = 1 : numel(t)
            II_row(end - k + 1) = str2double( t(end - k + 1) );
        end
    end
    II = [II ; II_row];
end

% Tent 混沌系统继续迭代 M 次，产生长度为 M 的混沌序列 E
% 将 E 按照升序排序，得到位置向量 P_E，利用生成的位置向量 P_E
% 对已经转成比特的数字图像矩阵 II 进行整行置乱
E = zeros(1, M);
for i = 1 : M
```

```
    E(i) = tent_mapping_once( x0, mju );
    x0 = E(i);
end

[~, P_E] = sort(E);
II = II( P_E, : );

% Tent 混沌系统继续迭代 8 * N 次，产生长度为 8 * N 的混沌序列 F
% 将 F 按照升序排序，得到位置向量 P_F，利用生成的位置向量 P_F
% 对数字图像矩阵 II 进行整列置乱
F = zeros(1, 8 * N);
for i = 1 : 8 * N
    F(i) = tent_mapping_once( x0, mju );
    x0 = F(i);
end

[~, P_F] = sort(F);
II = II( :, P_F );

%% 第二阶段：各 Bit 面置乱加密

% 将第一阶段得到的置乱矩阵 II 从左到右均分成 8 个 M * N 的比特矩阵
% 对 8 个矩阵分别使用 Henon 映射进行置乱，具体操作是，每个矩阵分割出若干个方形区域，
% 对每个方形区域进行 Henon 映射置乱

% 求出每个 Bit 面对应的 Henon 映射系统参数 a，c，iter_times
a = zeros(1, 8);  c = zeros(1, 8);  iter_times = zeros(1, 8);
for bit = 1 : 8
    F_index = 1 + 8 * N * (bit - 1) / 8;
    iter_times(bit) = 1 + mod(  ceil( F(F_index) * (10 ^ 14) ), 5  );
    a(bit) = mod(  ceil( F(F_index) * (10 ^ 14) ), 2 ^ 8  );
    c(bit) = mod(  ceil( (F(F_index) ^ 2) * (10 ^ 14) ), 2 ^ 8  );
end

% 对每个 Bit 面开始划分方形区域并在每个方形区域进行 Henon 映射置乱
t = zeros(M, 8 * N);
mark_h = 0;  mark_w = 0;
min_hw = min(M, N);  max_hw = max(M, N);
if max_hw == M
    delta_h = min_hw;  delta_w = 0;
else
    delta_h = 0;  delta_w = min_hw;
end
```

```
107.
108. while 1
109.     r = mod(max_hw, min_hw);  q = ( max_hw - r ) / min_hw;
110.     for bit = 1 : 8
111.         for i = 0 : min_hw - 1
112.             for j = 0 : min_hw - 1
113.                 x_pos1 = i;  y_pos1 = j;
114.                 for iter = 1 : iter_times(bit)
115.                     [x_pos2, y_pos2] = ...
116.                         henon_mapping_once( x_pos1, y_pos1, ...
117.                             min_hw, min_hw, a(bit), c(bit) );
118.                     x_pos1 = x_pos2;  y_pos1 = y_pos2;
119.                 end
120.                 for num = 1 : q
121.                     t( mark_h + (num - 1) * delta_h + x_pos2 + 1, ...
122.                        (bit - 1) * N + mark_w + (num - 1) * delta_w + y_pos
    2 + 1) = ...
123.                     II( mark_h + (num - 1) * delta_h + i + 1, ...
124.                        (bit - 1) * N + mark_w + (num - 1) * delta_w + j + 1
    );
125.                 end
126.             end
127.         end
128.     end
129.
130.     if r == 0
131.         break
132.     else
133.         max_hw = min_hw;  min_hw = r;
134.         mark_h = mark_h + q * delta_h;  mark_w = mark_w + q * delta_w;
135.         if delta_h == 0
136.             delta_h = min_hw;  delta_w = 0;
137.         else
138.             delta_h = 0;  delta_w = min_hw;
139.         end
140.     end
141.
142. end
143. II = t;
144.
145. % 8 个比特矩阵各自迭代完成后，将这 8 个比特矩阵按垂直于 Bit 平面的方向合并，将比特转
    化为
146. % 十进制像素值，得到中间密文图像 C0
147. C0 = zeros(M, N);
```

```
for i = 1 : M
    for j = 1 : N
        t = '';
        for bit = 1 : 8
            t = [ t, num2str( II( i, j + N * (bit - 1) ) ) ];
        end
        C0(i ,j) = bin2dec(t);
    end
end

%% 第三阶段: 中间密文图像 C0 再加密 -- 灰度(扩散)加密

% Tent 混沌系统继续迭代 M * N 次, 由此产生长度为 M * N 的混沌序列 R
R = zeros(1, M * N);
for i = 1 : M * N
    R(i) = tent_mapping_once( x0, mju );
    x0 = R(i);
end

% 进行简单的扩散加密, 将混沌序列 R 中的 M * N 个元素与 C0 中的 M * N 个元素一一对应起来
% 加密后的矩阵 C 的每个像素(i, j)处的灰度值的计算公式为
% C(i, j) = ( D(i, j) + C0(i, j) ) mod 2 ^ 8 = 256
% 其中 D(i, j) = ceil( R(i, j) * 2 ^ 48 ) mod 2 ^ 8 = 256
C = zeros(M, N);
for i = 1 : M
    for j = 1 : N
        R_index = N * (i - 1) + j;
        D = mod( ceil( R(R_index) * (2 ^ 48) ), 2 ^ 8 );  % 用到混沌序列 R 中对应的值
        if i == 1 || j == 1
            T = S;
        else
            t2 = mod(N * (i - 1) + j - 1, N);
            t1 = 1 + ( N * (i - 1) + j - 1 - t2 ) / N;
            T = C( t1, t2 );
        end
        C(i, j) = mod( D + C0(i, j) + T, 2 ^ 8 );
    end
end
C0 = uint8(C0);
C = uint8(C);

end
```

cmbr_decryption.m

```
function [ I, C0 ] = cmbr_decryption( C, x0, s, S )
%% 说明
%   功能
%   cmbr 是 chaotic mapping & bit recombination 的缩写
%   先对初始密文图像 C 进行灰度加密逆操作得到中间密文图像 C0
%   再对 C0 进行两个阶段的置乱逆操作得到解密图像 I
%   注意程序编写过程中默认图像是 256 个灰度级的

%   参数
%   C: 初始密文图像
%   x0: 输入 Tent 系统进行迭代的初值，是解密系统的解钥之一
%   s: 原明文图像的灰度总和，是解密系统的解钥之一
%   S: 解密系统的解钥之一
%   C0: 解密过程得到的中间密文图像
%   I: 解密过程得到的最终的解密图像

%% 输入参数的检验

% 要求初值 x0 是一个大于 0 且小于 1 的实数
if ~isreal(x0) || x0 <= 0 || x0 >= 1
    error('Tent 映射初值 x0 必须是一个大于 0 且小于 1 的实数');
end

%% 利用解钥 s 还原出 Tent 混沌系统的控制参数 mju 和初始迭代次数 times
% 然后一次性生成解密用的混沌序列 E, F, R

% 初始化，生成 Tent 混沌系统参数
C = double(C);
[M ,N] = size(C);
mju = 2 ^ ( s / (M * N * 255) );
times = 10 ^ 3 + mod(s, 10 ^ 3);

% 迭代 times 次消除初态效应的影响
x_iter = [];
for i = 1 : times
    x_iter = tent_mapping_once(x0, mju);
    x0 = x_iter;
end

% 生成混沌序列 E
E = zeros(1, M);
for i = 1 : M
```

```
    E(i) = tent_mapping_once(x0, mju);
    x0 = E(i);
end

% 生成混沌序列 F
F = zeros(1, 8 * N);
for i = 1 : 8 * N
    F(i) = tent_mapping_once(x0, mju);
    x0 = F(i);
end

% 生成混沌序列 R
R = zeros(1, M * N);
for i = 1 : M * N
    R(i) = tent_mapping_once(x0, mju);
    x0 = R(i);
end

%% 第一阶段：由初始密文图像 C 还原出中间密文图像 C0
C0 = zeros(M, N);
for i = 1 : M
    for j = 1 : N
        R_index = N * (i - 1) + j;
        D = mod( ceil( R(R_index) * (2 ^ 48) ), 2 ^ 8 );  % 用到混沌序列 R 中对应的值
        if i == 1 || j == 1
            T = S;
        else
            t2 = mod(N * (i - 1) + j - 1, N);
            t1 = 1 + ( N * (i - 1) + j - 1 - t2 ) / N;
            T = C( t1, t2 );
        end
        C0(i, j) = mod( C(i, j) - D - T, 2 ^ 8 );
%         while C0(i, j) < 0
%             C0(i, j) = C0(i, j) + 2 ^ 8;
%         end
    end
end

%% 第二阶段：对中间密文图像 C0 分解成 8 个 Bit 矩阵，对每个矩阵实行 henon 逆映射

II = zeros( M, 8 * N );
for i = 1 : M
    for j = 1 : N
```

```
        t = dec2bin( C0(i, j) );
        for k = 1 : numel(t)
            II(i, j + N * (8 - k)) = str2double( t(end - k + 1) );
        end
    end
end

% 求出每个 Bit 面对应的 Henon 映射系统参数 a, c, iter_times
a = zeros(1, 8);  c = zeros(1, 8);  iter_times = zeros(1, 8);
for bit = 1 : 8
    F_index = 1 + 8 * N * (bit - 1) / 8;
    iter_times(bit) = 1 + mod(  ceil( F(F_index) * (10 ^ 14) ), 5  );
    a(bit) = mod(  ceil( F(F_index) * (10 ^ 14) ), 2 ^ 8  );
    c(bit) = mod(  ceil( (F(F_index) ^ 2) * (10 ^ 14) ), 2 ^ 8  );
end

% 对每个 Bit 面开始划分方形区域并在每个方形区域进行 Henon 置乱逆操作
t = zeros(M, 8 * N);
mark_h = 0;  mark_w = 0;
min_hw = min(M, N);  max_hw = max(M, N);
if max_hw == M
    delta_h = min_hw;  delta_w = 0;
else
    delta_h = 0;  delta_w = min_hw;
end

while 1
    r = mod(max_hw, min_hw);  q = ( max_hw - r ) / min_hw;
    for bit = 1 : 8
        for i = 0 : min_hw - 1
            for j = 0 : min_hw - 1
                x_pos2 = i;  y_pos2 = j;
                for iter = 1 : iter_times(bit)
                    [x_pos1, y_pos1] = ...
                        henon_mappinginv_once( x_pos2, y_pos2, ...
                            min_hw, min_hw, a(bit), c(bit) );
                    x_pos2 = x_pos1;  y_pos2 = y_pos1;
                end
                for num = 1 : q
                    t( mark_h + (num - 1) * delta_h + x_pos1 + 1, ...
                       (bit - 1) * N + mark_w + (num - 1) * delta_w + y_pos
   1 + 1) = ...
                    II( mark_h + (num - 1) * delta_h + i + 1, ...
```

```
                        (bit - 1) * N + mark_w + (num - 1) * delta_w + j + 1
    );
                end
            end
        end
    end

    if r == 0
        break
    else
        max_hw = min_hw;  min_hw = r;
        mark_h = mark_h + q * delta_h;  mark_w = mark_w + q * delta_w;
        if delta_h == 0
            delta_h = min_hw;  delta_w = 0;
        else
            delta_h = 0;  delta_w = min_hw;
        end
    end

end
II = t;

%% 第三阶段：对进行 henon 逆映射后的数字图像 II 再进行先整列后整行的逆置乱操作
% 然后将 8 个 Bit 矩阵合并成十进制像素灰度值的明文图像矩阵 I

% 整列逆置乱
[~, P_F] = sort(F);
II( :, P_F ) = II;

% 整行逆置乱
[~, P_E] = sort(E);
II( P_E, : ) = II;

% Bit 矩阵合并
I = zeros(M, N);
for i = 1 : M
    for j = 1 : N
        t = '';
        for pos = 1 : 8
            t = [ t, num2str( II( i, 8 * (j - 1) + pos ) ) ];
        end
        I(i ,j) = bin2dec(t);
    end
end
```

```
171.
172. C0 = uint8(C0);
173. I = uint8(I);
174.
175. end
```

r_xy_near.m

```
1.  function [ r_xy, X, Y ] = r_xy_near( I, n, direction )
2.  % 参数说明
3.  % I -- 输入的图像灰度矩阵
4.  % n -- 在 I 中按照一定要求随机抽取的像素点的个数
5.  % direction -- 将 direction 指定方向的像素作为相邻像素
6.  %               Left: 水平向左的像素
7.  %               Right: 水平向右的像素
8.  %               Down: 垂直向下的像素
9.  %               Up; 垂直向上的像素
10. %               Left_Up: 对角线左上角的像素
11. %               Left_Down: 对角线左下角的像素
12. %               Right_Up: 对角线右上角的像素
13. %               Right_Down: 对角线右下角的像素
14. % r_xy -- 最终计算得到的两个长度为 n 的向量 X 和 Y 之间的相关系数
15. % X -- 随机抽取的像素点的灰度值（向量）
16. % Y -- 随机抽取的像素点的相邻点的灰度值（向量）
17.
18. I = double(I);
19. [H, W] = size(I);
20.
21. % 根据 direction 参数确定随机取样点的横纵坐标范围
22. switch direction
23.     case 'Left'
24.         R = floor( unifrnd(1, H, [1, n]) );
25.         C = floor( unifrnd(2, W, [1, n]) );
26.         delta_r = 0; delta_c = -1;
27.     case 'Right'
28.         R = floor( unifrnd(1, H, [1, n]) );
29.         C = floor( unifrnd(1, W-1, [1, n]) );
30.         delta_r = 0; delta_c = 1;
31.     case 'Up'
32.         R = floor( unifrnd(2, H, [1, n]) );
33.         C = floor( unifrnd(1, W, [1, n]) );
34.         delta_r = -1; delta_c = 0;
35.     case 'Down'
36.         R = floor( unifrnd(1, H-1, [1, n]) );
```

```
        C = floor( unifrnd(1, W, [1, n]) );
        delta_r = 1; delta_c = 0;
    case 'Left_Up'
        R = floor( unifrnd(2, H, [1, n]) );
        C = floor( unifrnd(2, W, [1, n]) );
        delta_r = -1; delta_c = -1;
    case 'Left_Down'
        R = floor( unifrnd(1, H-1, [1, n]) );
        C = floor( unifrnd(2, W, [1, n]) );
        delta_r = 1; delta_c = -1;
    case 'Right_Up'
        R = floor( unifrnd(2, H, [1, n]) );
        C = floor( unifrnd(1, W-1, [1, n]) );
        delta_r = -1; delta_c = 1;
    case 'Right_Down'
        R = floor( unifrnd(1, H-1, [1, n]) );
        C = floor( unifrnd(1, W-1, [1, n]) );
        delta_r = 1; delta_c = 1;
end

X = []; Y = [];  % 初始化随机取样点处的灰度值向量
for i = 1 : n
    X(i) = I( R(i), C(i) );
    Y(i) = I( R(i) + delta_r, C(i) + delta_c );
end

% 计算向量 X 与向量 Y 的相关系数
mx = mean(X); my = mean(Y);
cov_xy = mean( (X - mx) .* (Y - my) );
cov_x = mean( (X - mx) .* (X - mx) );
cov_y = mean( (Y - my) .* (Y - my) );
r_xy = cov_xy / sqrt( cov_x * cov_y );

end
```

simulation_experiment.m

```
%%  m 文件说明
%   对《混沌映射与比特重组的图像加密》一文的图像加密算法和解密算法的仿真实验
%   需要调用自定义函数文件 cmbr_encryption.m, cmbr_decryption.m

%%  初始化与设置参数
clear;  clc;  close all;

```

```
% I0 = imread('fruit.png');  % 与原文相似的蔬果图，大小为 323 * 500
% I0 = I0(:, 1:323);  % 截取方形区域

% I0 = imread('cat.png');  % 招财猫图，大小为 651 * 500

% I0 = zeros(256, 256);  % 结果：仿真能够进行

I0 = ones(256, 256) .* 255;  % 结果：仿真能够进行

x0 = 0.234;  S = 1280;

%%  加密部分

%   展示原始明文图像 I0 和其直方图
figure,  imshow(I0);  title('明文图像');
figure,  imhist(I0);  title('明文图像直方图');

%   用密钥 x0 对明文图像 I0 进行加密，得到中间密文图像 C0，密文图像 C 和解钥 s，并计时
tic
[ C, C0, s ] = cmbr_encryption( I0, x0, S );
toc

%   展示中间密文图像 C0，最终密文图像 C，和各自的直方图
figure,  imshow(C0);  title('中间密文图像');
figure,  imhist(C0);  title('中间密文图像直方图');

figure,  imshow(C);  title('最终密文图像');
figure,  imhist(C);  title('最终密文图像直方图');

% close all
%%  解密部分

%   展示初始密文图像 C 和其直方图
figure,  imshow(C);  title('初始密文图像');
figure,  imhist(C);  title('初始密文图像直方图');

%   用解钥 x0 和 s 对密文图像 C 进行解密，得到中间密文图像 C1 和明文图像 I1，并计时
tic
[I1, C1] = cmbr_decryption( C, x0, s, S );
toc

%   展示中间密文图像 C1，明文图像 I1 以及各自的直方图
figure,  imshow(C1);  title('中间密文图像');
```

```
figure,  imhist(C1);  title('中间密文图像直方图');

figure,  imshow(I1);  title('解密所得最终图像');
figure,  imhist(I1);  title('解密所得最终图像直方图');

if isequal(C0, C1)
    disp('解密过程得到的中间密文图像与加密过程的一致，无损');
else
    disp('解密过程得到的中间密文图像与加密过程的不完全一致，有损');
end

if isequal(I0, I1)
    disp('解密过程得到的解密图像与原明文图像一致，无损');
else
    disp('解密过程得到的解密图像与原明文图像不完全一致，有损');
end

% close all
```

performance_analysis.m

```
%%  m 文件说明
%   对《混沌映射与比特重组的图像加密》一文的图像加密算法和解密算法的性能分析
%   以下的每个分析都需要调用到独立的自定义函数文件 cmbr_encryption.m, cmbr_decryption.m

%% 明文敏感性分析
%   分析方法
%   在明文图像中随机选取一个像素，将该像素的值随机调为 0 到 255 的另外一个
%   不同的值，再进行加密，计算相应的指标 R 和 U. 这样的操作过程重复若干次(例如 100 次)
%   需要额外调用到本 m 文件下方的函数 pixel_change_rate( )和
%   normalized_mean_change_intensity( )

%   初始化，设置参数
clear;  clc;  close all;
I = imread('fruit.png'); I = I(:, 1 : 323);  % 取方形区域
[H, W] = size(I);
exp_times = 100;  % 随机选取像素，随机调整其灰度值的过程重复操作的次数
x0 = 0.234;  S = 1280;  % 加密密钥

%   按照上述分析方法重复实验
x_pos = 1 + floor( ( H - 1 ) * rand(1, exp_times) );
y_pos = 1 + floor( ( W - 1 ) * rand(1, exp_times) );
```

```
R = zeros(1, exp_times);  % 初始化 R 的结果序列
U = zeros(1, exp_times);  % 初始化 U 的结果序列

tic
for k = 1 : exp_times
    I1 = I;
    t = I( x_pos(k), y_pos(k) );
    while I1( x_pos(k), y_pos(k) ) == t
        I1( x_pos(k), y_pos(k) ) = floor( rand * 255 );
    end
    [ C, ~, ~ ] = cmbr_encryption( I, x0, S );
    [ C1, ~, ~ ] = cmbr_encryption( I1, x0, S );
    R(k) = pixel_change_rate( C, C1 );
    U(k) = normalized_mean_change_intensity( C, C1 );
    disp(['第', num2str(k), '次操作完成，剩余', num2str(exp_times - k), '次']);
end
toc

%   可视化抽取的像素位置分布，可视化序列 R 和序列 U
figure,  scatter(x_pos, y_pos);  title([num2str(exp_times), '次抽取的像素位置分布']);
figure,  histogram(R);  title([num2str(exp_times), '次实验所得 R 直方图']);
figure,  histogram(U);  title([num2str(exp_times), '次实验所得 U 直方图']);

%%  密文敏感性分析
%   分析方法
%   在密文图像中随机选取一个像素，将该像素的值随机调为 0 到 255 的另外一个
%   不同的值，再进行解密，计算相应的指标 R 和 U. 这样的操作过程重复若干次(例如 100 次)
%   需要额外调用到本 m 文件下方的函数 pixel_change_rate( )和
%   normalized_mean_change_intensity( )

%   初始化，设置参数
clear;  clc;  close all;
I = imread('fruit.png');  I = I(:, 1 : 323);
[H, W] = size(I);
exp_times = 100;  % 随机选取像素，随机调整其灰度值的过程重复操作的次数
x0 = 0.234;  S = 1280;  % 加密密钥
[ C, ~, s ] = cmbr_encryption( I, x0, S );

%   按照上述分析方法重复实验
x_pos = 1 + floor( ( H - 1 ) * rand(1, exp_times) );
y_pos = 1 + floor( ( W - 1 ) * rand(1, exp_times) );
```

```
R = zeros(1, exp_times);  % 初始化 R 的结果序列
U = zeros(1, exp_times);  % 初始化 U 的结果序列

tic
for k = 1 : exp_times
    C1 = C;
    t = C( x_pos(k), y_pos(k) );
    while C1( x_pos(k), y_pos(k) ) == t
        C1( x_pos(k), y_pos(k) ) = floor( rand * 255 );
    end
    [ I0, ~ ] = cmbr_decryption( C, x0, s, S );
    [ I1, ~ ] = cmbr_decryption( C1, x0, s, S );
    R(k) = pixel_change_rate( I0, I1 );
    U(k) = normalized_mean_change_intensity( I0, I1 );
    disp(['第', num2str(k), '次操作完成，剩余', num2str(exp_times - k), '次
']);
end
toc

%   可视化抽取的像素位置分布，可视化序列 R 和序列 U
figure,  scatter(x_pos, y_pos);  title([num2str(exp_times), '次抽取的像素位置
分布']);
figure,  histogram(R);  title([num2str(exp_times), '次实验所得 R 直方图']);
figure,  histogram(U);  title([num2str(exp_times), '次实验所得 U 直方图']);

%%  解，密钥敏感性分析
%   需要额外调用到本 m 文件下方的函数 pixel_change_rate( )和
%   normalized_mean_change_intensity( )

clear;  clc;  close all;
I = imread('fruit.png');  I = I(:, 1 : 323);
x1 = 0.234;  x2 = 0.234 + 1e-10;
S1 = 1280;  S2 = 1280 + 1;
[ C1, ~, s1 ] = cmbr_encryption( I, x1, S1 );  s2 = s1 + 1;

%%   单独针对密钥 x0
clc;  close all;
[ C1, ~, ~ ] = cmbr_encryption( I, x1, S1 );
figure,  imshow(C1);  title('密钥 x0 = 0.234, S = 1280');
[ C2, ~, ~ ] = cmbr_encryption( I, x2, S1 );
figure,  imshow(C2);  title('密钥 x0 = 0.234 + 1e-10, S = 1280');
disp(['同一幅明文图像 fruit.png 在密钥 S = 1280, 而 x0 分别为 0.234, 0.234 + 1e-
10 下', ...
    '加密得到的两幅密文图像的像素变化率 R 和归一化平均变化强度 U 分别为:']);
```

```
disp(['R = ', num2str( pixel_change_rate( C1, C2 ) )]);
disp(['U = ', num2str( normalized_mean_change_intensity( C1, C2 ) )]);

%%   单独针对密钥 S
clc;  close all;
[ C1, ~, ~ ] = cmbr_encryption( I, x1, S1 );
figure,  imshow(C1);  title('密钥 x0 = 0.234, S = 1280');
[ C2, ~, ~ ] = cmbr_encryption( I, x1, S2 );
figure,  imshow(C2);  title('密钥 x0 = 0.234, S = 1280 + 1');
disp(['同一幅明文图像 fruit.png 在密钥 x0 为 0.234, 而 S 分别为 1280, 1281 下
', ...
    '加密得到的两幅密文图像的像素变化率 R 和归一化平均变化强度 U 分别为:']);
disp(['R = ', num2str( pixel_change_rate( C1, C2 ) )]);
disp(['U = ', num2str( normalized_mean_change_intensity( C1, C2 ) )]);

%%   单独针对解钥 x0
clc;  close all;
[ I1, ~ ] = cmbr_decryption( C1, x1, s1, S1 );
figure,  imshow(I1);  title('解钥 x0 = 0.234, S = 1280, s = 13122840');
[ I2, ~ ] = cmbr_decryption( C1, x2, s1, S1 );
figure,  imshow(I2);  title('解钥 x0 = 0.234 + 1e-
10, S = 1280, s = 13122840');
disp(['同一幅密文图像在解钥 S = 1280, s = 7290671, 而 x0 分别为
0.234, 0.234 + 1e-10', ...
    '下解密得到的两幅明文图像的像素变化率 R 和归一化平均变化强度 U 分别为:']);
disp(['R = ', num2str( pixel_change_rate( I1, I2 ) )]);
disp(['U = ', num2str( normalized_mean_change_intensity( I1, I2 ) )]);

%%   单独针对解钥 s
clc;  close all;
[ I1, ~ ] = cmbr_decryption( C1, x1, s1, S1 );
figure,  imshow(I1); title('解钥 s = 13122840, S = 1280, x0 = 0.234');
[ I2, ~ ] = cmbr_decryption( C1, x1, s2, S1 );
figure,  imshow(I2); title('解钥 s = 13122840 + 1, S = 1280, x0 = 0.234');
disp(['同一幅密文图像在解钥 x0 = 0.234, S = 1280, 而 s 分别为
7290671, 7290671 + 1', ...
    '下解密得到的两幅明文图像的像素变化率 R 和归一化平均变化强度 U 分别为:']);
disp(['R = ', num2str( pixel_change_rate( I1, I2 ) )]);
disp(['U = ', num2str( normalized_mean_change_intensity( I1, I2 ) )]);

%%   单独针对解钥 S
clc;  close all;
[ I1, ~ ] = cmbr_decryption( C1, x1, s1, S1 );
figure,  imshow(I1);  title('解钥 x0 = 0.234, S = 1280, s = 13122840');
```

```
144. [ I2, ~ ] = cmbr_decryption( C1, x1, s1, S2 );
145. figure,  imshow(I2);  title('解钥
    x0 = 0.234, S = 1280 + 1, s = 13122840');
146. disp(['同一幅密文图像在解钥 x0 = 0.234, s = 7290671, 而 S 分别为
    1280, 1280 + 1', ...
147.      '下解密得到的两幅明文图像的像素变化率 R 和归一化平均变化强度 U 分别为:']);
148. disp(['R = ', num2str( pixel_change_rate( I1, I2 ) )]);
149. disp(['U = ', num2str( normalized_mean_change_intensity( I1, I2 ) )]);
150.
151. %%  相邻像素的相关性分析
152. %   需要额外调用函数文件 r_xy_near.m
153.
154. clear;  clc;  close all;
155. I = imread('fruit.png');  I = I(:, 1 : 323);
156. x0 = 0.234;  S = 1280;  n = 20000;
157. % direction = 'Left';
158. % direction = 'Right';
159. % direction = 'Up';
160. % direction = 'Down';
161. % direction = 'Left_Up';
162. % direction = 'Left_Down';
163. % direction = 'Right_Up';
164. direction = 'Right_Down';
165.
166. [ r_xy_I, X_I, Y_I ] = r_xy_near( I, n, direction );
167. [ C, ~, ~ ] = cmbr_encryption( I, x0, S );
168. [ r_xy_C, X_C, Y_C ] = r_xy_near( C, n, direction );
169.
170. figure,  scatter( X_I, Y_I );  title(['明文图像', num2str(n), '对相邻像素的灰
    度值分布']);
171. xlabel('(x,y)的像素值');  ylabel('(x,y)的相邻像素值');
172. disp(['明文图像随机取', num2str(n), '对相邻像素计算得相关系
    数: ', num2str(r_xy_I)]);
173. figure,  scatter( X_C, Y_C );  title(['密文图像', num2str(n), '对相邻像素的灰
    度值分布']);
174. xlabel('(x,y)的像素值');  ylabel('(x,y)的相邻像素值');
175. disp(['密文图像随机取', num2str(n), '对相邻像素计算得相关系
    数: ', num2str(r_xy_C)]);
176.
177. %%  信息熵分析
178. %   对于一幅密文图像, 它的理想的信息熵的值是 8.
179. %   调用 Matlab 自带函数 entropy( )计算信息熵
180.
181. clear;  clc;  close all;
```

```
I = imread('fruit.png');  I = I(:, 1 : 323);
x0 = 0.234;  S = 1280;
[C, ~, ~] = cmbr_encryption( I, x0, S );
disp(['密文图像的信息熵为: ', num2str( entropy(C) )]);

%%  加解密时间随着尺寸的变化

clear;  clc;  close all;
x0 = 0.234;  S = 1280;
size_sequence = 50 : 50 : 1000;
time_en = [];  time_de = [];
for k = size_sequence
    I = floor( rand(k, k) * 255 );

    tic
    [ C, ~, s ] = cmbr_encryption( I, x0, S );
    toc
    time_en = [time_en, toc];

    tic
    [ ~, ~ ] = cmbr_decryption( C, x0, s, S );
    toc
    time_de = [time_de, toc];

    disp(['图像大小为: ', num2str(k), ' * ', num2str(k), '的加解密已完成
']);
end

figure,  plot(size_sequence, time_en, 'r');
title('加密算法用时与明文图像大小的关系曲线');
xlabel('明文图像大小');  ylabel('加密算法用时');

figure,  plot(size_sequence, time_de, 'b');
title('解密算法用时与密文图像大小的关系曲线');
xlabel('密文图像大小');  ylabel('解密算法用时');

%%  计算像素变化率 R 的函数, R 越接近 1 - 2 ^ (-8)(大约是 0.9961), 明文敏感性越
好
function R = pixel_change_rate( I1, I2 )
%   I1, I2 是待比较的两个图像矩阵, 要求尺寸相同
[H1, W1] = size(I1);  [H2, W2] = size(I2);
if H1 ~= H2 || W1 ~= W2
    error('输入的两个图像矩阵规格不统一');
end
```

```
224.
225. I1 = double(I1);  I2 = double(I2);
226. R = sum( sum( I1~=I2 ) ) / (H1 * W1);
227. end
228.
229. %%  计算归一化平均变化强度 U 的函数，U 的理想值为 0.3446
230. function U = normalized_mean_change_intensity( I1, I2 )
231. %   I1, I2 是待比较的两个图像矩阵，要求尺寸相同
232. [H1, W1] = size(I1);  [H2, W2] = size(I2);
233. if H1 ~= H2 || W1 ~= W2
234.     error('输入的两个图像矩阵规格不统一');
235. end
236.
237. I1 = double(I1);  I2 = double(I2);
238. U = sum( sum( abs(I1 - I2) ) ) / ( H1 * W1 * 255 );
239. end
```