数字水印算法模块的实现

辛正非　韦继程　俞聪　陈竹筠

（上海第二工业大学　上海　201209）

摘要：针对当前数字媒体信息受剽窃、盗用等侵权行为的不断发生，数字水印技术便是一项由此应运而生的新兴技术。为了将待加密文件隐藏入载体图像，并具备安全性、高容量、鲁棒性等特性，实现了最低有效位算法、随机间隔算法和区域校验位算法。本文概述了数字水印、数字水印的评价指标，简述了最低有效位算法、随机间隔算法和区域校验位算法的实现方式，并通过 Python 及其库 opencv2 等编写了该算法，并通过峰值信噪比（PSNR）和结构相似性（SSIM）对加密前后的载体图像进行评价，计算了算法加密的容量，最低有效位算法下，一张载体图像的容量约为 5 份加密文件，随机间隔算法和区域校验位算法下一张载体图像的容量为 1 份加密文件。该算法进行加密的途径为修改图片的像素位，故将图片格式修改为 PNG 依然可以正常加密，具备一定的鲁棒性。

关键字：数字水印；最低有效位算法；随机间隔算法；区域校验位算法；

0　引言

随着互联网和移动互联网技术的发展，使得信息的发布和传输实现了“数字化”和“网络化”，以数字图像为代表的数字信息的传递和存取渠道得到了扩展，同时也大大提高了数字信息表达的效率和准确性。但是，这些数字信息很容易借助互联网被复制、处理、传播和公开，各类数字媒体的传播达到了前所未有的深度和广度，引发出数字信息传输的安全问题和数字产品的版权保护问题。近些年关于网络传播内容的版权纠纷不断增加，侵犯作品作者的合法版权的案件越来越多。在互联网环境中如何有效保护版权，如何提高水印信息的安全性，已成为信息安全和信息隐藏领域的热门课题。

起初，信息安全专家通过利用利用私钥或公钥将明文信息加密为其他人无法识别的密文信息，解密密文需要对应的密钥。另一种数字签名技术是通过私钥对每个消息进行签名。但是随着数据的增加，签名数量也随之增加，因此也为数字产品的传输带来诸多不便。随着科学技术的发展，新的攻击技术也在不断涌现，传统的加密技术和数字签名技术已经不能满足数字信息安全的需要，急需新的技术对数字信息进行更加安全的保护。

随着研究的不断深入，数字水印技术作为一种新型信息隐藏技术问世。这种技术可以为解密后的数据提供进一步的保护，而且可以在原始载体中嵌入大量的水印信息。数字水印技术主要用于数字图像的版权保护和完整性认证，作为一种保护数字图像版权的主要技术，通过水印算法将版权信息以一种隐蔽的形式嵌入到宿主图像。在数字图像版权受到侵犯时，能够通过水印提取算法将版权信息提取出来，作为数字图像归属的主要证据。

数字水印技术在实际使用中将会面临一些攻击和破坏，为了防止载体图片上的秘密信息被破坏，所以数字水印算法必须具有安全性、鲁棒性等特性，需要难以让攻击者发现载体图片上藏匿了具体的信息。

1　概述

1.1　数字水印

数字水印（Digital Watermark），是将特定的数字信号嵌入数字产品中保护数字产品版权、完整性、防复制或去向追踪的技术。

数字水印的生成通常有两种，无实际意义的随机序列水印和有实际意义的水印。无实际意义的水印常常选用伪随机序列作为数字水印，不能表示具体的、特定的信息，其最大的特点是比较难伪造，可以保证水印的安全性。但实际无意义的水印在检测时只能判断出待检验的图像中“有”、“无”水印两种结论，在生活中使用起来有一定的局限性。有实际意义的水印水印可以是文字、指纹、图像、音频等等，它本身就有实际的含义，又不容易被伪造和篡改，可以提供比无意义水印更多的信息量，同时对水印的嵌入和检测也有较高的要求。

数字水印的嵌入是指按照某种特定的嵌入算法将生成的水印信息加载到载体（图像、音频、视频等）中。数字水印的提取则是指在某种场合的需要时能够将数字水印有效地分离与提取出来，以便证明该数字产品的版权归属方或者识别版权所有者的身份。

数字水印通常被要求具有如下特点：

1. 透明性

水印的透明性主要指数字图像嵌入水印信息后与嵌入之前在人眼视觉上没有发生明显变化，也就是宿主图像与含水印图像对观察者的视觉系统是不可察觉的。另外，透明性也表现为即使用统计方法也不能恢复出原始水印。这里需要指出的是，透明性是相对于被保护数字图像的使用而言，即加在图像上的水印不应干扰图像的视觉欣赏效果，但也并不是不可见。

2. 鲁棒性

鲁棒性指含水印图像在受到攻击后仍然可以从中提取出水印信息，仍能保持其完整性和认证真实性。常见的对图像的攻击有滤波、压缩、旋转、平移等。透明性和鲁棒性是数字图像水印的两个重要特性，水印既要具备隐蔽性，又要具有一定的鲁棒性。主观上讲，理想的水印算法既能隐藏大量的水印数据，又可以抵抗各种攻击。

3. 安全性

安全性表现为水印能够抵抗恶意攻击的能力。一方面，水印的嵌入算法和提取算法框架是秘密的，嵌入的数字水印信息是利用密钥经过加密算法处理过的。另一方面，嵌入的数字水印统计上是不可检测的，非授权用户无法检测和破坏水印。嵌入水印的数字图像遭受恶意攻击后水印信息仍会一直存在，直到图像严重失真，丧失使用价值。

1.2 评价方式

本文对数字水印算法采用的评价指标如下：

1. 峰值信噪比

峰值信噪比（Peak signal-to-noise ratio, PSNR）主要用于衡量嵌入水印后的图像与原始图像之间的失真程度，是使用最为广泛的一种图像质量评价指标，用来评估图像的保真性，通过 MSE 进行定义，可以反映对应像素点间的误差，是基于误差敏感的图像质量评价。

MSE 计算方式如下:

$$MSE = \frac{1}{M*N}\sum_{i=1}^{M}\sum_{j=1}^{N}(X(i,j)-Y(i,j))^2$$

PSNR 计算方式如下

$$PSNR = 10*log_{10}(\frac{MAX^2}{MSE}) = 20*log_{10}(\frac{MAX}{\sqrt{MSE}})$$

其中 MSE 表示当前图像 X 和参考图像 Y 的均方误差，$N*M$ 为图像的分辨率，MAX 是表示图像的最大像素值，比如每个像素点用 8 位表示，那么 MAX 为 255。PSNR 的单位为 dB，PSNR 值越大，意味着图像的失真越少。一般情况下，PSNR 值较高的图象质量相对较高，当 PSNR 值大于 25 dB 时，图象质量的差异并不明显；当 PSNR 值大于 35 或 40dB 时，则人眼分辨不出任何差异。

2. 结构相似性

结构相似性（Structural Similarity，SSIM），是一种衡量图像相似度的指标，分别从亮度、对比度、结构三方面度量相似性。给定图像 X 和 Y 结构相似性可按照以下方式求出：

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

其中，μ_x, μ_y 分别表示图像 X 和 Y 的均值，σ_x^2, σ_y^2 分别表示图像 X 和 Y 的方差，σ_{xy} 表示图像 X 和 Y 的协方差。c_1, c_2 为常数，通常取 $c_1 = (k_1 * L)^2, c_2 = (k_2 * L)^2, k_1 = 0.01, k_2 = 0.03$ L 是像素值的动态范围（对于灰度图像，L 为 255）。SSIM 的取值范围为 [-1,1]。当两张图像一样时，SSIM 的值等于 1。结构相似性参数分别用均值、标准差和协方差来估计图像的亮度、对比度以及结构相似程度。由于 SSIM 的出色表现，SSIM 已经在超分辨率、图像去噪中都有广泛的应用。

3. 汉明距离

汉明距离能够反映序列间的相似程度，进而提供序列相似程度的客观依据。设长度为 N 的二值序列 V 和 U，$V = \{v_0, v_1, ..., v_{N-1}\}$，$U = \{u_0, u_1, ..., u_{N-1}\}$，$d(V,U) = \sum_{i=0}^{N-1}(v_i \bigoplus u_i)$。则 d(V,U) 称为两个序列 V 和 U 的汉明距离，它表示 V 和 U 相应元素不等的个数。设 V 中两个元素 v_i 和 v_j 在序列中的位置分别为 i,j，0 i < j < N，其位置间距记为 k，k=j-i。对所有位置间距为 k 的两个元素的异或值求和，结果称之为位置间距 k 的汉明距离，记为 $D_k, D_k = \sum_{i=0}^{N-k-1} v_i \bigoplus v_{j+k}$。对所有 k（1 k < N）计算其 D_k，称为对序列进行汉明距离统计。

4. 容量

使用相同的载体图片和相同的秘密文件对算法进行测试，计算加密相同数量秘密文件所需要的载体图片数量

5. 健壮性

对图片进行攻击，如更改图片格式、对图像进行缩放，再进行解密，查看是否依然可以解密成功。

2 算法实现

2.1 最低有效位算法

最低有效位（Least Significant Bits，LSB）算法，最低有效位又称最不显著位，是指数字图像像素值用二进制表示时的最低位。它利用载体对象的二进制的最低一位（或几位）来进行秘密信息的隐藏。水印信息和像素值都是二值比特序列，如果用 8 位比特的二进制

表示灰度图像的每一个像素值，改变最低位的值不会对视觉效果产生明显影响。

LSB 算法嵌入基本步骤：

1、将要隐藏的水印文本信息的转换为二进制数据；

2、读入载体图像，将图像矩阵的每个像素转换成二进制。用水印信息的二进制数据的每一比特位替换与之相对应的载体图像的最低有效位（LSB 位）；

3、将载体图像的每一像素再由二进制转换为十进制，从而获得含有水印信息的图像。

LSB 算法提取基本步骤：

1、将嵌入水印信息的载体图像的像素转换成二进制，依次取出每一个像素的二进制的最后一位，拼接成二进制数据。

2、将得到的二进制数据转换成文本数据，便可得到隐藏的信息。

这种提取水印的方法属于盲检测，即不需要原始载体图像的数据便可以提取出全部水印信息。正是因为提取算法简单，这种水印很容易被恶意提取出来并修改为其他信息。为了提高水印的安全性，在嵌入之前先对水印信息进行加密或者置乱，这样在没有密钥的前提下就无法准确获取到水印信息。

2.2 随机间隔算法

随机间隔算法是对 LSB 算法的改进，从 LSB 水印算法的嵌入过程来看，由于嵌入的位置是最低有效位，故对噪声的抵抗力较差，并且嵌入位置固定，容易遭受攻击。针对以上不足，研究者提出了随机间隔算法，这是一种改进的 LSB 算法，发送者和接受者使用同一个密钥作为随机数生成器的种子，生成随机序列 k1,k2,⋯,kl(m)，并进一步以此生成隐藏位置的指标集。改进后的 LSB 算法在安全性和鲁棒性等方面都得到了提高。

随机间隔法嵌入水印的过程与 LSB 基本算法嵌入水印的过程基本相似，随机间隔算法只是不再按顺序依次在载体图像的每一位像素中嵌入了，而是根据随机序列和步长来跳跃一段间隔。随机间隔算法提取水印是随机间隔法嵌入水印的逆操作，提取信息时使用与嵌入时相同的随机数序列，即可跳跃到相同的位置提取出隐藏信息。

因为有随机间隔步长的存在，随机间隔算法隐藏容量比基本 LSB 算法小，存储信息的容量 $Capactiy = Width * Height/Step$, 其中 Width 和 Height 是载体图片的长和宽，Width*Height 是载体图片的大小。Step 为随机间隔法的步长。由此可见，在给定载体图片的情况下，随机间隔算法的步长与存储信息容量成反比，存储信息的容量也与随机间隔算法的步长成反比，二者双向影响。

2.3 区域校验位算法

区域校验位算法也是一种 LSB 算法的改进。利用区域校验位算法隐藏信息，首先把载体分成几个不相交的区域，再利用校验位在每一个区域中隐藏 1 比特信息。

嵌入过程:

根据设定的区域大小在载体图像上划分区域，根据要嵌入的载体信息 m_i，计算每一个区域的奇偶校验位 $p(I_i)$。一个载体区域 I_i 的奇偶校验位定义为:

$$p(I_i) = \sum_{j \in I} LSB(C_j)\%2$$

若 $p(I_i)$ 与　一致，则 $p(I_i)$ 已经隐藏了 m_i；若 $p(I_i)$ 与 m_i 不一致，则改变 I_i 中任意一个元素的 LSB，使得 $p(I_i) = m_i$。

提取过程:

提取过程是嵌入过程的逆过程，根据嵌入时的区域大小在划分区域，依次在每块区域上计算其奇偶校验位 $p(I_i)$ 即为本区域隐藏的信息。然后将每一块区域提取出的信息拼接在一起，即可得到隐藏的水印信息。

区域校验位算法具有如下优点：

1、载体区域中最多只需更改 1 个元素的 1 个比特位，这样秘密消息的嵌入对载体的统计特性改变最小。抵抗被动攻击的安全性较高。

2、与基本 LSB 算法相比，一个区域中只修改一个元素，水印嵌入对载体图片的统计特性影响更小，更不易被察觉。

但是，区域校验位算法也具有如下缺陷：

1、隐藏容量比基本 LSB 技术小。若区域大小为 N，则容量恰好缩小了 N 倍。

2、类似于 LSB 技术，该技术抵抗修改的能力依然很脆弱。

在给定载体图片的情况下，区域校验位算法的区域大小与存储信息容量成反比，存储信息的容量也与区域校验位算法的区域大小成反比，二者双向影响。

3 结果分析与对比

通过对上述三种算法的实现（见附录）与运行，得到了三种算法得 PSNR、SSIM、解密前后文件得汉明距离三个测试指标，其具体值如下表：

算法	PSNR	SSIM	汉明距离
LSB	51.3359	0.99514	0
随机间隔算法	51.4171	0.99524	0
区域校验位算法	51.3723	0.99517	0

由以上 PSNR 和 SSIM 值可知，加密后的载体图像相比于原来的图像，其差距非常小，人的肉眼将难以分辨加密前后载体图像的区别。计算加密前后的文件尺寸，加密后的文件尺寸增长小于 5%。

对于 LSB 算法，加密一份文件仅需要约 0.2 张载体图片，随机间隔算法和区域校验位算法需要 1 张图片。由于三种算法的原理均为修改图片像素值，故将格式更改为 PNG 格式后，依然可以正常解密。

4 基于 RPC 提供加解密服务

远程过程调用 (Remote Procedure Call，RPC) 是一种计算机通信协议。允许运行在一台计算机的程序调用另一个地址空间的子程序（一般是开放网络中的一台计算机），而程序员就像调用调用本地程序一样，无需额外做交互编程。

RPC 协议本质上定义了一种通信的流程，而具体的实现技术是没有约束的，每一种 RPC 框架都有自己的实现方式，比如你可以规定自己的 RPC 请求/响应包含消息头和消息体，使用 gob/json/pb/thrift 来序列化/反序列化消息内容，使用 socket/http2 进行网络通信，只要 client 和 server 消息的发送和解析能对应即可。

为了将实现的算法封装为模块化的组件，并提供接口调用服务，对算法做了如下基于 RPC 协议的封装：

1. IDL 定义通信的结构体，RPC 请求体包含需要加密的二进制数据（编码后的需要加密的 python 文件）
2. RPC 的服务端接收到需加密的文件后，将其存入临时目录
3. 调用算法生成图片，将 python 文件内容存入图像中
4. RPC 将图片数据以二进制形式响应给 rpc 客户端，由客户端反序列化为图片数据

5 总结

本文根据最低有效位算法、随机间隔算法和区域校验位算法的原理进行代码实现，完成了通过各算法进行加密和解密的工作，并通过峰值信噪比（PSNR）和结构相似性（SSIM）对加密前后的载体图片进行评价，结论是加密前后图像的差距极小，不会被人肉眼发现。对解密前后的加密文件的二进制位计算了汉明距离，其值均为 0，说明解密完全正确。最低有效位算法下，一张载体图像的容量约为 5 份加密文件，随机间隔算法和区域校验位算法下一张载体图像的容量为 1 份加密文件。该算法进行加密的途径为修改图片的像素位，故将图片格式修改为 PNG 依然可以正常加密，具备一定的鲁棒性。

A 策略模式下算法调用类

```
class Handler:
    def __init__(self, carrier_path, secret_path, result_path, method: Method=None) -> None:
        self._method = method

        self._img, self._shape = read_image.read_img_by_cv2(carrier_path)
        self._secret = read_secret.read_secret_bin(secret_path)
        self._length = len(self._secret)

        self._encrypted_img = None

        self._carrier_path = carrier_path
        self._secret_path = secret_path
        self._result_path = result_path

    @property
    def method(self):
        return self._method
```

```
@method.setter
def method(self, method: Method):
    self._method = method

def PSNR(self):
    primary = np.array(
        self._img
    )

    encrypted = np.array(
        self._encrypted_img
    )

    MSE = np.mean(
        (primary - encrypted) ** 2
    )

    if MSE == 0:
        psnr = float('inf')
    else:
        psnr = 20 * np.log10(
            255 / np.sqrt(MSE)
        )

    # print("PSNR: ", psnr)

    # if psnr >= 50:
    #     print("加密前后图像误差极小")
    # elif psnr >= 30:
    #     print("难以察觉加密前后图像的误差")
    # elif psnr >= 20:
    #     print("人眼可以察觉加密前后图像误差")
    # elif psnr >= 10:
    #     print("存在明显的图像差异，但仍可判断为可能是同一张图片")
    # else:
    #     print("压根就不是同一张图")

    return psnr

def SSIM(self):
    ssim = structural_similarity(self._img, self._encrypted_img)

    # print("SSIM: ", ssim)

    return ssim

def Hamming(self, decrypted):
    res = hamming_loss(np.array(
        [
            int(x) for x in self._secret
        ]
    ), np.array(
        [
```

```
                int(x) for x in decrypted
            ]
        ))
        return res

    def run(self, **args):
        interval_step = args["interval_step"] if "interval_step" in args else 0
        check_size =  args["check_size"] if "check_size" in args else 0
        encrypted_img = self._method().encrypt(
            carrier_path=self._carrier_path,
            secret_path=self._secret_path,
            interval_step=interval_step,
            check_size=check_size
        )
        cv2.imwrite(self._result_path, encrypted_img, [cv2.IMWRITE_JPEG_QUALITY, 89])

        self._encrypted_img, shape = read_image.read_img_by_cv2(self._result_path)

        # if self._shape != shape:
        #     print("图片尺寸不一致")

        psnr = self.PSNR()
        ssim = self.SSIM()

        bits = self._method().decrypt(
            self._length,
            encrypted_img,
            "./decrypted.py",
            interval_step=interval_step,
            check_size=check_size
        )
        hamming = self.Hamming(bits)

        if hamming == 0:
            # print("解密结果完全一致")
            pass
        else:
            # print(hamming)
            pass

        return psnr, ssim, hamming
```

B 策略模式下算法父类

```
class Method(ABC):
    @abstractmethod
    def encrypt(self, carrier_path, secret_path, **args):
        pass

    @abstractmethod
    def decrypt(self, length, img, result_path, **args) -> str:
```

```
        pass

    def back_process(self, path, bits) -> str:
        with open(path, "wb") as f:
            for i in range(0, len(bits), 8):
                ascii_code = int(bits[i:i+8], 2)
                ascii_code = chr(ascii_code)

                b = bytes(ascii_code, encoding="utf-8")
                f.write(b)
        return bits
```

C LSB 算法

```
class LSB_Method(Method):

    def encrypt(self, carrier_path, secret_path, **args):
        img, shape = read_image.read_img_by_cv2(carrier_path)
        info = read_secret.read_secret_bin(secret_path)

        cnt = 0
        l = len(info)

        for h in range(0, shape[1]):
            for w in range(0, shape[0]):
                pix = img[w, h]

                if cnt >= l:
                    break

                pix = pix - pix % 2 + int(info[cnt])
                cnt += 1

                img[w, h] = pix

        return img

    def decrypt(self, length, img, result_path, **args) -> str:
        shape = img.shape

        cnt = 0
        bits = ""
        for h in range(0, shape[1]):
            for w in range(0, shape[0]):

                pix = img[w, h]

                bits += str(pix % 2)
                cnt += 1

                if cnt == length:
```

```
                break

            if cnt == length:
                break

        return self.back_process(result_path, bits)
```

D 随机间隔算法

```
class LSB_RandomInterval(Method):

    def encrypt(self, carrier_path, secret_path, **args):
        img, shape = read_image.read_img_by_cv2(carrier_path)
        info = read_secret.read_secret_bin(secret_path)
        random.seed(2)

        cnt = 0
        l = len(info)
        step_max = int(shape[0] * shape[1] / l)

        if args["interval_step"] > step_max:
            raise Exception("步长设置过大")

        random_seq = [0] * l
        for i in range(0, l):
            random_seq[i] = int(random.random() * args["interval_step"] + 1)

        q = 1
        for cnt in range(l):
            w, h = convert_1d_2_2d.convert(q, shape[0])
            pix = img[w, h]
            pix = pix - pix % 2 + int(info[cnt])
            q = q + random_seq[cnt]
            img[w, h] = pix

        return img

    def decrypt(self, length, img, result_path, **args) -> str:
        random.seed(2)
        shape = img.shape

        cnt = 0
        bits = ""

        random_seq = [0] * length
        for i in range(0, length):
            random_seq[i] = int(random.random() * args["interval_step"] + 1)

        q = 1
        for cnt in range(length):
            w, h = convert_1d_2_2d.convert(q, img.shape[0])
```

```
            pix = img[w, h]
            bits += str(pix % 2)
            q += random_seq[cnt]

        return self.back_process(result_path, bits)
```

E 区域校验位算法

```
class LSB_RegionalCheck(Method):

    def encrypt(self, carrier_path, secret_path, **args):
        img, shape = read_image.read_img_by_cv2(carrier_path)
        info = read_secret.read_secret_bin(secret_path)
        # random.seed(2)

        check_size = args["check_size"]

        cnt = 0
        l = len(info)

        q = 1
        size_max = int(shape[0] * shape[1] / l)
        if shape[0] * shape[1] < check_size * l:
            raise Exception("Check size 设置过大")

        pixels = list()
        for p in range(1, l + 1):
            for i in range(1, check_size + 1):
                w, h = convert_1d_2_2d.convert((p - 1) * check_size + i, shape[0])
                pixels.append(img[w, h])

            temp = 0
            for i, v in enumerate(pixels):
                temp += v % 2
            pixels = list()
            temp %= 2

            if temp != int(info[p - 1]):
                q = int(random.random() * check_size) + 1
                w, h = convert_1d_2_2d.convert((p - 1) * check_size + q, shape[0])
                pix = img[w, h]
                img[w, h] = pix - 1

        return img

    def decrypt(self, length, img, result_path, **args) -> str:
        check_size = args["check_size"]
        random.seed(2)
        shape = img.shape

        cnt = 0
```

```
bits = ""

pixels = list()
for p in range(1, length + 1):
    for i in range(1, check_size + 1):
        w, h = convert_1d_2_2d.convert((p - 1) * check_size + i, img.shape[0])
        pixels.append(img[w, h])

    temp = 0
    for i, v in enumerate(pixels):
        temp += v % 2
    pixels = list()
    temp %= 2
    bits += str(temp)

return self.back_process(result_path, bits)
```