IMAGE INTEGRITY VERIFICATION USING CONTENT-BASED WATERMARKING

By

CHENG-TING YEH

Bachelor of Science

Iowa State University of Science and Technology

Ames, Iowa

2002

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the Degree of
MASTER OF SCIENCE
May 2005

IMAGE INTEGRITY VERIFICATION USING

CONTENT-BASED WATERMARKING

Thesis Approved:

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. H. K. Dai, for his intelligent supervision, constructive guidance, inspiration and friendship, and most important of all, for giving me the opportunity to work on such interesting topic. Without his help, this work would not be possible. I would also like to extend my appreciation to my other committee members Dr. John P. Chandler and Dr. Douglas R. Heisterkamp, whose guidance, assistance, encouragement, and friendship are also invaluable.

I would also like to give my genuine appreciation to my family members for their love, especially to my parents who provided me strong encouragement and support.

Finally, I would like to thank Computer Science Department at Oklahoma State University for supporting me during my graduate study.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF SYMBOLS

A	Bi-level watermark image
b	Block size of the partitioned non-overlapping blocks in pixels
$D(\cdot)$	Decryption function
$E(\cdot)$	Encryption function
E_K	Encoded 64-bit key
$H(\cdot)$	Cryptographic one-way hash function
I	Cover image to be used to embed the watermark image
I^W	Watermarked image
LSB	Least significant bit
M	Width in pixels of the cover image or watermarked image
N	Height in pixels of the cover image or watermarked image
W	Scaled or periodically replicated watermark image A whose dimensions equal to the dimensions of I
X	Non-overlapping block of pixels
X°	Non-overlapping block of pixels where each element in X° equals to X except that the LSBs of all the pixels are set to zero.
$\oplus$	Bitwise exclusive-or operator

- Assignment operator to store the encoded 64-bit key E_K in the least significant bits of the block $\tilde{X}$

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Over the past years, the Internet has gained great popularity and has become a commonplace to make collection of images stored in Internet-attached servers accessible to vast number of users. However, making those images public also creates opportunities for malicious parties to intercept and modify the images transmitted to others, or even replace the authentic ones with their respective forgeries. In addition, with the sophisticated image processing tools commercially available today, many have the capabilities to modify the content of the digital media with ease, giving more opportunities for malicious parties to produce results that mimic the work of professionals where alterations are undetectable by the human eye. Intellectual property protection is one of the most pressing concerns for content creators and owners who distribute and deliver their contents in the new digital world. For this reason, in order to reduce the constant proliferation and threats for digital media in the World Wide Web, digital watermarking is one of the crucial techniques to protect the contents of products in digital form and enable copyright or ownership information to be embedded into the multimedia data. Furthermore, digital watermarking technology is viewed as an enabling agent allowing more widespread sharing and use of that content while decreasing worry over piracy [CYY98].

1.2 DIGITAL WATERMARKING

The technology of digital watermarking is just beginning to mature. But the notion of embedding copyright or ownership information in multimedia data already has great potential. Digital watermarking is the hiding of data within the content's digital representation, it is a technique to insert an information-carrying digital signature into intellectual property in digital format so that the signature can later be extracted to identify the source, creator, owner, distributor, or authorized consumer of the document or image [BO96]. Although digital watermarks are digital signatures in some way, they differ on applications and usage. Digital watermarks, similar to digital signatures, are encrypted electronic signatures obtained from the content data to authenticate the identity of the sender of a message, or of the signer of the document. When the authentication mark of a digital signature is intentionally modified, the decryption of such signature will yield content data that are completely different from the original data. On the other hand, unlike digital signatures, the signatures of digital watermarks enable localization properties where the exact location of possible alterations of the content data could be determined. In addition, the objective of digital watermarks is to permanently and unalterably mark the image so that the credit or assignment is beyond dispute. In the event of illicit use, the watermark would facilitate the claim of ownership, the receipt of copyright revenues, or successful prosecution. Moreover, watermarking technology has also been proposed for tracing images that have illicitly redistributed. In the past, the infeasibility of large-scale photocopying and distribution often limited copyright infringement, but modern digital networks make large-scale broadcasting simple and inexpensive. For this reason, digital watermarking also allows each image to be uniquely marked for every buyer and if the buyer makes an illicit copy, the copy itself identifies the buyer as the source.

1.3 WATERMARK CLASSIFICATION

Of the classification schemes that apply to watermarks, the distinction between visible and invisible (perceptible and imperceptible) seems to be the most fundamental. A visible or perceptible watermark typically consists of a conspicuously visible message or a logo embedded within an image indicating the ownership of the image. On the other hand, an invisible or imperceptible watermark when embedded into an image appears visually very similar to the original image. In other words, the existence of an invisible watermark can only be determined using an appropriate watermark extraction or detection algorithm. Within such classification of watermarks, an invisible watermark can further be classified as robust and fragile. A robust watermark is designed to survive against malicious attacks [WD96, CKLS97], because the watermark can still be extracted even if the watermarked image has been processed by common image processing techniques such as scaling, cropping, and compression. Fragile watermarks [YM97, Won98, LLC00, and WM01], on the other hand, are designed to detect major or minor changes to the source image as well as localizing the areas that has been tampered. The usage of robust and fragile watermarks depends on the type of application as proposed in

[MBY97, MW98]. Among the many of them, robust watermarks can be found in copy protection applications such as digital video discs (DVD), fingerprinting for recipient tracing, and content ownership verification. Meanwhile, fragile watermarks can be used in news broadcasting, medical, forensic, and military applications where the content verification and identity authentication become crucial in order to detect forgeries and impersonations.

Depending on the way the watermark is inserted and the nature of the watermarking algorithm, the detection and extraction process can take on very distinct approaches [HM00]. One major differentiating characteristic between watermark techniques is the obliviousness of the algorithm. A watermark scheme is considered oblivious when it does not require the contents of the original image (cover image) during the extraction or detection step. Schemes that do require the presence of the original image during the verification step are considered non-oblivious.

1.4 WATERMARKING APPLICATIONS

There are many approaches available to protect digital data; these include encryption, authentication, and time stamping. Watermarking methods are often evaluated based on their common properties and robustness, tamper resistant, and fidelity [CMB00]. However, the requirements that a watermarking system has to comply with are always based on the application. Thus, it is inappropriate to compare two watermarks according to the same standards because its robustness can be identified depending on the specific application-driven requirements.

1.4.1 WATERMARKING FOR COPYRIGHT PROTECTION

Copyright protection is probably the most prominent application of watermarking today. The objective is to embed information about the source and copyright data in order to prevent other parties from claiming exclusive rights of the digital document. Although watermarks are used to resolve rightful ownership, this application also requires a level of robustness in the watermarking scheme. The driving force of this application is the Internet that contains millions of freely available images that the rightful owners want to protect. Additional watermark design requirements besides mere robustness apply, for instance, the watermark must be unambiguous and still resolve rightful ownership if other parties embed additional watermarks.

1.4.2 FINGERPRINTING FOR TRAITOR TRACKING

Monitoring and owner-identification applications place the same watermark in all copies of the same content. However, electronic distribution of content allows each copy distributed to be customized for each recipient. This capability permits a unique watermark to be embedded in each individual copy to avoid mass circulation. Transactional watermarks, also called fingerprints, allow a content owner or content distributor to identify the source of an illegal copy. This is potentially valuable both as a prevention for illegal reproduction and as a technological aid to investigation.

One possible application of fingerprinting watermarks is in the distribution of movie dailies [CMB00]. During the course of making a movie, the result of each day's photography is often distributed to a number of people involved in its production. These dailies are highly confidential, yet occasionally, a daily is leaked to the press. When this happens, studios quickly try to identify the source of the leak. Clearly, if each copy of the daily contains a unique transactional watermark that identifies the recipient, then identification of the source of the leak is much easier.

1.4.3 WATERMARKING FOR COPY PROTECTION

A desirable feature in multimedia distribution is the existence of a copy protection mechanism that disallows unauthorized copying of the digital media. Fingerprinting watermarks as well as watermarks for monitoring, identification, and proof of ownership do not prevent illegal copying. Rather, they serve as powerful restriction tool. However it is also possible for recording and playback devices to react to embedded signals. In this way, a recording device might inhibit recoding of a signal if it detects a watermark that indicates recoding is prohibited. Of course, for such a system to work, all manufactured recorders must include watermark detection circuitry. Such systems are currently being developed for DVD video and for digital music distribution.

1.4.4 WATERMARKING FOR IMAGE AUTHENTICATION

In authentication applications, the objective is to detect modifications of the data. This can be achieved with so called “fragile watermarks” that have a low robustness to certain modifications like compression, but are impaired by other modifications. Furthermore, the robustness requirements may change depending on the data type and application. Nevertheless, among all possible watermarking applications, authentication watermarking requires the lowest level of robustness by definition because their main purpose is to check if the content of the digital document has been modified or tampered and to localize the tampered locations.

1.5 BASIC WATERMARKING PRINCIPLES

All watermarking methods share the same generic building blocks: a watermark embedding system and a watermark recovery system (also called watermark extraction or watermark decoder) [KP00]. Figure 1.1 shows the generic watermark embedding process. The input to the scheme is the watermark, cover-data, and an optional public or secret key and the output of the watermarking scheme is the watermarked data. The key may be used to enforce security and prevent unauthorized parties from recovering and manipulating the watermark. The use of one key or combination of keys in the watermarking is a technique referred as secret and public key watermarking.

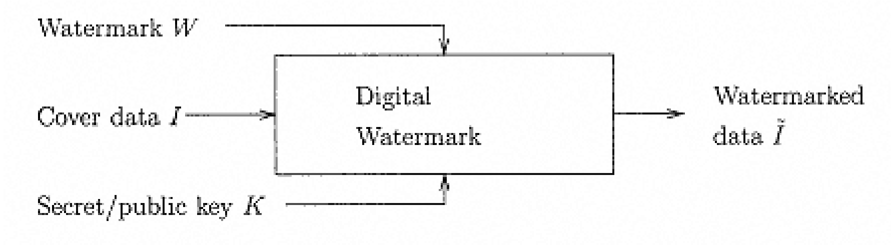


Figure 1.1: Generic watermark insertion scheme.

There are a few very general properties shared by all proposed watermarking systems. There are:

- **Imperceptibility:** The modifications caused by a watermarking embedding scheme should be below the perceptible threshold, which means that some sort of perceptibility criterion should be used not only to design the watermark, but also quantify the distortion. As a consequence of the required imperceptibility, the individual samples (or pixels, voxels, features, etc.) that are used for watermark embedding are only modified by a small amount.

- **Redundancy:** To ensure robustness despite of the imperceptibility threshold, the watermark information is redundantly distributed over many samples (or pixels, voxels, features, etc.) of the cover-data, thus providing a global robustness which means that the watermark can usually be recovered from a small fraction of the watermarked data.
- **Keys:** In general, watermarking schemes use one or more cryptographic keys to ensure security against manipulations and removal of the watermark. As soon as the watermark can be read by someone, the same person may easily destroy it because not only the embedding strategy, but also the locations of the watermark are known in this case.

The generic watermarking recovery process is shown in Figure 1.2. Inputs to the scheme are the watermarked data, the secret or public keys, and, depending on the method, the original data and/or original watermark. The output is either the recovered watermark or some kind of confidence measure indicating how likely it is for the given watermark at the input to be present in the data under inspection.

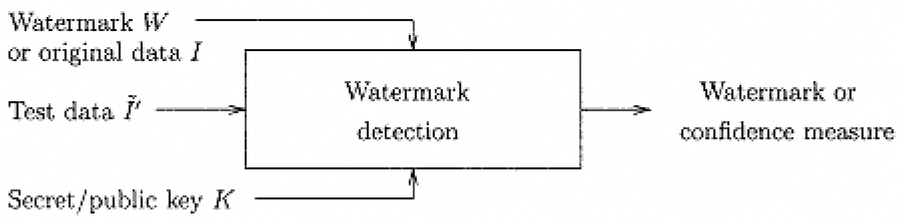


Figure 1.2: Generic watermark extraction scheme.

Three types of watermarking systems can be identified. Their difference is in the nature and combination of inputs and outputs:

- **Private Watermarking:** Also called nonblind watermarking and requires at least the original data. Private watermarking systems extract the watermark from the possibly distorted data and use the information from the original data as a hint to determine where the watermark was originally inserted. In addition, private watermarking systems also require a copy of the embedded watermark to yield an answer of “yes” if a watermark was embedded in the cover data and an answer of “no” otherwise.
- **Semiprivate Watermarking:** Also known as semiblind watermarking and does not use the original data for detection but yields the same output (i.e. “yes” or “no”) depending whether the cover data contains an embedded watermark or not. Potential applications of private and semiprivate watermarking are for evidence in court to prove ownership, copy control in applications such as DVDs where the disc reader needs to know whether it is allowed to play the content or not, and fingerprinting where the goal is to identify the original recipient of pirated copies.
- **Public Watermarking:** Also referred as blind or oblivious watermarking and remains as the most challenging problem since it requires neither the original data nor the embedded watermark. A potential application for public watermarking is for image integrity by protecting the contents of the image from addition and removal of objects.

1.6 RESEARCH SCOPE

In recent years, research efforts on digital watermarking have emerged in such fields as signal processing, computer science, and cryptography; furthermore, an increasing number of proposed watermarking schemes target a wide spectrum of applications. This research will focus on the design and implementation of watermarking algorithms for image authentication. In addition, we will study the different attacks against fragile watermarks and describe a new oblivious, more secure, and efficient fragile watermarking scheme for grayscale or color still-images with capabilities of detecting geometric transformations, removal of objects, addition of foreign objects, and tamper localization without any a priori knowledge of the embedded watermark image.

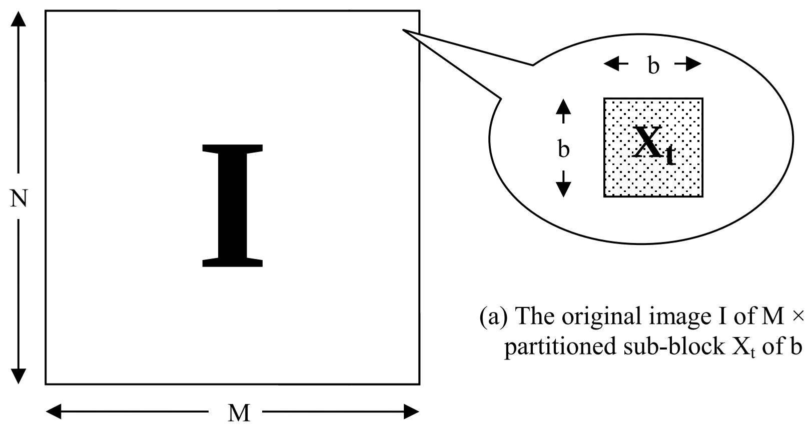
CHAPTER 2

RELEVANT WORK

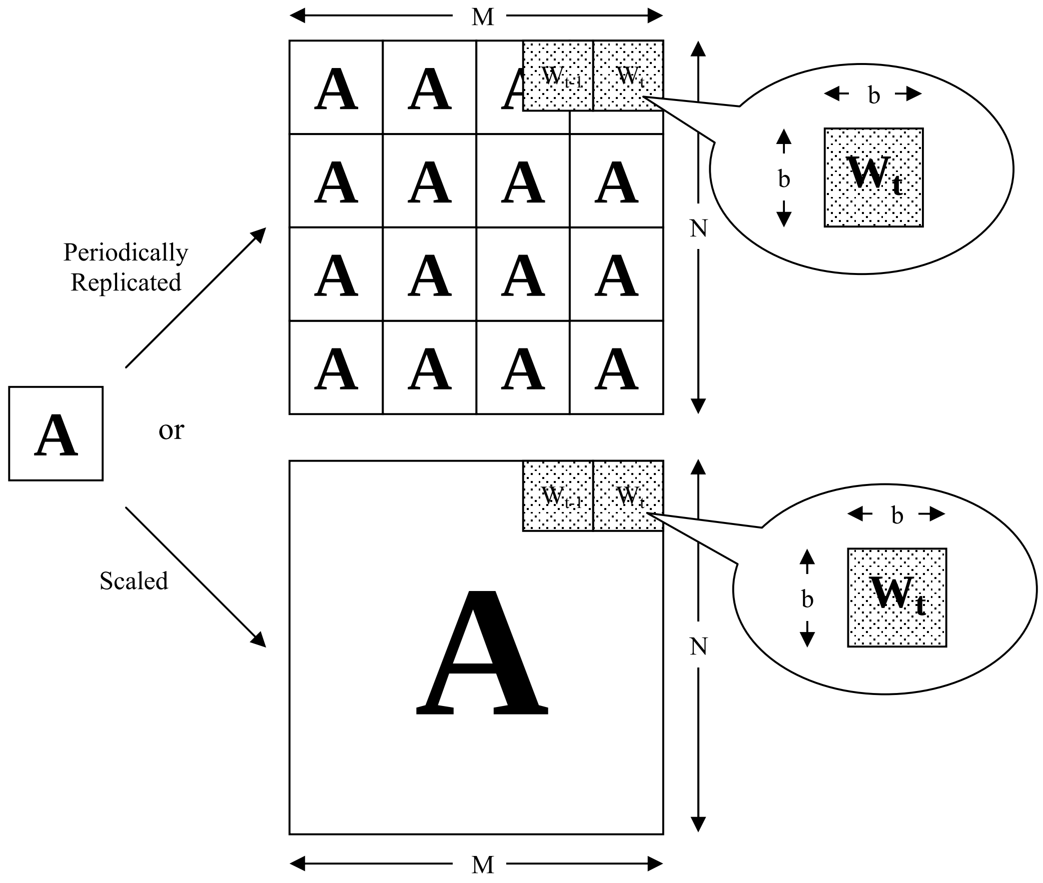
An oblivious watermark scheme is desirable because it lacks the requirement of transmitting the original image from the sender to the receiver at the time of the extraction process. Another important aspect of oblivious schemes enables the insertion and detection of the watermark in a block-to-block basis due to the fact that the watermark is embedded in different blocks of the original image. A block-based approach can be convenient in terms of simplicity and lack of computational overhead. However, watermark insertion in a block-to-block basis is still vulnerable to common watermarking attacks if they do not take content dependency into consideration. Many fragile watermarking schemes have been proposed in recent years [YM97, Won98, WM01, and LLC00]. Among them, Wong has proposed a blockwise fragile authentication watermarking [Won98] and has improved it by using public-key based scheme [WM01]. This chapter will describe the block-based watermarking scheme proposed in [WM01] in which it will serve as the basic structure for our improved watermarking scheme. In addition, a detailed description of common attacks against watermarking will be discussed and finally, we will show the weaknesses of the scheme proposed in [Won98, WM01].

For simplicity, Figure 2.1 illustrates the notation that will be used throughout this thesis to describe the watermarking algorithms. Let M and N be the width and height respectively of an image (cover image or watermarked image) I that will be uniformly partitioned into a sequence of non-overlapping blocks of $b \times b$ pixels in size. Define X to be a non-overlapping block in the sequence and X° to be the block of pixels where each element in X° equals the corresponding element in X except that the least significant bits (LSBs) of all the pixels are set to zero.

Also, let A be a bi-level image that will be used as the invisible watermark to be embedded into the cover image. Note that the watermark image to be used in the embedding step needs to be of the same size as the image I (i.e., $M \times N$ pixels); for this reason, if the dimensions of the watermark A and cover image I are not equal, the watermark A is scaled or periodically replicated forming another embedding watermark image W where the width and height of W equals the width and height of I respectively. In the similar fashion, this newly formed watermark image W is also uniformly partitioned into a sequence of non-overlapping blocks following the same partitioning scheme as performed for the image I.



(a) The original image I of M × N pixels and partitioned sub-block X_t of b × b pixels.



(b) Bi-Level image A periodically replicated or scaled to match the dimensions of image I, then partitioned into sub-blocks of b × b pixels.

Figure 2.1: Notation Diagram of image I and watermark image W

Finally, let $H(\cdot)$ be a cryptographic one-way hash function such as the MD5 [Riv92] and also define $E(\cdot)$ and $D(\cdot)$ as the encryption and decryption functions of a public-key cryptosystem such as the RSA [RSA78]. Lastly, define $\oplus$ to be element-wise exclusive-or operator between two non-overlapping blocks, and $\circ$ be the assignment operator, where $X^{\circ} \circ E_K$ denotes storing the encoded 64-bit key E_K in the LSBs of the block X°.

2.1 PUBLIC-KEY WATERMARKING

The authors of [Won98, WM01] describe a fragile watermarking technique that embeds an invisible watermark into a cover image. The watermark image is embedded in the LSBs of each pixel aided by a public-key cryptosystem and one-way hash function allowing the detection of any changes made in the watermarked image feasible. In addition, their scheme can also be used for ownership verification because it uses a secret key K that is only known by the owner at the time of insertion to embed the watermark image. Figure 2.2(a) shows an original image and Figure 2.2(b) illustrates an image with an invisible watermark added using the technique described in [WM01]. Their watermarking scheme exhibits the following properties:

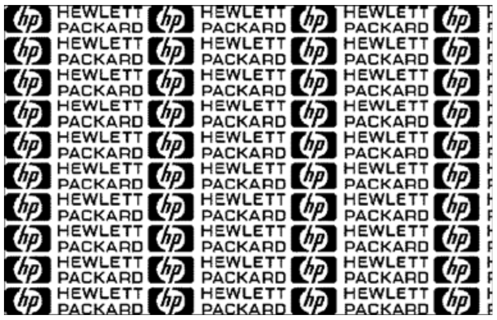
- During the watermark extraction step, if the correct key K is applied to the watermark extraction procedure to Figure 2.2(b), a proper watermark image is obtained indicating the authenticity of the image as seen in Figure 2.2(c).
- If an image is unmarked (i.e., if it does not contain a watermark), cropped, resized, or if an incorrect key is applied during the extraction step, the

watermark extraction process recovers an image that resembles random noise as seen in Figure 2.2(d).

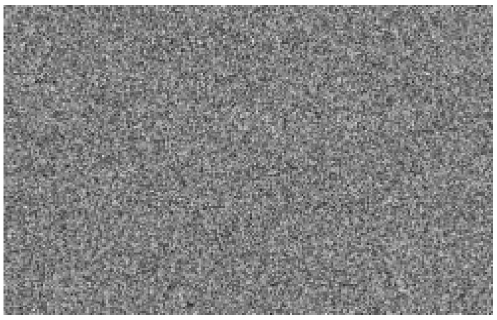
- If one changes certain pixels in the watermarked image, then the specific locations of the changes are reflected at the output of the watermarking extraction procedure. Figure 2.2(e) illustrates the addition of foreign objects to the image in Figure 2.2(b). Figure 2.2(f) shows the extracted watermark from Figure 2.2(e), indicating the specific area where changes have been made.

(a) Original image.

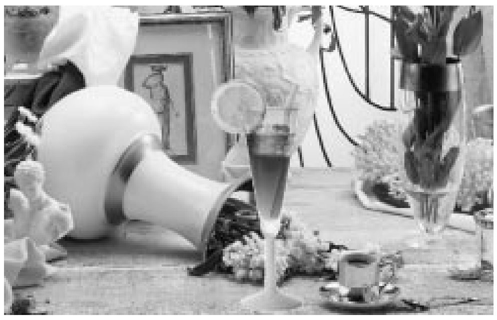
(b) An invisible watermark added to the original image in (a) without introducing visual artifacts and preserving the original image quality.



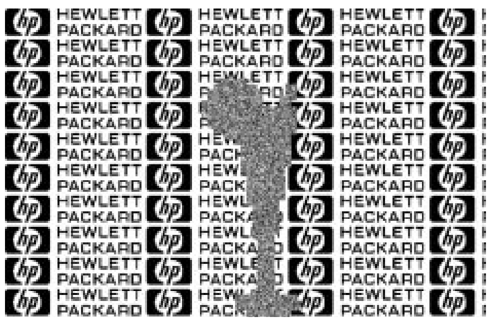
(c) Extracted watermark from the image in (b) using the proper verification key.



(d) Extracted watermark from the image in (b) using an incorrect key. Similar output will result if the image contains no watermark, or if the watermarked image is cropped or resized.



(e) The watermarked image in (b) has been changed. A glass was pasted on top.



(f) Extracted watermark from (e) indicating the location where the changes have occurred.

Figure 2.2: Sample watermarking properties produced by the scheme proposed in [WM01].

2.1.1 WATERMARK INSERTION SCHEME

The watermark insertion scheme proposed in [Won98] can be summarized as follows:

Step 1: Let I be an $M \times N$ image to be watermarked. Partition I into a sequence of n non-overlapping blocks X_t of size $b \times b$ pixels, where $0 \le t < n$ and $b = 8$.

Step 2: Let A be a visually meaningful binary image to be used as watermark. This image is replicated periodically or scaled to get an image W large enough to cover I. In the similar fashion, partition W into a sequence of non-overlapping blocks W_t following the same partitioning scheme performed for image I. To each block X_t, where $0 \le t < n$, there will be a corresponding binary block W_t.

Step 3: Let X_t° be the block obtained from X_t by clearing the LSBs of all the pixels. Using a cryptographically secure hash function $H(\cdot)$, compute the fingerprint $H_t \equiv H(M, N, X_t^\circ)$, where H_t denotes a 64-bit[1] message digest output.

Step 4: Compute $\hat{H}_t = H_t \oplus W_t$ by applying the element-wise exclusive-or operation.

Step 5: Generate the digital signature $S_t = E(\hat{H}_t)$ using the sender's private key of a public-key cryptosystem.

[1] The MD5 message digest produces a 128-bit output. Since each block in the sequence is 8 × 8 pixels in size and clearing the LSBs of all the pixels in the block allows 64 bits of storage, then the first or the last 64 bits of the 128-bit MD5 output is used as the message digest for each block. Depending on the choice, the same convention must be used during the watermark extraction process.

Step 6: Finally, perform $X_t^{\circ} \circ S_t$ to form the watermarked block X_t^W, where the digital signature S_t is stored in the LSBs of X_t°.

Step 7: Repeat steps 3 through 6 for each block in the sequence and all the output blocks X_t^W are assembled to form the watermarked image I^W of $M \times N$ pixels in size.

2.1.2 WATERMARK EXTRACTION SCHEME

The watermark extraction scheme proposed in [WM01] can be summarized as follows:

Step 1: Let I^W be a $M \times N$ watermarked image. Partition I^W into a sequence of n non-overlapping blocks X_t^W of size $b \times b$ pixels, where $0 \le t < n$ and $b = 8$.

Step 2: Let X_t° be the block obtained from X_t^W by clearing the LSBs of all the pixels. Using the hash function $H(\cdot)$ chosen during the watermark insertion, compute the fingerprint $H_t \equiv H(M, N, X_t^{\circ})$, where H_t denotes a 64-bit[2] message digest output.

Step 3: Extract the LSBs of all the pixels in X_t^W to form the decryption string ds_t and perform the decryption function $D(ds_t)$ to obtain the digital signature S_t.

[2] The MD5 message digest produces a 128-bit output. Since each block in the sequence is 8 × 8 pixels in size and clearing the LSBs of all the pixels in the block allows 64 bits of storage, then the first or the last 64 bits of the 128-bit MD5 output is used as the message digest for each block. Depending on the choice, the same convention must be used during the watermark insertion process.

Step 4: Compute $C_t = S_t \oplus H_t$ by applying the element-wise exclusive-or operation.

Step 5: If C_t and W_t are equal, the watermark is verified. Otherwise, the marked image I^W has been modified at block X_t^W.

Step 6: Repeat steps 2 through 5 for each block in the sequence.

2.1.3 WEAKNESSES OF WATERMARKING SCHEMES

Traditional watermarking techniques requires the receiver of the watermarked image to have the watermark such as a sequence of random numbers or a meaningful pattern used by the sender in their possession in order to compare with the extracted one [LLC00, LY03]. This implies that the watermark has to be designed separately and transmitted in some way to the particular receiver. Although the designing can be trivial, the designed watermark itself may be tapped during the transmission and have to be maintained carefully by the second party once received. The maintenance and administration can be costly and error prone at the destination when there are sensitive images coming from different sources, each having his/her own watermark for authentication. It is therefore desirable to have a scheme that relieves both parties from the designing and maintaining of a particular watermark [LLC00]. Another common limitation of many watermarking schemes is the lack of obliviousness, which makes the scheme impractical due to the requirement of the original image in the verification process. This again requires an overhead in maintaining a large database and extra access to a secure transmission channel that could put the information security of the watermark scheme at risk.

The watermarking schemes described in [Won98, WM01] use a 64-bit key to embed the watermark information into the cover image and keys of this size are insecure because it can be factored in seconds on a modern computer. Note also that an authentication scheme is really secure only if any change in the marked image is detectable, even if these changes cannot be seemingly used for any malicious purposes [BKR02]. In addition to the transmission overhead, there are other kinds of attacks that could be applied to a watermarking scheme. For instance, grayscale watermarking schemes are also generalized for color images by simply applying the same technique to the three different color planes independently [BKR02]. The attacker could interchange the different color planes (red, green, and blue) and produce a tampered image that is unnoticeable by the extraction process, although it may be hard to image how this attack could be used for malicious reasons but it will be more secure if this sort of alteration does not pass as undetectable. A possible solution to the color-swapping attack is to take into account the three color planes as one while computing the message digest H_t, this way, no matter how the color planes are ordered, each ordering used as input to the hash function will yield a totally different output.

Some more sophisticated and powerful attacks could be applied to watermarking schemes such as the cut-and-paste and birthday attack as described in [HM00, BKR02]. In the cut-and-paste attack, the attacker uses valid non-overlapping image blocks from legitimately watermarked images stored in the library and pastes them into the sender's image to produce a forgery that will pass undetected by the watermark verification scheme. Suppose that the attacker has a collection of legitimately watermarked images, all of them are of size $M \times N$ and were embedded using the same watermark image.

The sender of the image watermarks the t^{th} block of the image by computing the following:

$$\cdots,\ X_t^{\circ} \circ \underbrace{E\left(\underbrace{H\left(M, N, X_t^{\circ}\right)}_{fingerprint\ H_t} \oplus W_t\right)}_{Signature\ S_t}, \cdots$$

where $H\left(M, N, X_t^{\circ}\right)$ is the fingerprint H_t and $E\left(H\left(M, N, X_t^{\circ}\right) \oplus W_t\right)$ is the signature S_t. Note that the encryption function $E(\cdot)$ is unknown by the attacker; however, since each block X_t is marked separately without any further information about its container image except its dimensions (i.e., computing the fingerprint H_t), it is possible for an attacker to select different blocks from authentic images and build with them a new image whose watermark will be falsely verified as legitimate; for instance, assume the attacker has a collection of legitimately W-watermarked images from the receiver and Y_t is a legitimate watermarked block obtained from the library, the attacker could replace the sender's t^{th} block by:

$$Y_t^{\circ} \circ E\left(H\left(M, N, Y_t^{\circ}\right) \oplus W_t\right)$$

This replacement of the image block X_t by Y_t won't be noticed by the receiver of the image due to the fact that watermark verification for blocks X_t and Y_t will produce the same result. The receiver's verification process for the t^{th} block (that is, the attacker's replacement block) is as follows:

. . .

Step 2: Obtain Y_t° by clearing the LSBs of all the pixels from the received image block. Compute the fingerprint $H_t \equiv H\left(M, N, Y_t^\circ\right)$.

Step 3: Extract the LSBs of all the pixels of Y_t to obtain $E\left(H\left(M, N, Y_t^\circ\right) \oplus W_t\right)$. Compute $D\left(E\left(H\left(M, N, Y_t^\circ\right) \oplus W_t\right)\right)$, which gives $H\left(M, N, Y_t^\circ\right) \oplus W_t$ because the decryption function $D(\cdot)$ is the inverse of the encryption function $E(\cdot)$ canceling each other out.

Step 4: Compute $H\left(M, N, Y_t^\circ\right) \oplus \left(H\left(M, N, Y_t^\circ\right) \oplus W_t\right)$, which gives W_t (hence verified).

The complexity of this attack lies on the attacker to have a collection of legitimately W-watermarked images from the same sender. Moreover, a whole counterfeited but validly-watermarked image could be constructed by performing the counterfeiting attack proposed in [HM00] if the cut-and-paste attack is applied repeatedly to all image block contiguously (See [BKR02]).

Birthday attacks [MVV97, section 9.7] is another common watermarking attack that constitute a well known and powerful way of threatening digital signatures. In a birthday attack, the attacker searches for collisions (i.e., pair of blocks that hash to the same value, thus having the same signature). Using a hash functions that produces a message digest of m possible values, there is more than a 50% likelihood of finding a collision whenever there are approximately $\sqrt{m}$ blocks available. As a result, the only protection against this kind of attack is to increase the key size of the hash output.

The output of the hashing function used in [Won98, WM01] produces message digests of size ≤ 64 bits; hence collisions are expected to be found when the attacker has collected approximately 2^{32} blocks[3]. In order to succeed with a birthday attack, suppose the attacker has over a million of legitimately watermarked images of 640×480 pixels in size, the attacker partitions each image into blocks of 8×8 pixels making a total of 4800 individually signed blocks per image and an overall count of $1{,}000{,}000 \times 4800 \geq 2^{32}$ signatures, enough for a birthday attack.

The sender of the image watermarks the t^{th} block of the image by computing the following:

$$\cdots,\ X_t^{\circ} \circ E\left(H\left(M, N, X_t^{\circ}\right) \oplus W_t\right), \cdots$$

The attacker aims to replace a watermarked block X_t by another block B as follows:

(1) Perturb block B to generate a sufficient number of visually equivalent variants $B(1), B(2), \ldots, B(r)$ by varying the second LSBs of each of the arbitrary chosen pixels of B (the LSBs cannot be used since the signature S_t will be stored there).

(2) The attacker proceeds to find a collision by searching for a watermarked image block C_t in the database collection and a visually equivalent variant block $B(i)$ such that:

$$H\left(M, N, B(i)^{\circ}\right) = H\left(M, N, C_t^{\circ}\right),$$

[3] There are m = 2^{64} possible combinations from the 64-bit message digest output in [Won98, WM01] and collisions are expected to be found when there are $\sqrt{m} = m^{1/2} = (2^{64})^{1/2} = 2^{32}$ blocks available.

for some $i \in \{1, 2, \ldots, r\}$, and C is a legitimately W-watermarked image from the sender.

(3) The attacker replaces the sender's t^{th} block by:

$$B(i)^{\circ} \circ E\left(H\left(M, N, C_t^{\circ}\right) \oplus W_t\right)$$

This replacement of the image block X_t by $B(i)$ has a probability of success greater than 50% because of the birthday paradox [BKR02] and in the same way, won't be noticed by the receiver of the image. The receiver's verification process for the t^{th} block is as follows:

. . .

Step 2: Obtain $B(i)^{\circ}$ by clearing the LSBs of all the pixels from the received image block. Compute $H\left(M, N, B(i)^{\circ}\right)$ $\left(= H\left(M, N, C_t^{\circ}\right)\right)$

Step 3: Extract the LSBs of all the pixels from the t^{th} block to obtain $E\left(H\left(M, N, C_t^{\circ}\right) \oplus W_t\right)$. Compute $D\left(E\left(H\left(M, N, C_t^{\circ}\right) \oplus W_t\right)\right)$, which gives $H\left(M, N, C_t^{\circ}\right) \oplus W_t$.

Step 4: Compute $H\left(M, N, B(i)^{\circ}\right) \oplus \left(H\left(M, N, C_t^{\circ}\right) \oplus W_t\right)$ $\left(= H\left(M, N, C_t^{\circ}\right) \oplus \left(H\left(M, N, C_t^{\circ}\right) \oplus W_t\right)\right)$, which gives W_t (hence verified).

The whole purpose of a birthday attack is to find collisions in the middle from the visually equivalent variants $B(1), B(2), \ldots, B(r)$ with legitimately watermarked blocks $C(1), C(2), \ldots, C(r)$ as follows:

$$\begin{array}{cc|ccc}
B(1) & \rightarrow & C(1)_t & \leftarrow & C(1) \\
B(2) & \rightarrow & C(2)_t & \leftarrow & C(2) \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
B(r) & \rightarrow & C(r)_t & \leftarrow & C(r)
\end{array}$$

There are r legitimately W-watermarked images from the sender. According to [NS90], for $r \approx m^{1/2}$, where $m = 2^{bit-length\ of\ H(M,N,\ldots)}$, the "probability to meet in the middle" is $\geq p$, in other words:

$$P\begin{pmatrix} \text{"}Finding\ B(i) \in \{B(1), B(2), \ldots, B(r)\}\ and\ C \in \{C(1), C(2), \ldots, C(r)\} \\ such\ that\ H(M,N,B(i)) = H(M,N,C_t^{\circ})\text{"} \end{pmatrix} \geq p$$

where p is a fixed probability bounded away from zero matching the model B of the meet-in-the-middle attack as proposed in [NS90]. Moreover, with a very large collection of images stored in the library, the attacker has a very good opportunity of creating undetectable fake images if this process is repeated sufficient number of times.

2.2 CONTENT-BASED WATERMARKING

In the previous section, we discussed some simple attacks that could be carried out to blockwise-independent watermarking techniques to counterfeit an existing watermark image without either the original watermark owner's consent or the knowledge of the watermark insertion key. In order to make watermarking techniques more robust, it requires that the watermark should be bound with significant components of the original image. Consequently, the authors of [HM00] and [BKR02] suggest the use of contextual information to patch up some of the weaknesses of blockwise-independent schemes. Using contextual information, the signature of the block is considered valid if it is surrounded by correct blocks (see Figure 2.3). For instance, in the event that the signature of block B is changed, the signature verification will fail in all the blocks that depend on B, besides in block B itself. Thus a number as small as possible of reliance is desirable for an accurate localization of the tampered areas of an image.

The watermarking scheme proposed in [LLC00] is one of the first methods to use content dependency for image authentication and integrity verification. Their method suggests a slight variation of the scheme proposed in [Won98, WM01]. It partitions each block in halves, then the right half of block X_t is replaced with the right half of the next block $X_{(t+1) \bmod n}$ along the zig-zag scan path (see Figure 2.3c) so that neighboring blocks are related by blended data. Each combined block is then encrypted and embedded into the LSBs of the block $\widetilde{X}_t$. The same operations as those done on the insertion process should be performed on the extraction process.

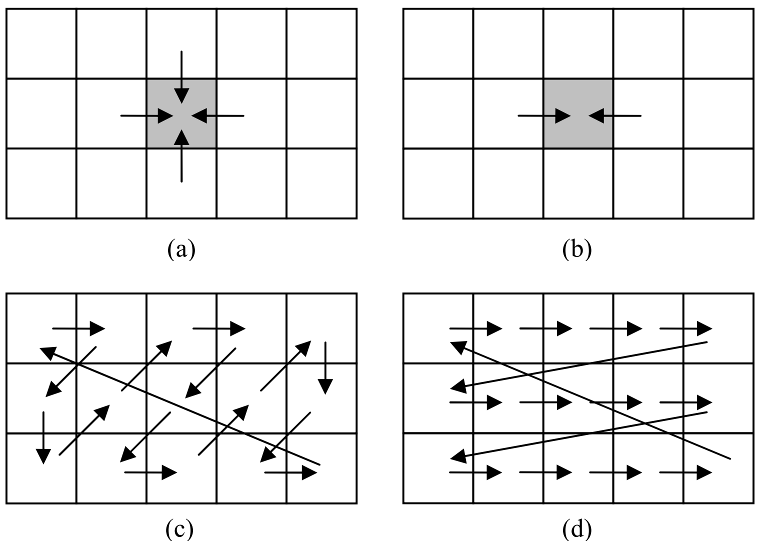


Figure 2.3: To compute the signature of block B (shown in grey), the contents of B and its neighboring blocks are taken into account.
(a) Four dependencies per block
(b) Two dependencies per block
(c) One dependency per block (zig-zag scan)
(d) One dependency per block (raster scan)

Although [LLC00] proposed a watermarking scheme that extracts the local features of the image to be used as the watermark in addition to block swapping, their scheme is still vulnerable to the simple watermarking attacks discussed in section 2.1. For birthday attacks, the attacker could perform the same operations in the similar fashion for the schemes in [Won98, WM01]. Thus, in order to succeed with a cut-and-paste attack, the attacker has only to copy the LSB-cleared contents of two half-blocks from two neighboring blocks, say X_t° and X_{t+1}°, and paste them together with the digital signature found in the LSBs of a watermarked block X_t. For this reason, [BKR02] proposes an alternative method to introduce contextual information to the fingerprint H_t to hinder the simple watermarking attacks which they called Hash Block Chaining 1 and an improved version called Hash Block Chaining 2.

2.2.1 HASH BLOCK CHAINING VERSION 1

As pointed out in [MVV97, BK99], the solution to hold back many simple attacks against watermarking schemes is to introduce contextual information. The authors of [BKR02] propose the introduction of additional dependencies to the hash function $H(\cdot)$ when computing the fingerprint H_t of each block which they called Hash Block Chaining version 1 (HBC1). In HBC1, the neighboring block of X_t°, besides X_t° itself is added as input to the hash function $H(\cdot)$ when computing the digital signature H_t. In this case, if a watermarked block X_t^W is modified, signature verification will fail in all those blocks that depend on X_t^W, besides in block X_t^W itself. Thus, a number as small as possible of dependencies (ideally, a single dependency per block) is desirable for an accurate localization of image changes. The modified computation of the signature H_t becomes as follows:

$$H_t \equiv H\left(M, N, X_t^{\circ}, X_{(t-1) \bmod n}^{\circ}, t\right)$$

Note that the block index t is added to detect blockwise rotation and likewise, the width M and height N of the original image is added is to detect image cropping. Using HBC1, the simple cut-and-paste attack can no longer be carried out because if a bogus block is pasted in place of X_t^W, with very high probability this alteration will introduce a change in the computation of the next fingerprint $H_{(t+1) \bmod n}$. Similarly, if a birthday attack is performed, the changed contents of X_t^W stimulate with very high probability a change in the dependent signature $H_{(t+1) \bmod n}$. Thus, the attacker will have to forge the signature of $X_{(t+1) \bmod n}^W$ in order to perpetrate another attack. But this induces a change in $X_{(t+2) \bmod n}^W$. Therefore, the attacker will face the problem that bad signatures propagate continuously over all blocks, eventually destroying the forged signature of the very first faked block.

Although HBC1 is effective against cut-and-paste attack, counterfeiting, and simple birthday attack, but it is not secure against an improved version of cut-and-paste attack called transplantation attack because of its limited context from neighboring blocks [BKR02]. The transplantation attack could be carried as follows: Let $C_A \to C_B$ denote the fact that the hashing of block C_B depends on the content of block C_A. Suppose that images X^W and Y^W have the blocks shown below:

$$\begin{array}{ccccccccccc} \ldots & \to & X_A^W & \to & X_D^W & \to & X_B^W & \to & X_C^W & \to & \ldots \\ \ldots & \to & Y_A^W & \to & Y_E^W & \to & Y_B^W & \to & Y_C^W & \to & \ldots \end{array}$$

where the block contents of X_A^W are identical to those of Y_A^W, the same holds for X_B^W and Y_B^W, and X_C^W and Y_C^W, but not for X_D^W and Y_E^W, then the attacker can change the block pairs as follows without been detected by the verification process:

$$\begin{array}{ccccccccccc} \ldots & \rightarrow & X_A^W & \rightarrow & Y_E^W & \rightarrow & X_B^W & \rightarrow & X_C^W & \rightarrow & \ldots \\ \ldots & \rightarrow & Y_A^W & \rightarrow & X_D^W & \rightarrow & Y_B^W & \rightarrow & Y_C^W & \rightarrow & \ldots \end{array}$$

Document images usually have large white areas, which makes them very susceptible to transplantation attacks. For instance, assume X_A^W, X_B^W, X_C^W, Y_A^W, Y_B^W, and Y_C^W were completely white noiseless blocks, then the attacker will have a pretty big chance of performing with success the transplantation attack due to the fact of the setup of the block dependencies.

HBC1 cannot withstand a more sophisticated birthday attack either. This attack replaces simultaneously two consecutive blocks X_t^W and X_{t+1}^W by forged blocks B' and B'' respectively. Three consecutive fingerprints are affected by this situation: H_t (which depends on X_t^W), H_{t+1} (which depends on both X_t^W and X_{t+1}^W), and H_{t+2} (which depends on X_{t+1}^W). The sender of the image watermarks the t^{th} block and $(t+1)^{th}$ block as follows:

$$\cdots, \overbrace{X_t^\circ \circ E\big(H\big(M, N, X_t^\circ, X_{t-1}^\circ, t\big) \oplus W_t\big)}^{t^{th}\ block}, \overbrace{X_{t+1}^\circ \circ E\big(H\big(M, N, X_{t+1}^\circ, X_t^\circ, t+1\big) \oplus W_{t+1}\big)}^{(t+1)^{th}\ block}, \cdots$$

The attacker aims to replace X_t^W and X_{t+1}^W by two blocks B' and B'' respectively as follows: (For simplicity, we omit the "mod *n*" in the indices)

(1') Perturb block B' to generate a sufficient number of visually equivalent variants $B'(1), B'(2), \ldots, B'(r')$ by varying the second LSBs of each of the arbitrary chosen pixels of B' (the LSBs cannot be used since the signature S_t will be stored there).

(2') The attacker proceeds to find a collision for the fingerprint in the t^{th} block (without loss of generality, assume that the collision occurs on $B'(1), B'(2), \ldots, B'(s')$ for a sufficiently large $s' \le r'$) by searching for a watermarked image block $C'(i)_t$ in the database collection and a visually equivalent variant block $B'(i)$ such that:

$$H\left(M, N, B'(i)^{\circ}, X_{t-1}^{\circ}, t\right) = H\left(M, N, C'(i)_t^{\circ}, C'(i)_{t-1}^{\circ}, t\right),$$

for some $i \in \{1, 2, \ldots, s'\}$, and $C'(i)$ is a legitimately W-watermarked image from the sender.

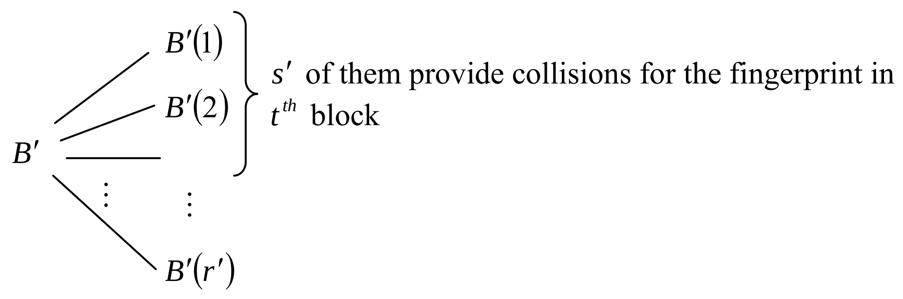
(1") Perturb block B'' to generate a sufficient number of visually equivalent variants $B''(1), B''(2), \ldots, B''(r'')$ by varying the second LSBs of each of the arbitrary chosen pixels of B'' (the LSBs cannot be used since the signature S_t will be stored there).

(2") Find a collision for the fingerprint in the $(t+2)^{th}$ block (without loss of generality, assume that the collision occurs on $B''(1), B''(2), \ldots, B''(s'')$ for a sufficiently large $s'' \le r''$) by searching for a watermarked image block $C''(j)_t$ in the database collection and a visually equivalent variant block $B''(j)$ such that:

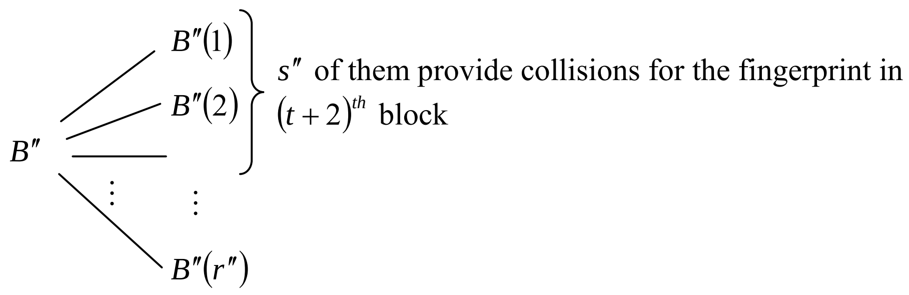
$$H\left(M, N, X_{t+2}^{\circ}, B''(j)^{\circ}, t+2\right) = H\left(M, N, C''(j)_{t+2}^{\circ}, C''(j)_{t+1}^{\circ}, t+2\right),$$

for some $j \in \{1, 2, \ldots, s''\}$, and $C''(j)$ is a legitimately W-watermarked image from the sender.

(3) Find collision for the fingerprint in $(t+1)^{th}$ block:



B'
$B'(1)$
$B'(2)$
$B'(r')$
s' of them provide collisions for the fingerprint in t^{th} block



B''
$B''(1)$
$B''(2)$
$B''(r'')$
s'' of them provide collisions for the fingerprint in $(t+2)^{th}$ block

(B', B'') are visually equivalent to $(B'(i), B''(j))$ for all $i \in \{1, 2, \ldots, r'\}$ and all $j \in \{1, 2, \ldots, r''\}$. As s' and s'' are sufficiently large, we can find a collision for the fingerprint in $(t+1)^{th}$ block such that:

$$H\left(M, N, B''(j)^{\circ}, B'(i)^{\circ}, t+1\right) = H\left(M, N, C_{t+1}^{\circ}, C_{t}^{\circ}, t+1\right),$$

for some $i \in \{1, 2, ,\ldots, s'\}$ and $j \in \{1, 2, \ldots, s''\}$, and C is a legitimately W-watermarked image from the sender.

The receiver of the image will receive the following block sequence:

$$\cdots,\quad \overbrace{B'(i)^{\circ} \circ E\big(H\big(M,N,C'(i)^{\circ}_{t},C'(i)^{\circ}_{t-1},t\big)\oplus W_t\big)}^{t^{th}\ block},$$
$$\overbrace{B''(j)^{\circ} \circ E\big(H\big(M,N,C^{\circ}_{t+1},C^{\circ}_{t},t+1\big)\oplus W_{t+1}\big)}^{(t+1)^{th}\ block},$$
$$\overbrace{X^{\circ}_{t+2} \circ E\big(H\big(M,N,X^{\circ}_{t+2},C''(j)^{\circ}_{t+1},t+2\big)\oplus W_{t+2}\big)}^{(t+2)^{th}\ block},\quad \ldots$$

The verification process for the t^{th} block is as follows:

. . .

Step 2: Obtain $B'(i)^{\circ}$ by clearing the LSBs of all the pixels from the t^{th} received image block. Compute

$$H\big(M,N,B'(i)^{\circ},X^{\circ}_{t-1},t\big)\quad \big(=H\big(M,N,C'(i)^{\circ}_{t},C'(i)^{\circ}_{t-1},t\big)\big)$$

Step 3: Extract the LSBs of all the pixels from the t^{th} block to obtain $E\big(H\big(M,N,C'(i)^{\circ}_{t},C'(i)^{\circ}_{t-1},t\big)\oplus W_t\big)$. Compute

$$D\big(E\big(H\big(M,N,C'(i)^{\circ}_{t},C'(i)^{\circ}_{t+1},t\big)\oplus W_t\big)\big),$$

which gives $H\big(M,N,C'(i)^{\circ}_{t},C'(i)^{\circ}_{t-1},t\big)\oplus W_t$.

Step 4: Compute $H\big(M,N,B'(i)^{\circ},X^{\circ}_{t-1},t\big)\oplus\big(H\big(M,N,C'(i)^{\circ}_{t},C'(i)^{\circ}_{t-1},t\big)\oplus W_t\big)$ $\big(=H\big(M,N,C'(i)^{\circ}_{t},C'(i)^{\circ}_{t-1},t\big)\oplus\big(H\big(M,N,C'(i)^{\circ}_{t},C'(i)^{\circ}_{t-1},t\big)\oplus W_t\big)\big)$, which gives W_t (hence verified).

The verification process for the $(t+1)^{th}$ block is as follows:

. . .

Step 2: Obtain $B''(j)^{\circ}$ by clearing the LSBs of all the pixels from the $(t+1)^{th}$ received image block. Compute

$$H\big(M,N,B''(j)^{\circ},B'(i)^{\circ},t+1\big)\quad \big(=H\big(M,N,C^{\circ}_{t+1},C^{\circ}_{t},t+1\big)\big)$$

Step 3: Extract the LSBs of all the pixels from the $(t+1)^{th}$ block to obtain $E\left(H\left(M,N,C_{t+1}^{\circ},C_{t}^{\circ},t+1\right)\oplus W_{t+1}\right)$. Compute

$$D\left(E\left(H\left(M,N,C_{t+1}^{\circ},C_{t}^{\circ},t+1\right)\oplus W_{t+1}\right)\right),$$

which gives $H\left(M,N,C_{t+1}^{\circ},C_{t}^{\circ},t+1\right)\oplus W_{t+1}$.

Step 4: Compute

$$H\left(M,N,B''(j)^{\circ},B'(i)^{\circ},t+1\right)\oplus\left(H\left(M,N,C_{t+1}^{\circ},C_{t}^{\circ},t+1\right)\oplus W_{t+1}\right)$$

$$\left(=H\left(M,N,C_{t+1}^{\circ},C_{t}^{\circ},t+1\right)\oplus\left(H\left(M,N,C_{t+1}^{\circ},C_{t}^{\circ},t+1\right)\oplus W_{t+1}\right)\right),$$

which gives W_{t+1} (hence verified).

Finally, the verification process for the $(t+2)^{th}$ block is as follows:

. . .

Step 2: Obtain X_{t+2}° by clearing the LSBs of all the pixels from the $(t+2)^{th}$ received image block. Compute

$$H\left(M,N,X_{t+2}^{\circ},B''(j)^{\circ},t+2\right)\quad\left(=H\left(M,N,C''(j)_{t+2}^{\circ},C''(j)_{t+1}^{\circ},t+2\right)\right)$$

Step 3: Extract the LSBs of all the pixels from the $(t+2)^{th}$ block to obtain $E\left(H\left(M,N,C''(j)_{t+2}^{\circ},C''(j)_{t+1}^{\circ},t+2\right)\oplus W_{t+2}\right)$. Compute

$$D\left(E\left(H\left(M,N,C''(j)_{t+2}^{\circ},C''(j)_{t+1}^{\circ},t+2\right)\oplus W_{t+2}\right)\right),$$

which gives $H\left(M,N,C''(j)_{t+2}^{\circ},C''(j)_{t+1}^{\circ},t+2\right)\oplus W_{t+2}$.

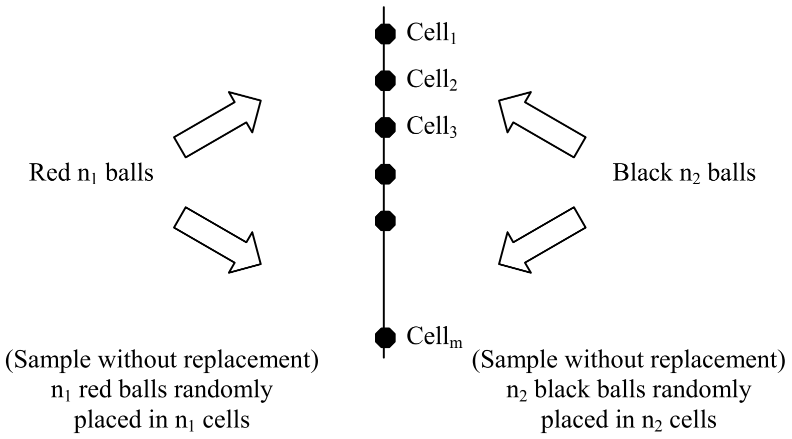
Step 4: Compute

$$H\left(M,N,X_{t+2}^{\circ},B''(j)^{\circ},t+2\right)\oplus\left(H\left(M,N,C''(j)_{t+2}^{\circ},C''(j)_{t+1}^{\circ},t+2\right)\oplus W_{t+2}\right)$$

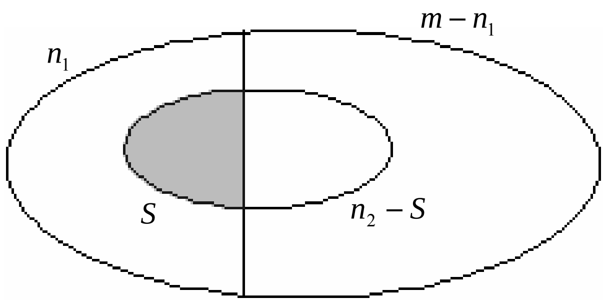
$$\left(=H\left(M,N,C''(j)_{t+2}^{\circ},C''(j)_{t+1}^{\circ},t+2\right)\oplus\left(H\left(M,N,C''(j)_{t+2}^{\circ},C''(j)_{t+1}^{\circ},t+2\right)\oplus W_{t+2}\right)\right),$$

which gives W_{t+2} (hence verified).

The complexity of the improved birthday attack is similar to the complexity of a simple birthday attack, both matching the model B of the meet-in-the-middle attack proposed in [NS90] illustrated as follows:


Cell1
Cell2
Cell3
Red n1 balls
Black n2 balls
Cellm
(Sample without replacement) n1 red balls randomly placed in n1 cells
(Sample without replacement) n2 black balls randomly placed in n2 cells

The random variable S corresponds to the number of collisions (a red ball and a black ball in the same cell) and is a hypergeometric distribution (population of m objects consisting of n_1 type-1 objects and $m-n_1$ type-2 objects; sample size n_2 ; S is the number of type-1 objects) as seen below, the expectation of S is n_1n_2/m.



For steps (1’) and (2’):

$$\begin{array}{ccc|ccc}
B'(1) & \rightarrow & \cdots & \cdots & \leftarrow & C'(1) \\
B'(2) & \rightarrow & \cdots & \cdots & \leftarrow & C'(2) \\
\vdots & \vdots & & & & \vdots \\
B'(s') & \rightarrow & \cdots & \cdots & \leftarrow & C'(s') \\
\vdots & \vdots & & & & \vdots \\
B'(r') & \rightarrow & \cdots & \cdots & \leftarrow & C'(r)
\end{array}$$

r legitimately W-watermarked images from the sender

The expected number of collisions s' is:

$$s' \cong \frac{r'r}{m}$$

Similarly for steps (1”) and (2”) with the same source of r legitimately W-watermarked images from the sender, the expected number of collisions s'' is:

$$s'' \cong \frac{r''r}{m}$$

Then, for step (3)

$$\begin{array}{l|l}
\qquad\qquad s' \cdot s'' & \quad r \\
B'(i),\ B''(j) & \\
\forall i \in \{1,2,\ldots,s'\} & \\
\forall j \in \{1,2,\ldots,s''\} &
\end{array}$$

Then, the “probability to meet in the middle” is $\geq p$, that is:

$$P\left(\begin{array}{l}
\text{"}Finding\ B'(i) \in \{B'(1), B'(2), \ldots, B'(s')\}, \\
B''(j) \in \{B''(1), B''(2), \ldots, B''(s'')\}, and \\
C \in \{C(1), C(2), \ldots, C(r)\}\ such\ that \\
H(M, N, B''(j), B'(i), t+1) = H(M, N, C^{\circ}_{t+1}, C^{\circ}_{t}, t+1)\text{"}
\end{array}\right) \geq p$$

where p is a fixed probability bounded away from zero when $(s' \cdot s'') \cdot r \approx m$

i.e., $\left(\frac{r'r}{m} \cdot \frac{r''r}{m}\right) \cdot r \approx m$

i.e., $r' \cdot r'' \approx \left(\frac{m}{r}\right)^3$

say $r \approx m^{1/2}$, then $r' \approx r'' \approx \left(m^{3/2}\right)^{1/2} = m^{3/4}$.

2.2.2 HASH BLOCK CHAINING VERSION 2

The authors of [BKR02] improved HBC1 to prevent both transplantation attack and improved birthday attack. Their enhanced version is called Hash Block Chaining version 2 (HBC2) and makes use of nondeterministic signatures schemes. Some signature schemes (for example, DSA and Schnorr's scheme [MVV98], section 11.5) are nondeterministic in the sense that each individual signature depends not only on the hash function, but also on some randomly chosen parameter. The computation of the fingerprint H_t becomes as follows:

$$H_t \equiv H\left(M, N, X_t^{\circ}, X_{(t-1)\bmod n}^{\circ}, t, S_{t-1}\right)$$

where S_{t-1} is the nondeterministic signature of block X_{t-1} and $S = \phi$ for the first block because by the time H_0 is computed, S_{n-1} would not be known yet. The improved birthday attack could no longer be successful in HBC2 because the signature of one block depends not only on the content of its neighboring block, but also on its nondeterministic signature. Assume an attacker has successfully performed an improved birthday attack by replacing two valid consecutive blocks X_t and X_{t+1} by B_t and B_{t+1} and their respective signatures S_t, S_{t+1}, and S_{t+2} by L_t, L_{t+1}, and L_{t+2} leaving the content of X_{t+2} untouched except its signature. The replacement of two valid consecutive blocks is much harder in HBC2 than in HBC1 due to the nondeterministic signature and signature-dependency [BKR02]. Unlike HBC1, HBC2 will report an alteration when computing H_{t+3} because it depends not only on the contents of X_{t+2} but also on the original embedded digital signature of X_{t+2} which in this case was replaced by L_{t+2}; in other words,

$$H\left(M, N, \overset{\circ}{X}_{t+3}, \overset{\circ}{X}_{t+2}, t+3, S_{t+2}\right) \neq H\left(M, N, \overset{\circ}{X}_{t+3}, \overset{\circ}{X}_{t+2}, t+3, L_{t+2}\right)$$

HBC2 is capable of detecting whether any blocks have been modified, reshuffled, deleted, inserted, or transplanted from a legitimate watermarked image. In addition, it has the ability to detect the altered blocks of a tampered image or to identify the boundaries of a region if a large validly watermarked region is copied and pasted onto signed image.

CHAPTER 3

IMPROVEMENT

As discussed in Chapter 2, the blockwise fragile authentication watermarks proposed in [Won98, LLC00, and WM01] are not secure and are susceptible to the cut-and-paste and the well known birthday attacks. For this reason, the authors of [HM00] and [BKR02] suggest the introduction of additional context dependencies to hinder some of the weaknesses of blockwise-independent schemes. In order to make these schemes more secure, [BKR02] makes the signature of each block depend on the contents of its neighboring blocks and attempts to maximize the change localization resolution by using only one dependency per block with the scheme which they called HBC1. However, neighbor-content dependencies are still not enough to hold back enhanced forgery techniques such as the transplantation and improved birthday attack. As a result, [BKR02] proposes an improved version of HBC1 which they called HBC2 that uses a non-deterministic digital signature together with a signature-dependent scheme to thwart these sophisticated watermarking attacks. As mentioned in [BKR02], the only protection against birthday attacks is to increase the hash size, for this reason, we propose and implement an improved version of HBC2 that increases the hash output to 128 bits instead of 64 bits while using the same block size of 8×8 pixels. The hash key for the first and last block will remain as 64 bits in length and all the remaining blocks in the sequence will use a 128-bit hash key that will be stored in two separate but consecutive blocks of the scan path which at the same time are exclusive-ored with the encrypted digital signature; this way, this method will provide a more secure way to protect the digital signatures and the contents of the cover image. This chapter will be organized as follows: the following two sections will describe our insertion and extraction scheme. Section 3.3 will show a comparison of experimental results obtained from our implementation of HBC2 and our improved scheme. Finally, section 3.4 will contain a brief discussion of why our watermarking scheme is more secure against the proposed watermarking attacks.

3.1 PROPOSED WATERMARK INSERTION SCHEME

Our proposed watermark insertion scheme can be summarized as follows:

Step 1: Let I be an $M \times N$ image to be watermarked. Partition I into a sequence of n non-overlapping blocks X_t of size $b \times b$ pixels where $0 \leq t < n$ and $b = 8$.

Step 2: Let A be a visually meaningful binary image to be used as watermark. This image is replicated periodically or scaled to get an image W large enough to cover I. In the similar fashion, partition W into a sequence of non-overlapping blocks W_t following the same partitioning scheme performed for image I. To each block X_t, where $0 \le t < n$, there will be a corresponding binary block W_t.

Step 3: Let S'_{t-1} be the non-deterministic signature of the previous block in the sequence. Initially, set $S'_0 = \phi$ because the previous non-deterministic signature for the first block hasn't been computed yet.

Step 4: Let X_t° and $X_{(t-1)\bmod n}^{\circ}$ be the block obtained from X_t and its previous block respectively by clearing the LSBs of all pixels. Note that the previous block for the first block is the last block of the sequence. Using a cryptographically secure hash function $H(\cdot)$, compute the fingerprint $H_t \equiv H\left(M, N, X_t^{\circ}, X_{(t-1)\bmod n}^{\circ}, t, S'_{t-1}\right)$, where H_t corresponds a 128-bit message digest output.

Step 5: Partition H_t into two equal halves of 64 bits each. Denote H_t^1 and H_t^2 as the first and second half respectively of the 128-bit message digest obtained in step 4.

Step 6: Compute $\hat{H}_t = H_t^1 \oplus W_t$ by applying the element-wise exclusive-or operation between the first half of the message digest H_t and the respective binary block of the watermark image used.

Step 7: Generate the digital signature $S_t = E\left(\hat{H}_t\right)$ using the private key of a public-key cryptosystem.

Step 8: If $t = 0$ (i.e., first block of the sequence), set $S'_0 = S_0$; otherwise, compute $S'_t = S_t \oplus H_{(t-1)\bmod n}^2$ by applying the element-wise exclusive-or operation between the digital signature S_t and the second half of the message digest obtained from the previous block.

Step 9: Finally, form the watermarked block X_t^W by performing $X_t^{\circ} \circ S'_t$ where the exclusive-ored signature S'_t is stored in the LSBs of X_t°.

Step 10: Repeat steps 4 through 9 for each block in the sequence and all the output blocks X_t^W are assembled together to form the watermarked image I^W of $M \times N$ pixels in size.

3.2 PROPOSED WATERMARK EXTRACTION SCHEME

Our proposed watermark extraction scheme can be summarized as follows:

Step 1: Let I^W be a $M \times N$ watermarked image. Partition I^W into a sequence of n non-overlapping blocks X_t of size $b \times b$ pixels where $0 \leq t < n$ and $b = 8$.

Step 2: Let S'_{t-1} be the non-deterministic signature of the previous block in the sequence. Initially, set $S'_0 = \phi$ because as mentioned in the watermark insertion step, there is no previous non-deterministic signature for the first block in the sequence.

Step 3: Let X_t° and $X_{(t-1) \bmod n}^{\circ}$ be the block obtained from X_t^W and its previous block respectively by clearing the LSBs of all pixels. Note that the previous block for the first block is the last block of the sequence. Using a cryptographically secure hash function $H(\cdot)$, compute the fingerprint

$H_t \equiv H\left(M, N, X_t^{\circ}, X_{(t-1)\bmod n}^{\circ}, t, S'_{t-1}\right)$, where H_t corresponds a 128-bit message digest output.

Step 4: Partition H_t into two equal halves of 64 bits each. Denote H_t^1 and H_t^2 as the first and second half respectively of the 128-bit message digest obtained in step 4.

Step 5: Extract the LSBs of all the pixels in X_t^W to form the decryption string ds_t.

If $t = 0$ (i.e., first block of the sequence), set $S'_0 = ds_0$; otherwise, compute $S'_t = ds_t \oplus H_{(t-1)\bmod n}^2$ by applying the element-wise exclusive-or operation between the decryption string ds_t and the second half of the message digest obtained from the previous block.

Step 6: Perform the decryption function $D(S'_t)$ to obtain the signature S_t. Compute $C_t = S_t \oplus H_t^1$ by applying the element-wise exclusive-or operation between the signature S_t and the first half of the message digest obtained in step 3.

Step 7: If C_t and W_t are equal, the watermark is verified. Otherwise, the marked image I^W has been modified at block X_t^W.

Step 8: Repeat steps 3 through 7 for each block in the sequence.

3.3 EXPERIMENTAL RESULTS

We have tested our HBC2 implementation and our proposed watermarking scheme on high-quality color images and the results are promising. We were able to embed a watermark image into the test images without any visible artifacts, and retain faithful color and details. In addition, the embedding process produces bit changes beyond the LSBs. For illustration, we show in Figure 3.1 a sample source image of 800×600 pixels in size. Figure 3.2 and Figure 3.3 shows a bi-level watermark image of 70×30 pixels and a periodically replicated watermark image to match the dimensions of the source image of 800×600 pixels respectively. Figure 3.4 shows the stamped source image with the watermark image embedded using the HBC2 and our proposed watermarking scheme. Figure 3.5 and Figure 3.6 illustrates the extracted watermark image using the proper and improper verification keys respectively. Figure 3.7 shows a library image from the attacker's collection and the altered version of the watermarked image acknowledged by the receiver after performing a transplantation attack that appears to be "original" to the naked eye. Finally, Figure 3.8 exemplifies the extracted watermark by HBC2 and our proposed scheme with the regions of alterations automatically identified.

Figure 3.1: Sample source image of 800 x 600 pixels in size to be used as the cover image.

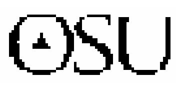


Figure 3.2: Bi-Level watermark image of 70 x 30 pixels

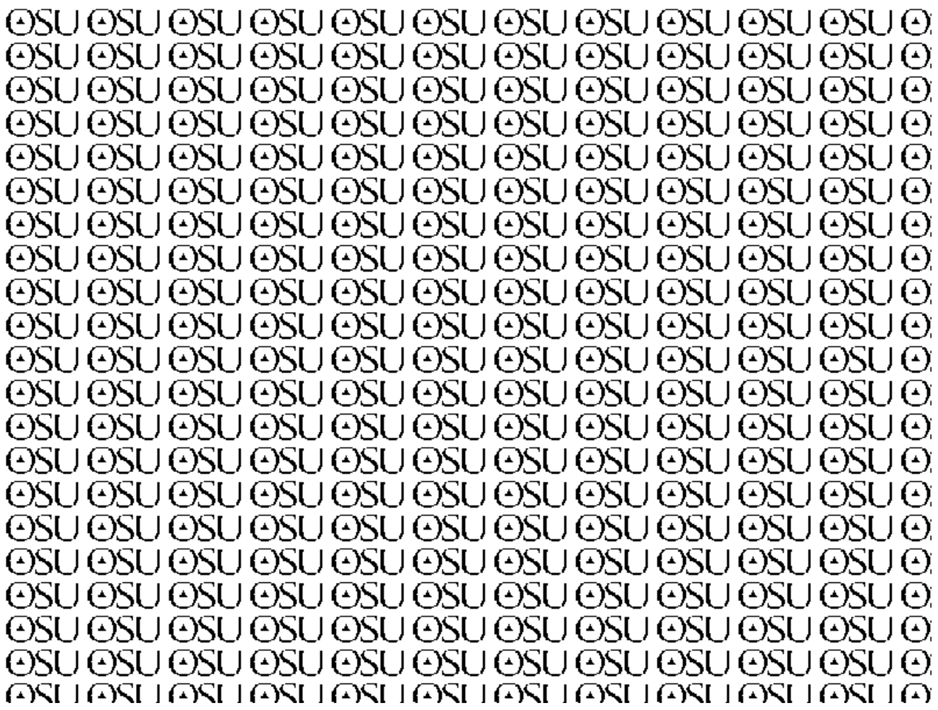
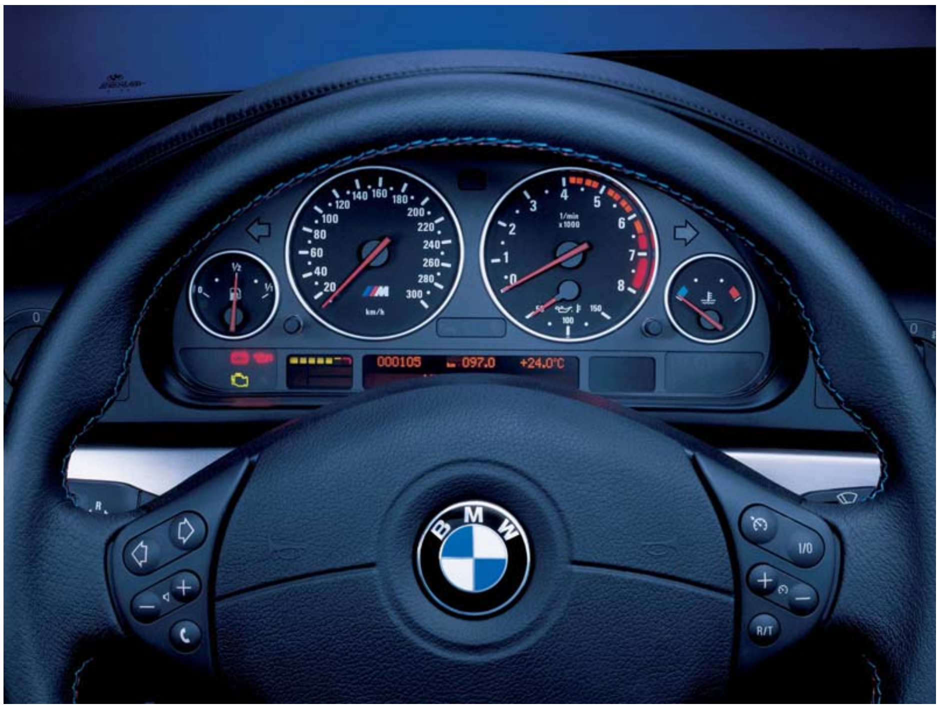


Figure 3.3: Bi-Level watermark image periodically replicated to 800 x 600 pixels.


(a)

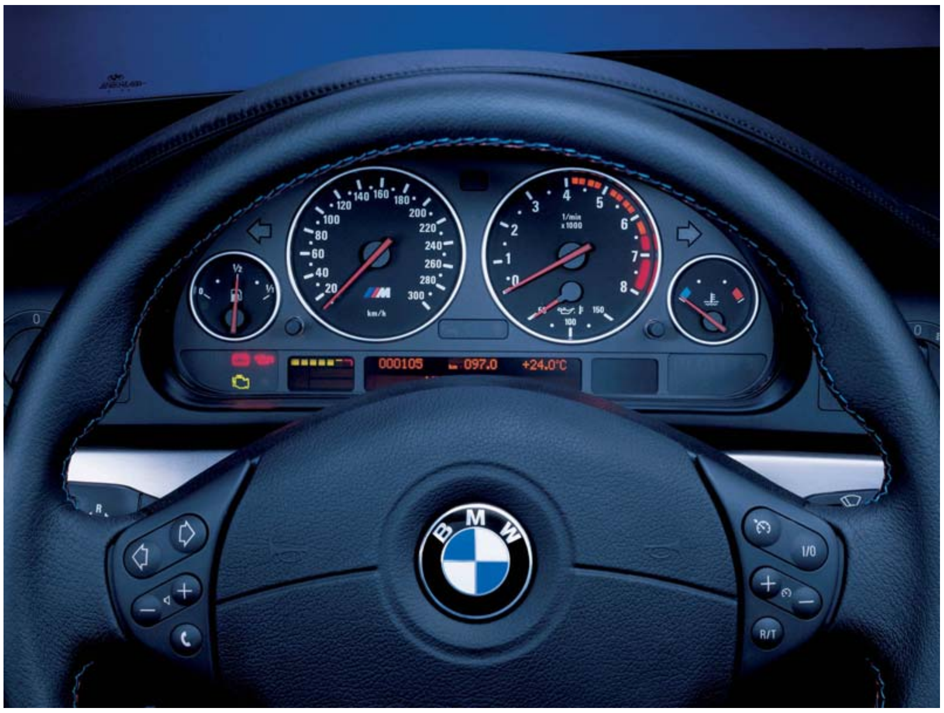
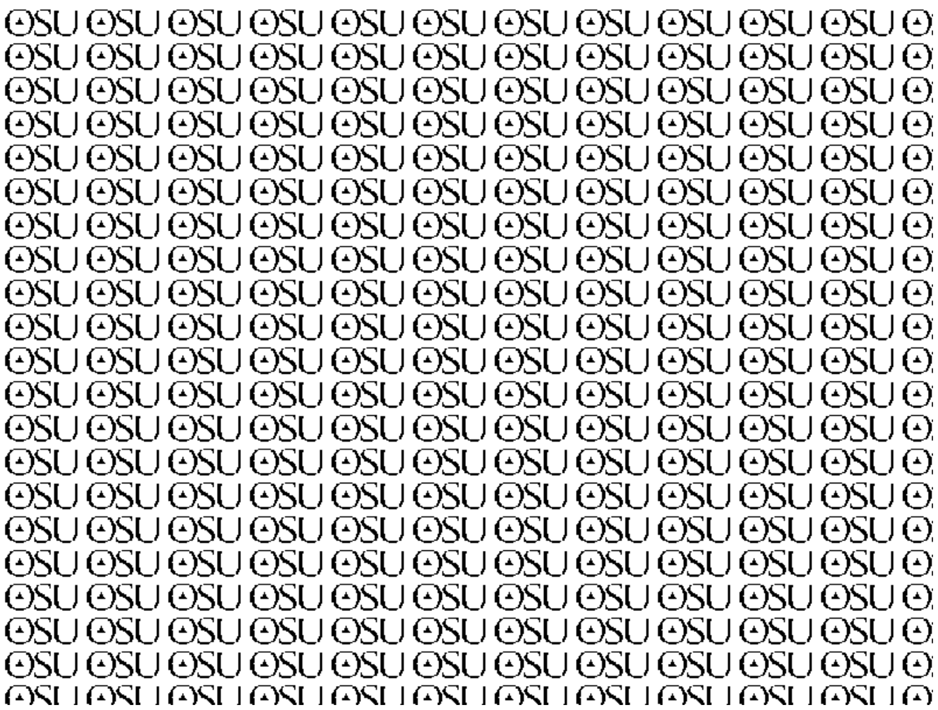

(b)

Figure 3.4: Stamped source image of Figure 3.1 with the watermark image of Figure 3.3 using the RSA key E = 5 and N = 18204938255760143519.
(a) Using HBC2 watermarking scheme.
(b) Using our proposed watermarking scheme.


(a)

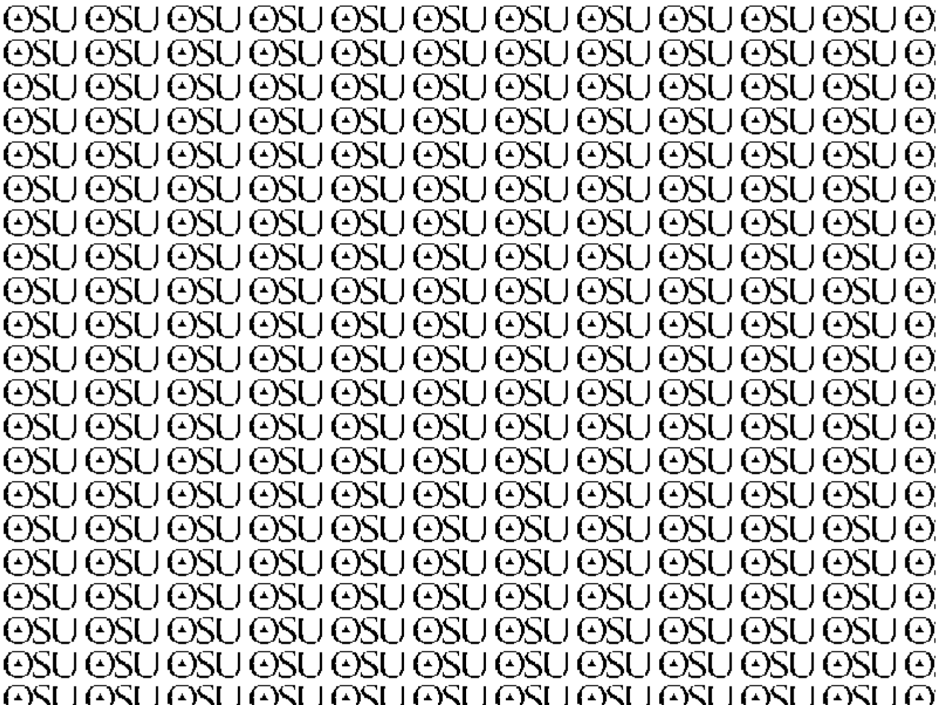
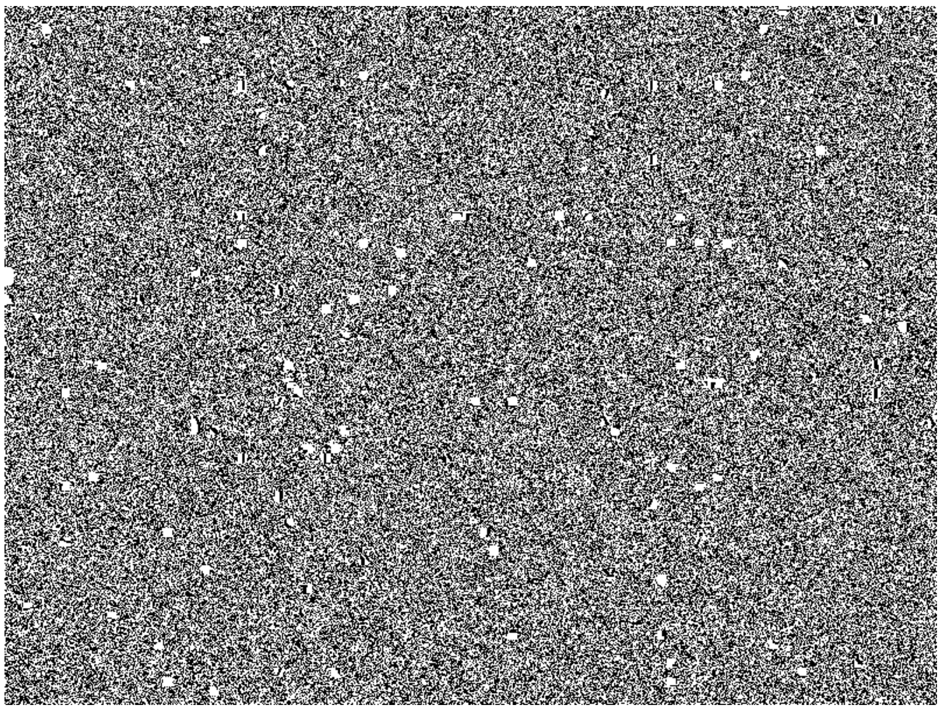

(b)

Figure 3.5: Extracted watermark image using the proper verification key of
D = 14563950597781346957 and N = 18204938255760143519.
(a) Using HBC2 watermarking scheme.
(b) Using our proposed watermarking scheme.


(a)

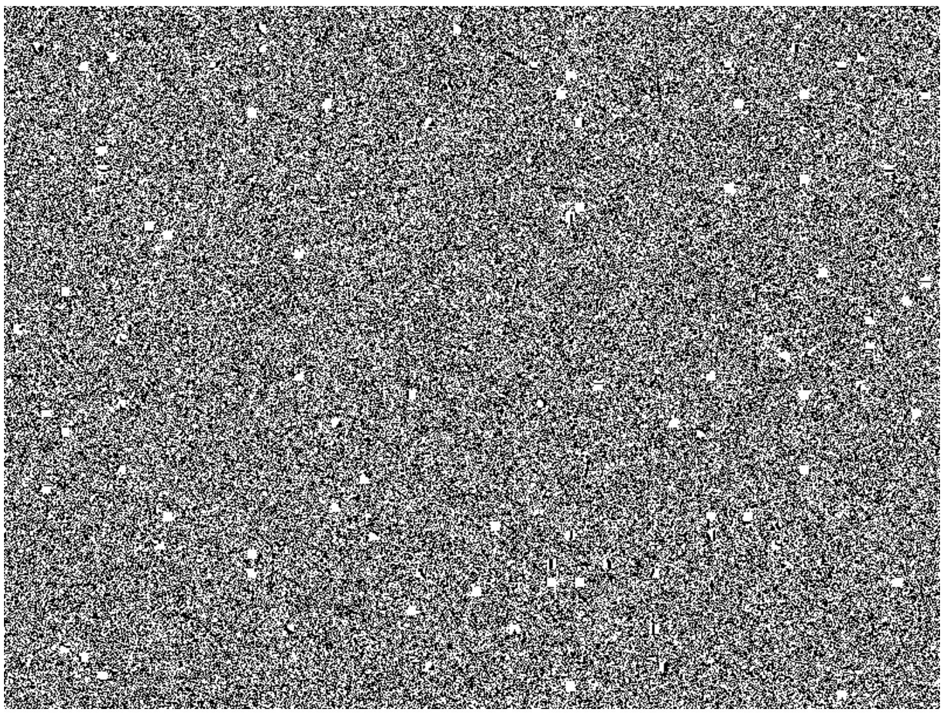
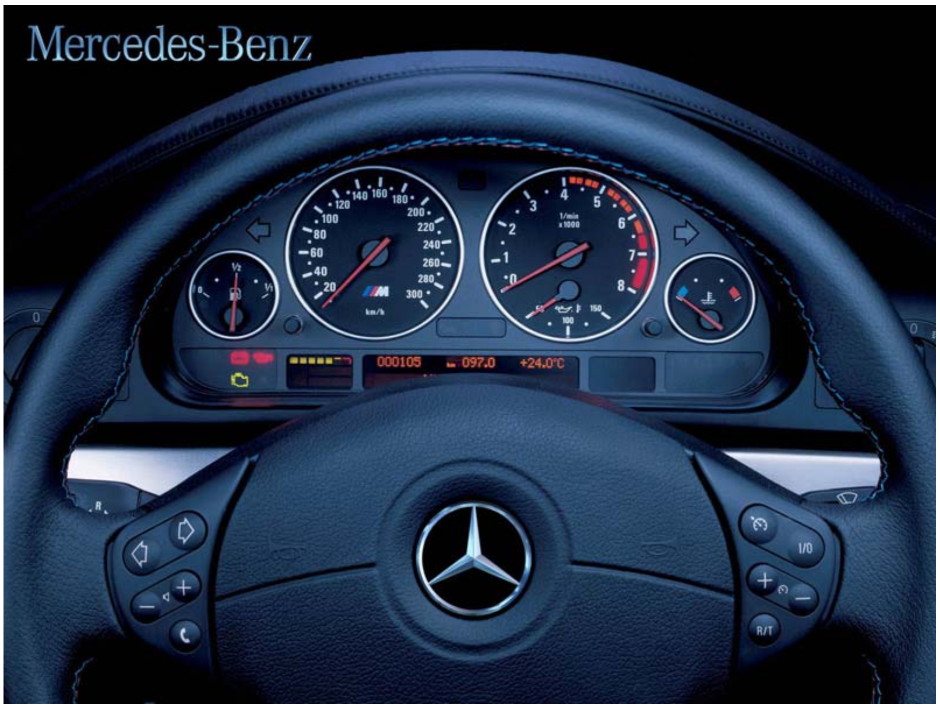
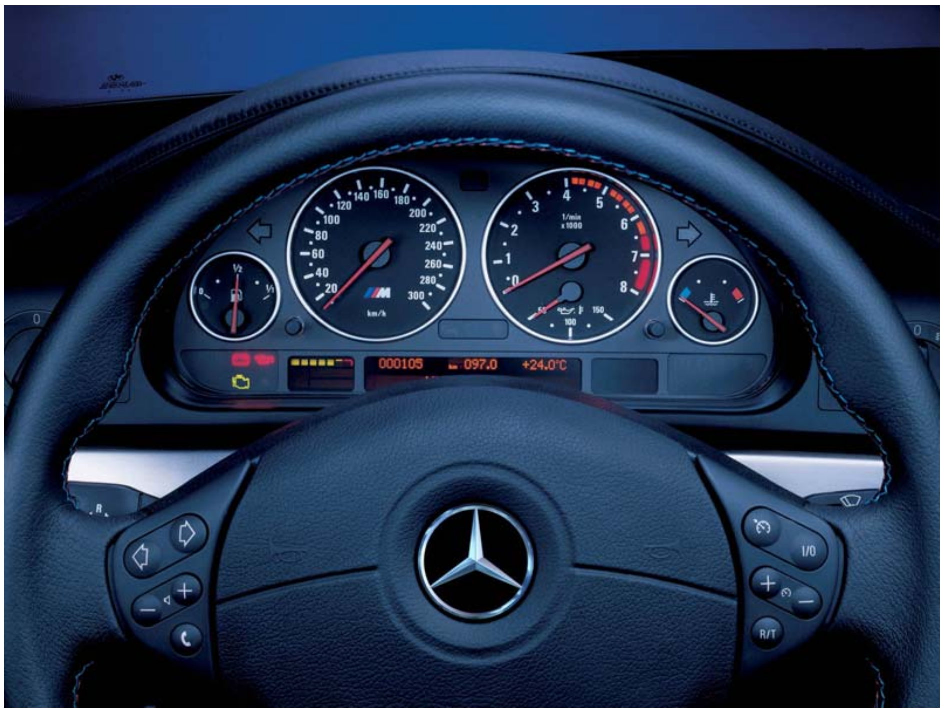

(b)

Figure 3.6: Extracted watermark image using the improper verification key of
D = 13761193859040633293 and N = 17201492332096682459.
(a) Using HBC2 watermarking scheme.
(b) Using our proposed watermarking scheme.

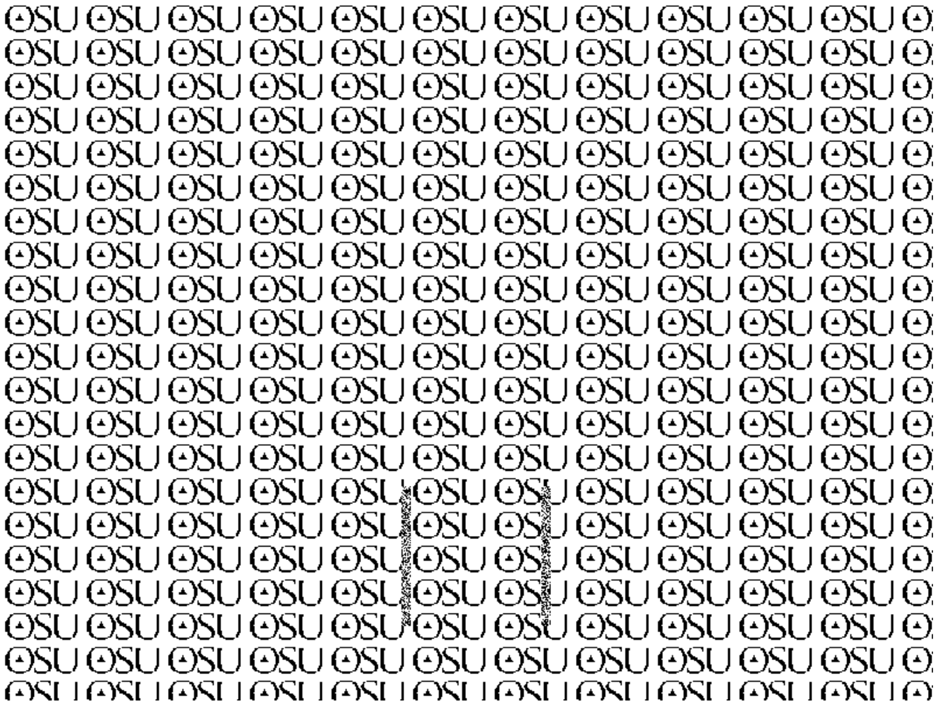


(a)



(b)

Figure 3.7: An attacker’s library image and a modified watermarked image.
(a) Library image from attacker’s large collection of images.
(b) Altered image after performing transplantation attack acknowledged by the receiver.


(a)

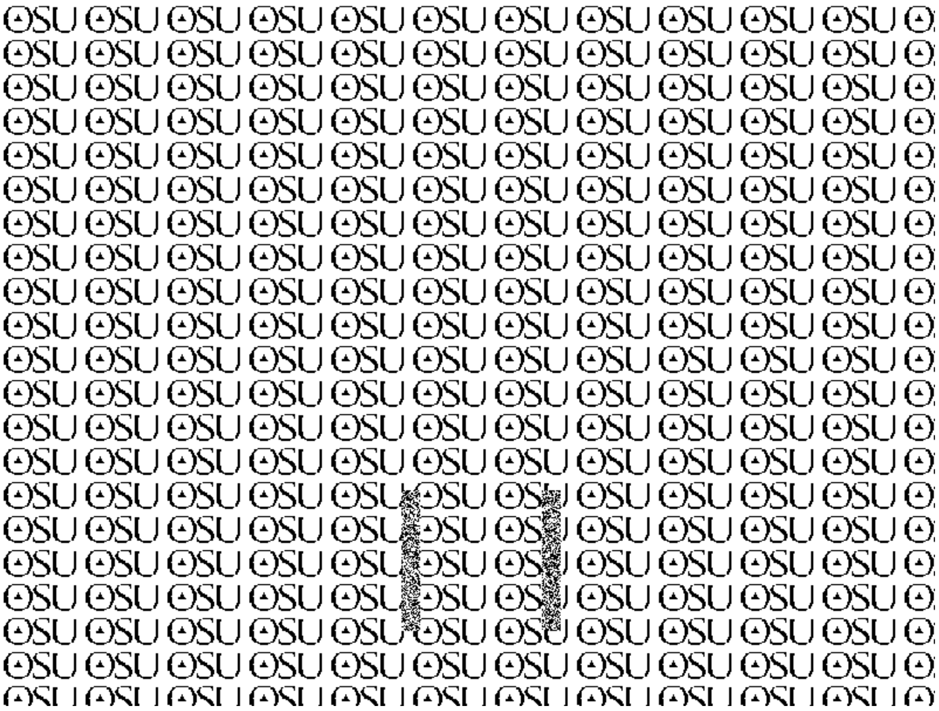

(b)

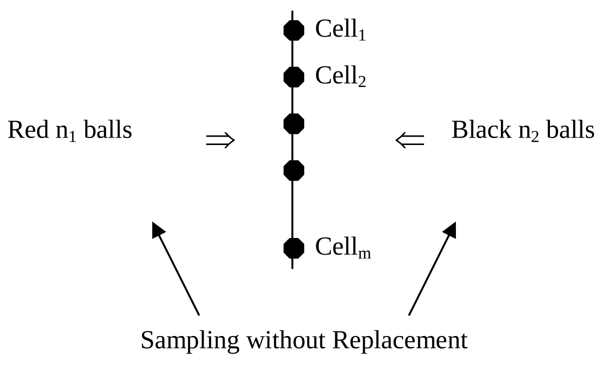
Figure 3.8: Extracted watermark from image of Figure 3.7(b) using the proper verification key of D = 14563950597781346957 and N = 18204938255760143519.
(a) Using HBC2 watermarking scheme.
(b) Using our proposed watermarking scheme.

Without the proper verification key, the watermark is very difficult to extract and impossible to identify. In addition to the sample test cases presented earlier, our proposed scheme as same as the HBC2 is able to detect geometric transformations such as scaling, cropping, and color swapping without mentioning cut-and-paste, birthday, transplantation, and improved birthday attacks. This is because geometric transformations that adjust the dimensions of the image affects the values of M and N used in the computation of the one-way hash function $H(\cdot)$. Similarly, swapping color planes in the watermarked image would yield a different color ordering for the block X_t^W which also affects the computation of the message digest H_t.

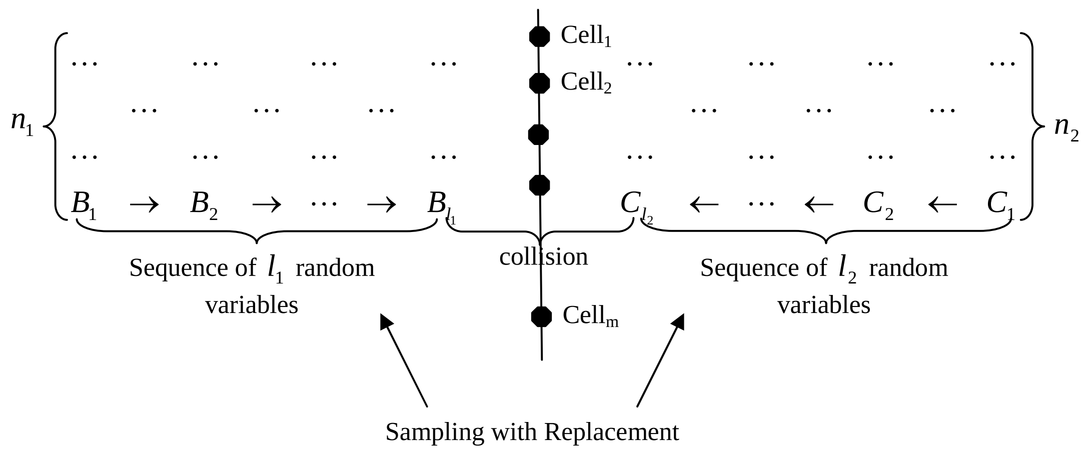
3.4 DISCUSSION OF IMPROVED WATERMARKING SCHEME

According to [BKR02], there is more than 50% chance of finding a collision between blocks whenever the attacker has collected approximately $\sqrt{m}$ blocks where m is the total number of possible values produced by the hash function $H(\cdot)$. The hash output in the schemes proposed in [Won98, LLC00, and WM01], HBC1, and HBC2 is 64 bits in length; hence collisions are expected to be found when the attacker has collected around 2^{32} blocks. Keeping this in mind, we took another step with the proposed HBC2 scheme and increase the hash function output from 64 to 128 bits and use two separate but consecutive blocks of 8×8 pixels as storage. Although this increase in hash length will not permanently prevent the attacker from finding collisions, it sure will provide a better protection against birthday-type attacks. According to [NS90], classical birthday paradox attacks are credited to the probability to meet in the middle and can be carried out using one of the following three models:

Model A: Sampling without replacement



Model B: Sampling with replacement, fix l_1, l_2



Cell$_1$
Cell$_2$
n_1
n_2
$B_1 \rightarrow B_2 \rightarrow \cdots \rightarrow B_{l_1}$
$C_{l_2} \leftarrow \cdots \leftarrow C_2 \leftarrow C_1$
Sequence of l_1 random variables
collision
Sequence of l_2 random variables
Cell$_m$
Sampling with Replacement

Model C: Mixture of Model A and Model B.

No matter which model is chosen to apply the birthday paradox attack, the three models behave similarly asymptotically:

$$\text{if } n_1 \cdot n_2 \approx m \text{ then the probability of collision is } \geq p$$

where p is a fixed probability bounded away from zero. Also, with the increase of bit-length of the hash output and using the same small-size blocks of 8×8 pixels, our scheme will be able to host more embedded data (for hash-function values) while maintaining accurate localization of image changes and produce high-quality watermarked images as well. Finally, our scheme alike to HBC2 is nondeterministic in nature and also maintains the content-dependency among blocks.

CHAPTER 4

CONCLUSION

In this research, we have advanced some more steps heading to a more secure blockwise fragile authentication watermarking. Many block-based watermarking schemes proposed in the literature lack of blockwise dependency and as a result, they are at risk from simple counterfeiting techniques such as cut-and-paste and birthday attacks. In addition, the lack of content dependency allows attackers to produce forgeries without either the original watermark owner’s consent or knowledge of the insertion key. These facts strongly suggest that the possibility of a counterfeiting attack should be taken into account before adopting an oblivious and blockwise independent watermarking technique for any application [HM00]. Since the Internet has become a commonplace to make collection of images accessible to a vast number of users and the increase of sophisticated image processing software commercially available today, malicious parties have increased their opportunities of producing counterfeit work that closely resemble originals that are undetectable by the human eye putting intellectual property and digital media at risk and constant proliferation.

In Chapter 2, we have discussed public-key watermarking schemes proposed in [Won98, WM01] to be insecure against attacks as simple as block cut-and-paste and the well-known birthday attack and argued the need of content dependency to make these schemes more robust. Although [LLC00] was one of the first to use content dependency in their watermarking scheme by dividing and swapping blocks, and by extracting the local features of an image and use it as the watermark, their scheme is still subject to the counterfeiting techniques mentioned earlier. For this reason, [BKR02] proposed the HBC1 scheme which counters these attacks by making the signature of each block depend on the contents of the neighboring block. Moreover, we have also discussed about schemes that augments the hash input with content of neighboring blocks such as HBC1 are still vulnerable to superior malicious attacks known as transplantation and improved birthday attack. Finally, as a solution for the improved attacks, [BKR02] also proposed an enhanced version of HBC1 called HBC2 that uses non-deterministic signatures and signature dependent schemes to frustrate attackers from performing transplantation and improved birthday attacks on blockwise independent schemes.

To finish this research, in Chapter 3, we have improved HBC2 and proposed a new invisible fragile watermarking technique for image verification that uses 128-bit message digests instead of 64. This increase in length does not change the proposed block size of 8×8 pixels; rather, it uses two separate but consecutive blocks in the sequence to store the key by means of the exclusive-or operation. Also, doubling the hash size provides a more secure way to protect the contents of the image and decreases the chances for collisions to be found. Thus, in order for an attacker to perform a counterfeit attack, he or she would have to collect many more visually significant variants and perform more work to succeed.

Our watermarking process produces a verification key for each stamped image and does not introduce visual artifacts retaining the quality of the images. In addition, our scheme detects and reports any changes made to the watermarked image since the time the watermark was inserted. The embedded watermark can only be extracted by someone who has possession of a proper verification key. This implies that someone who does not have a valid key will not be able to forge a watermark that will pass undetected. Alterations to a watermarked image produce output that resembles random noise on the extracted watermark image, which can be visually and automatically identified. This technique offers a more reliable way for image verification to detect and localize unauthorized image modifications. Finally, our technique provides means of ensuring data integrity; adds more security to the contents of digital media and allows recipients of an image to verify its authenticity with ease as well as display the ownership information embedded within an image.

REFERENCES

[BK99] P. S. L. B. Barreto and H. Y. Kim. Pitfalls in public key watermarking. In *Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing*, pages 241-242, October 1999.

[BKR02] P. S. L. M. Barreto, H. Y. Kim, and V. Rijmen. Toward secure public-key blockwise fragile authentication watermarking. In *Proceedings on Vision, Image and Signal Processing*, 149(2): 57-62, April 2002.

[BO96] H. Bergel and L. O'Gorman. Protecting ownership rights through digital watermarking. *IEEE Computer Magazine*, 29(7):101-103, July 1996.

[CKLS97] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shammon. Secure spread spectrum watermarking for multimedia. In *Proceedings of the IEEE Transactions on Image Processing*, 6(12):1673-1687, IEEE Computer Society, December 1997.

[CMB00] I. J. Cox, M. Miller, and J. Bloom. Watermarking applications and their properties. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, pages 27-29, IEEE Computer Society, March 2000.

[CYY98] S. Craver, B. L. Yeo, and M. Yeung. Technical trials and legal tribulations. *Communications of the ACM*, 41(7):46-54, Association for Computing Machinery, July 1998.

[HM00] M. Holliman and N. Memon. Counterfeiting attacks on oblivious block-wise independent invisible watermarking schemes. In *Proceedings of the IEEE Transactions on Image Processing*, 9(3):432-441, IEEE Computer Society, March 2000.

[KP00] S. Katzenbeisser, F. A. P. Petitcolas. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, 2000.

[LLC00] C. T. Li, D. C. Lou, and T. H. Chen. Image authentication and integrity verification via content-based watermarks and a public key cryptosystem. In *Proceedings of the International Conference on Image Processing*, volume 3, pages 694-697, IEEE Computer Society, September 2000.

[LY03] C. T. Li and F. M. Yang. One-dimensional neighborhood forming strategy for gragile watermarking. *Journal of Electronic Imaging*, 12(2):2840291, April 2003.

[MBY97] F. Mintzer, G. W. Braudaway, and M. Yeung. Effective and ineffective digital watermarks. In *Proceedings of the International Conference on Image Processing*, volume 3, pages 9-12, IEEE Computer Society, October 1997.

[MVV97] A. J. Menezes, P. C. van Oorshot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[MW98] N. Memon and P. W. Wong. Protecting digital media content. Communications of the ACM, 41(7):35-43, Association for Computing Machinery, July 1998.

[NS90] K. Nishimura and M. Sibuya. Probability to meet in the middle. *Journal of Cryptology*, 2(1):13-22, 1990.

[Riv92] R. L. Rivest. The MD5 message digest algorithm. *Technical Report*, 1992.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method of obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, volume 21, pages 120-126, Association of Computing Machinery, February 1978.

[WD96] R. B. Wolfgang and E. J. Delp. A watermark for digital images. In *Proceedings of the International Conference on Image Processing*, volume 3, pages 219-222, IEEE Computer Society, September 1996.

[WM01] P. W. Wong and N. Memon. Secure and public key image watermarking schemes for image authentication and ownership verification. In *Proceedings of the IEEE Transactions on Image Processing*, 10(10):1593-1601, IEEE Computer Society, October 2001.

[Won98] P. W. Wong. A public key watermarking for image verification. In *Proceedings of the International Conference on Image Processing*, volume 1, pages 455-459, IEEE Computer Society, 1998.

[YM97] M. Yeung and F. Mintzer. An invisible watermarking technique for image verification. In *Proceedings of the International Conference on Image Processing*, volume 2, pages 680-683, IEEE Computer Society, 1997.

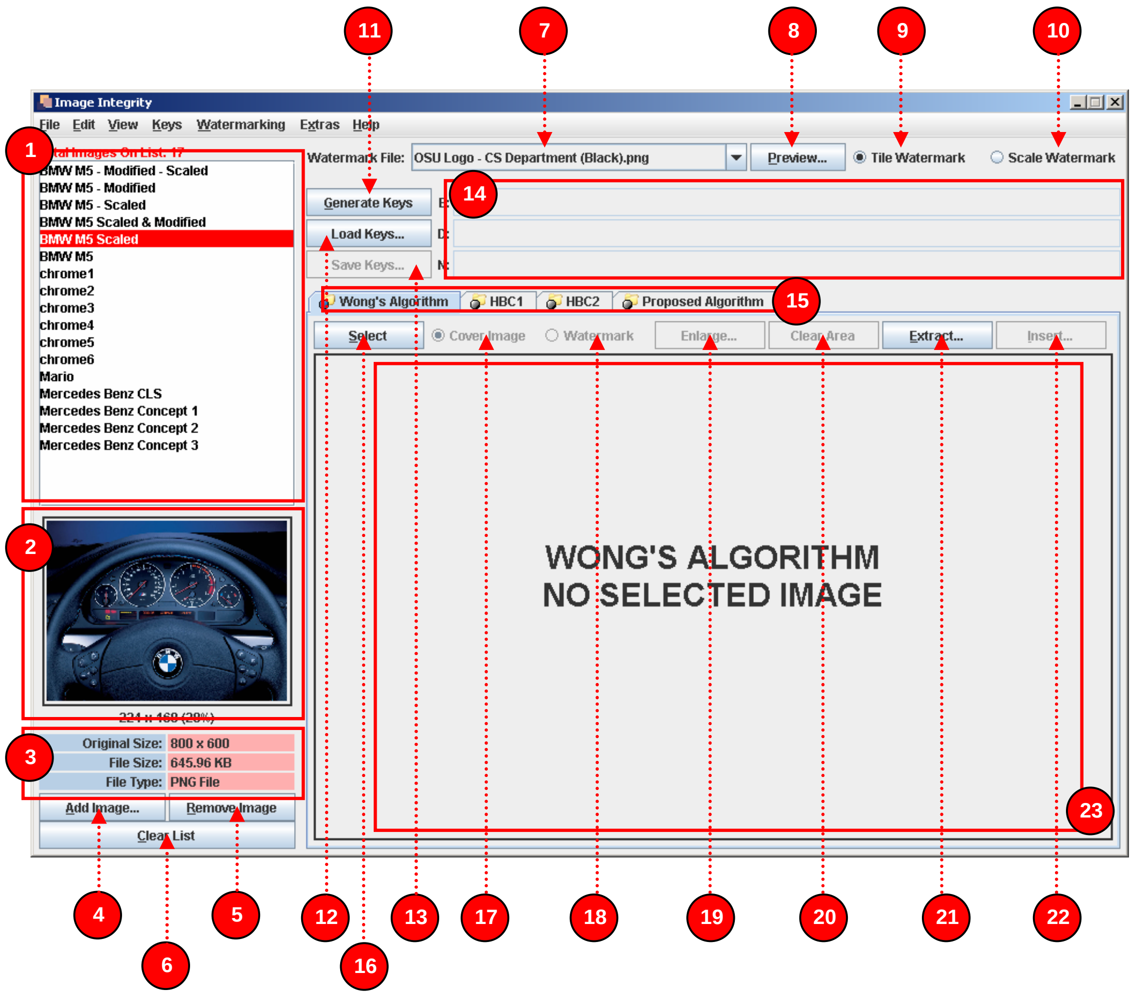
APPENDIX A

USER'S MANUAL

We implemented an application that will allow users to embed an information carrying signal into color images and likewise, extract the watermark from previously watermarked images. Our application was written in Java using the J2SE v1.5.0 Software Development Kit package and tested on an AMD Athlon 1.05 GHz processor with 512 MB of RAM running Microsoft Windows XP SP2. The main application interface is simple and very user friendly and allows users the flexibility to choose and compare among the four different watermarking algorithms presented throughout this research (original scheme proposed in [WM01], HBC1 and HBC2 schemes proposed in [BKR01], and our scheme proposed in Chapter 3). The remaining of this chapter will provide an insight and quick guide for users on how to use our implemented application.

A.1 GRAPHICAL USER INTERFACE COMPONENTS LAYOUT

The user graphical user interface layout of our application is described as follows:



Watermark File: OSU Logo - CS Department (Black).png

[✓ Tile Watermark] Scale Watermark

Original Size:	800 x 600
File Size:	645.96 KB
File Type:	PNG File

1	Image List
2	Thumbnail Display Area
3	Image Information Panel
4	Add Image Button
5	Remove Image Button
6	Clear Image List Button
7	Watermark File Combo Box
8	Watermark Preview Button
9	Tiled Watermark Option
10	Scaled Watermark Option
11	Generate Keys Button
12	Load Keys Button
13	Save Keys Button
14	Key Information Panel
15	Algorithm Selection Tabs
16	Select Image Button
17	Cover Image Option
18	Watermark Option
19	Enlarge Image Button
20	Clear Display Area Button
21	Watermark Extraction Button
22	Watermark Insertion Button
23	Selected Image Display Area

A.2 ADDING IMAGES TO THE IMAGE LIST

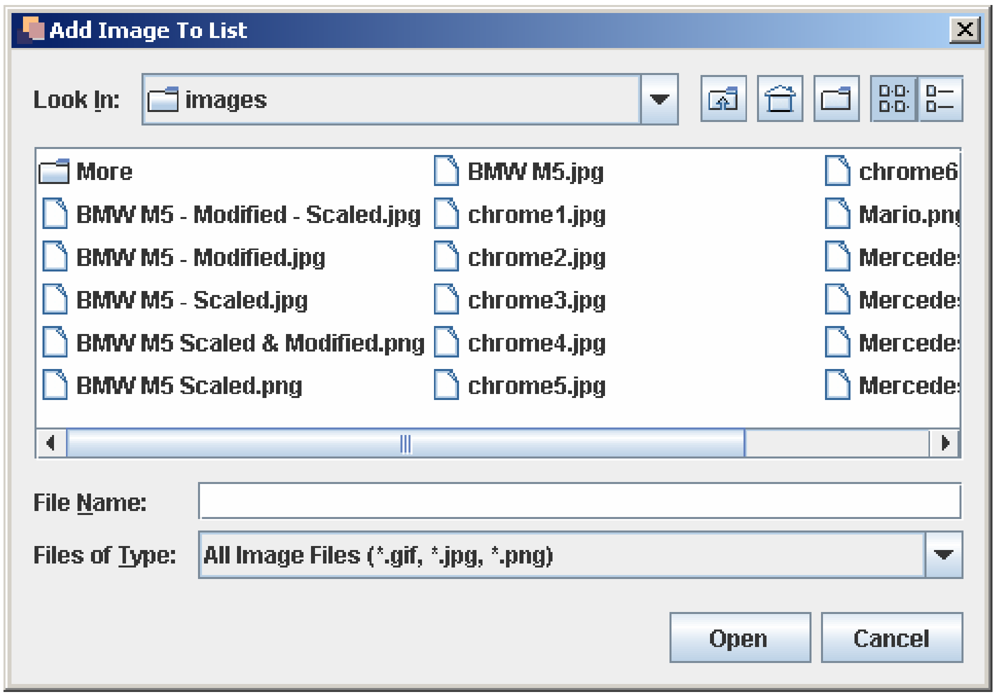
When the application starts, it will automatically insert all acceptable image files located in the “images” folder of the program directory. To insert additional images to be used as cover images in the watermarking process, choose one of the following methods:

(a) From the menu bar, go to Edit → Add Image.

(b) Right click on the left portion of the main application window and choose “Add Image” from the pop-up context menu.

(c) Click on the “Add Image” button (#4 from the component layout).

Regarding which method is used, the following dialog box will appear on the screen:


Add Image To List

Look In: images

More
BMW M5 - Modified - Scaled.jpg
BMW M5 - Modified.jpg
BMW M5 - Scaled.jpg
BMW M5 Scaled & Modified.png
BMW M5 Scaled.png
BMW M5.jpg
chrome1.jpg
chrome2.jpg
chrome3.jpg
chrome4.jpg
chrome5.jpg
chrome6
Mario.pn
Mercede
Mercede
Mercede
Mercede

File Name:

Files of Type: All Image Files (*.gif, *.jpg, *.png)

Navigate to the desired image file location, select the image, and click on “Open”. The image will be inserted and highlighted in the image list.

A.3 DELETING IMAGES FROM THE IMAGE LIST

To delete an image from the image list, select the image from list and choose one of the following methods:

(a) From the menu bar, go to Edit → Remove Image.

(b) Right click on the left portion of the main application window and choose "Remove Image" from the pop-up context menu.

(c) Click on the "Remove Image" button (#5 from the component layout).

A.4 CLEARING IMAGES FROM THE IMAGE LIST

To remove all the images that are currently present in the image list, choose one of the following methods:

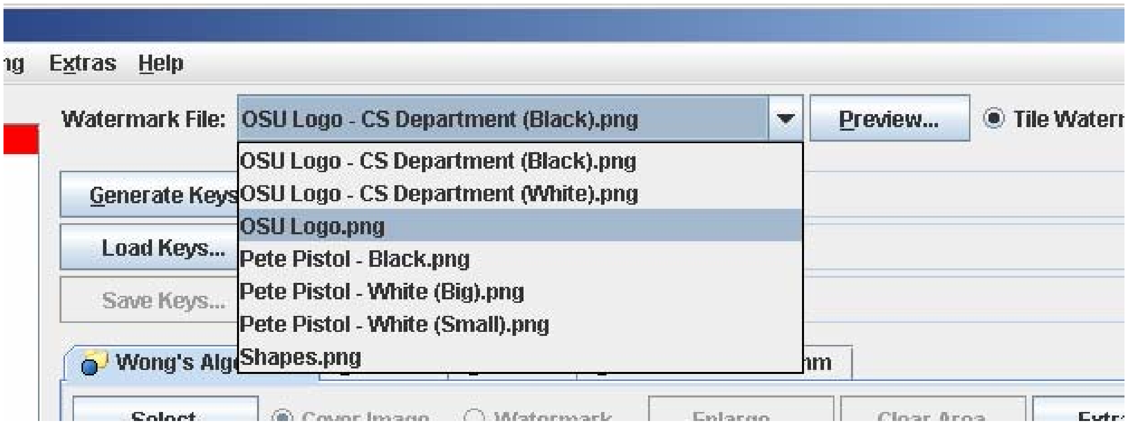
(a) From the menu bar, go to Edit → Clear List.

(b) Right click on the left portion of the main application window and choose "Clear List" from the pop-up context menu.

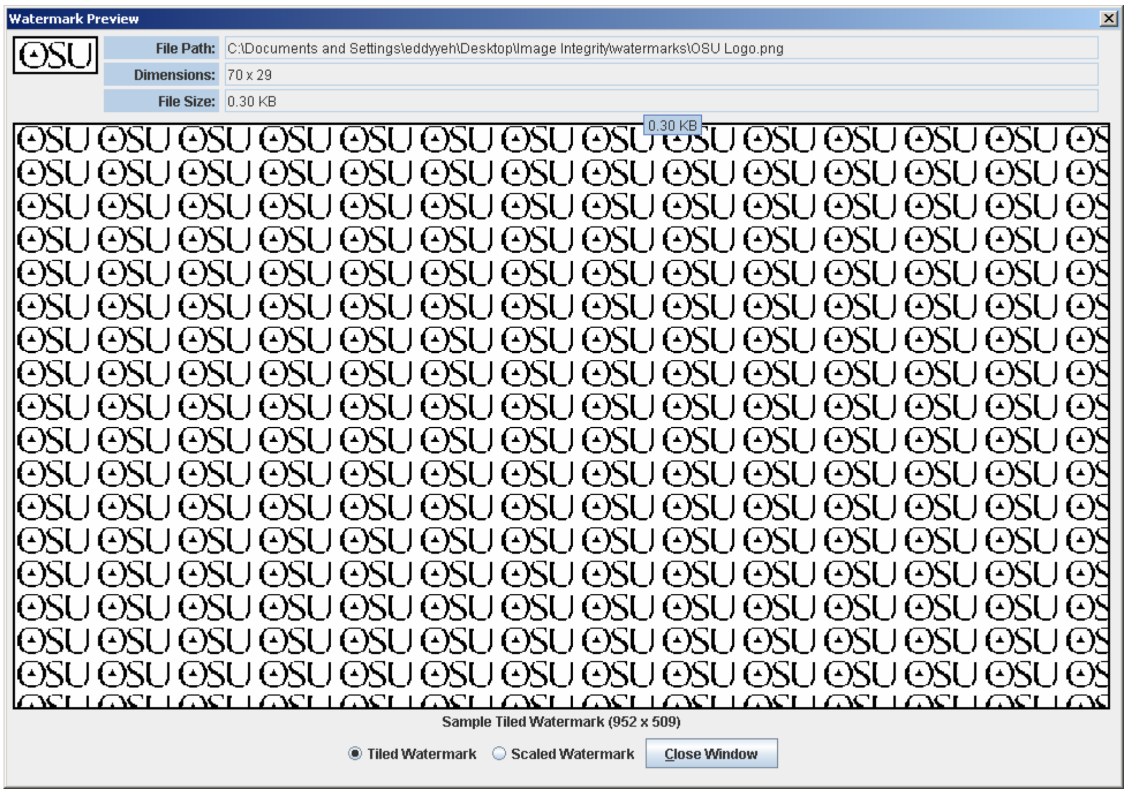
(c) Click on "Clear List" button (#6 from the component layout).

A.5 PREVIEWING A WATERMARK IMAGE

When the application starts, it will automatically insert all the bi-level (1-bit) watermark files located in the "watermarks" folder of the program directory. To preview a watermark, choose the watermark file name from the watermark combo box (#7 from the component layout) as shown below:


ng Extras Help
Watermark File: OSU Logo - CS Department (Black).png
Preview...
Tile Watern
Generate Keys
Load Keys...
Save Keys...
Wong's Alg
OSU Logo - CS Department (Black).png
OSU Logo - CS Department (White).png
OSU Logo.png
Pete Pistol - Black.png
Pete Pistol - White (Big).png
Pete Pistol - White (Small).png
Shapes.png

After choosing from the available watermarks, click on the watermark preview button located next to the watermark combo box (#8 from the component layout) and the following dialog will appear on the screen:


Watermark Preview

File Path: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\watermarks\OSU Logo.png

Dimensions: 70 x 29

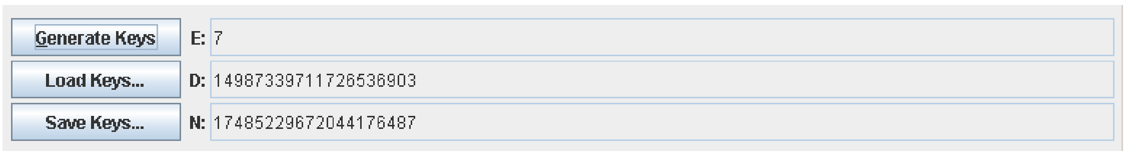
File Size: 0.30 KB

Sample Tiled Watermark (952 x 509)

By default, a tiled sample of the watermark will be shown. Users can switch between a tiled or scaled sample by choosing from the appropriate radio buttons located at the bottom of the dialog window. Click on “Close Window” button after finish viewing.

A.6 GENERATING RSA KEYS FOR WATERMARK INSERTION AND EXTRACTION

As described in this research, all the proposed algorithms require a 64-bit key for the watermark insertion process. To generate a random key, click on the generate keys button (#11 from the component layout). The newly generated key-pair will be displayed in the key information panel (#14 of the component layout) as shown below:

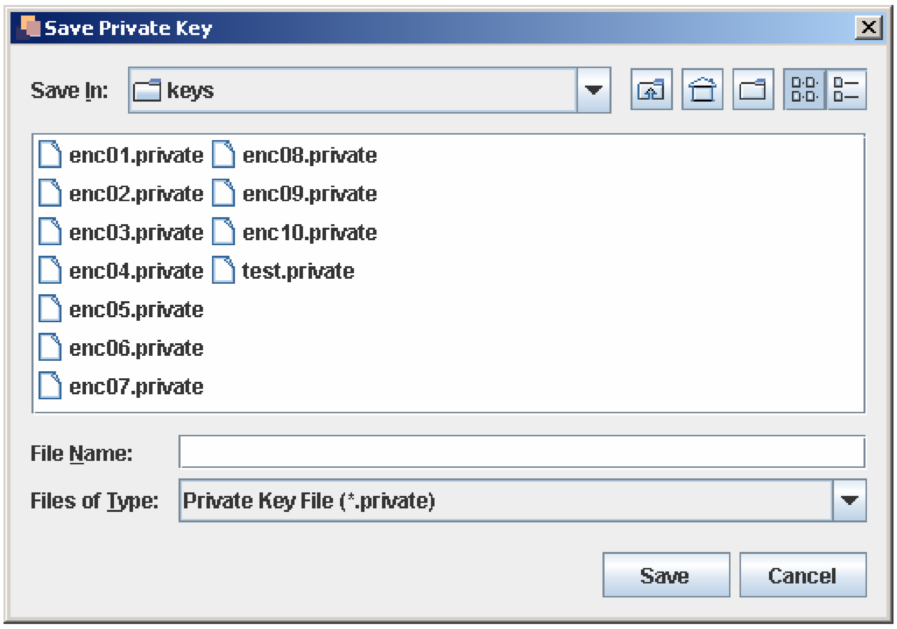


E: 7
D: 14987339711726536903
N: 17485229672044176487

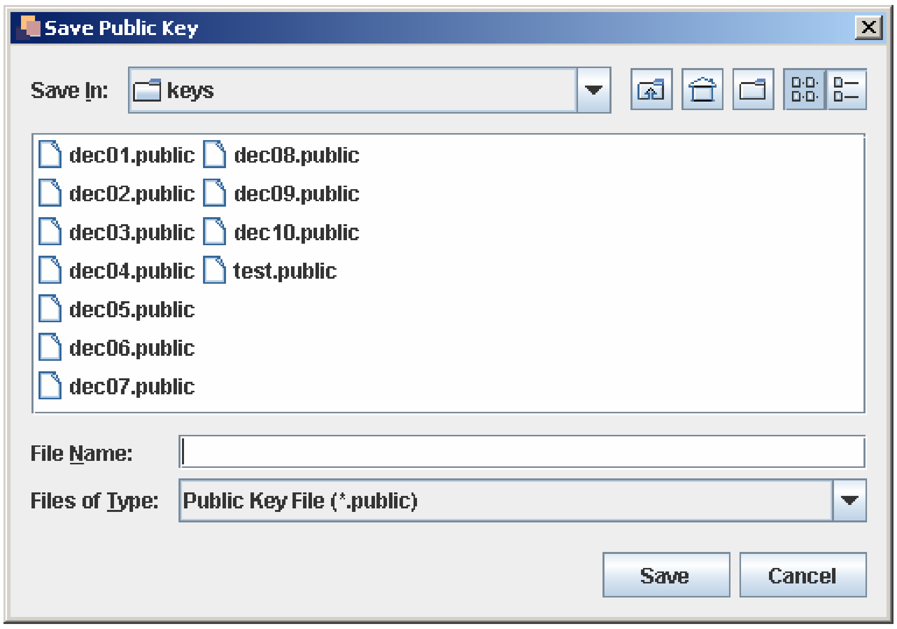
Note that the generate keys button will generate key-pairs for the insertion and extraction process. If a key-pair is used for watermark insertion, its respective decryption key-pair should be used for the extraction process. To save RSA keys, refer to “Saving RSA Key-Pairs” section.

A.7 SAVING RSA KEY-PAIRS FOR WATERMARK INSERTION AND EXTRACTION

After generating a RSA key-pair, users have the choice of saving the newly generated key for watermark extraction or even for future key reuse. To save a RSA key-pair, click on the save keys button (#13 from the component layout). The following dialog will appear on the screen:



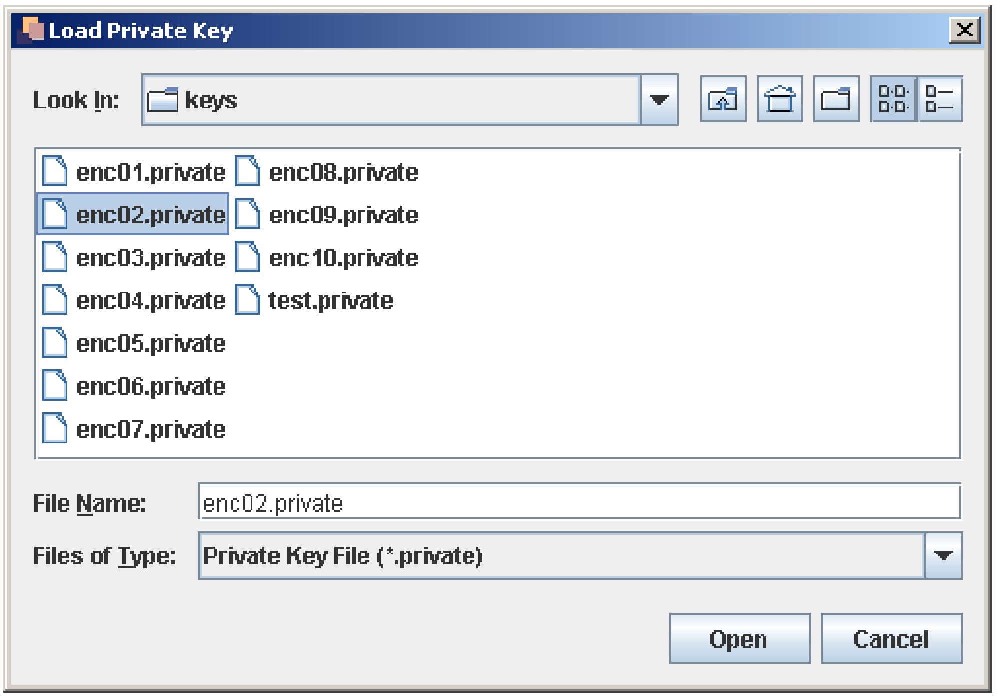
Navigate to the desired save directory and type the file name to be used to store the private key-pair and click on “Save”. A second dialog will appear on the screen as seen as follows:



Navigate to the desired save directory and type the file name to be used to store the public key-pair and click on “Save”. Both private and public key-pair will be saved.

A.8 LOADING PRIVATE KEY-PAIR FOR WATERMARK INSERTION

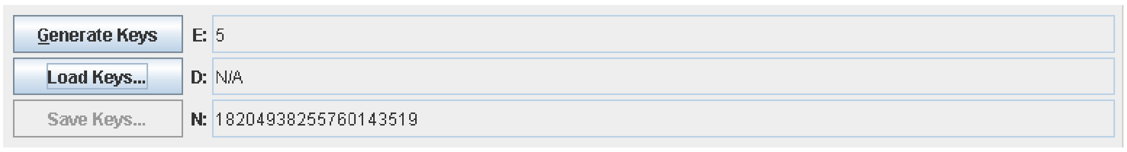
To load a previously saved private key, click on the load keys button (#12 from the component layout) and the following dialog will appear on the screen:



File Name: enc02.private

Files of Type: Private Key File (*.private)

Select the appropriate private key file to open and click on "Open". The application will read the contents of the specified file and update the key information panel as seen below:



E: 5

D: N/A

N: 18204938255760143519

A.9 INSERTING A WATERMARK IMAGE INTO A COVER IMAGE

To insert a watermark into a cover image, follow the following steps:

(1) Select the cover image from the image list (#1 from the component layout).

(2) Select the watermark image to be used for insertion from the watermark file combo box (#7 from the component layout).

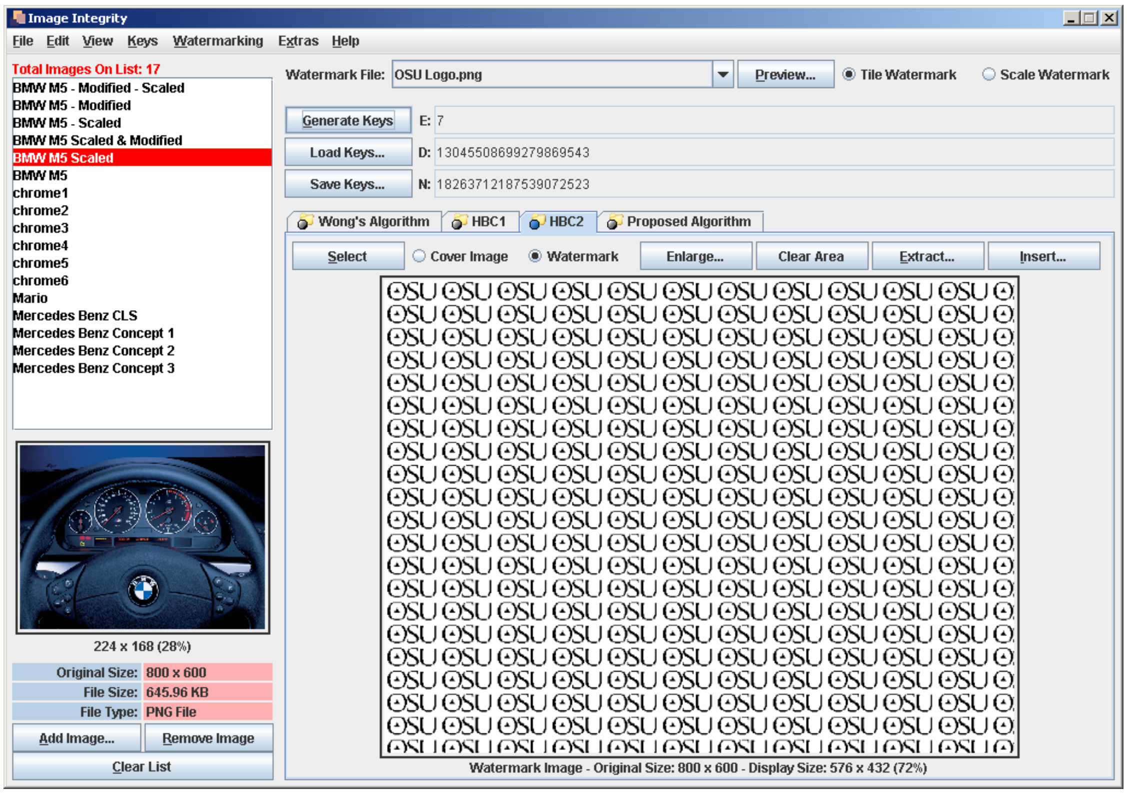
(3) Watermark images and cover images should be of the same width and height, select the resizing option (i.e. tiled or scaled) from the options located at the right of the preview watermark button.

(4) Generate or load a private key pair by clicking on the generate keys button (#11 from the component layout) or load keys button (#12 from the component layout) respectively.

(5) Select the appropriate watermarking insertion algorithm tab (#15 from the component layout).

(6) Click on the select image button (#16 from the component layout) to select the cover image and watermark image to be used in the insertion process.

After clicking the “Select Image” button, the selected image will be displayed in the area specified as 23 in the component layout. Users can clear the contents of the display area by clicking the clear display area button (#20 of the component layout). Also, users can choose which image to be displayed (cover image or watermark image) in the display area by changing the current displayed image option (#17 and #18 from the component layout) as shown below:



Total Images On List: 17

BMW M5 - Modified - Scaled
BMW M5 - Modified
BMW M5 - Scaled
BMW M5 Scaled & Modified
BMW M5 Scaled
BMW M5
chrome1
chrome2
chrome3
chrome4
chrome5
chrome6
Mario
Mercedes Benz CLS
Mercedes Benz Concept 1
Mercedes Benz Concept 2
Mercedes Benz Concept 3

224 x 168 (28%)

Original Size:	800 x 600
File Size:	645.96 KB
File Type:	PNG File

Watermark File: OSU Logo.png

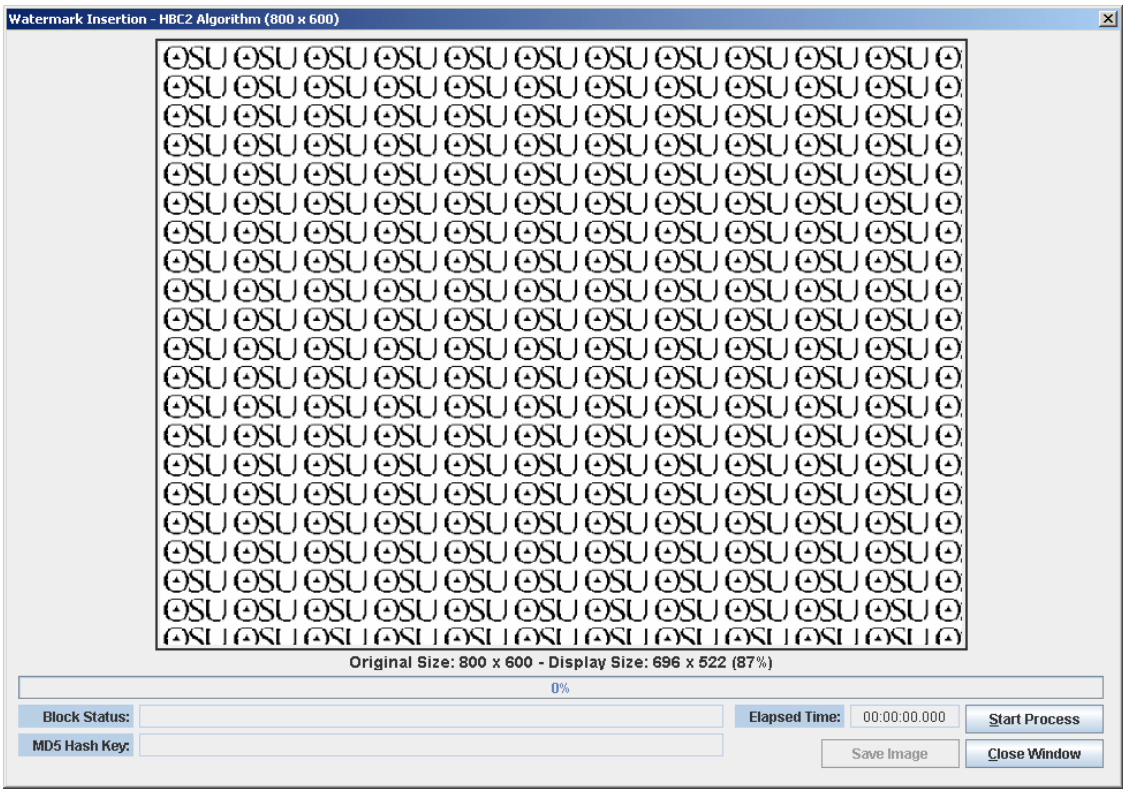
[✓ Tile Watermark] Scale Watermark

E: 7
D: 13045508699279869543
N: 18263712187539072523

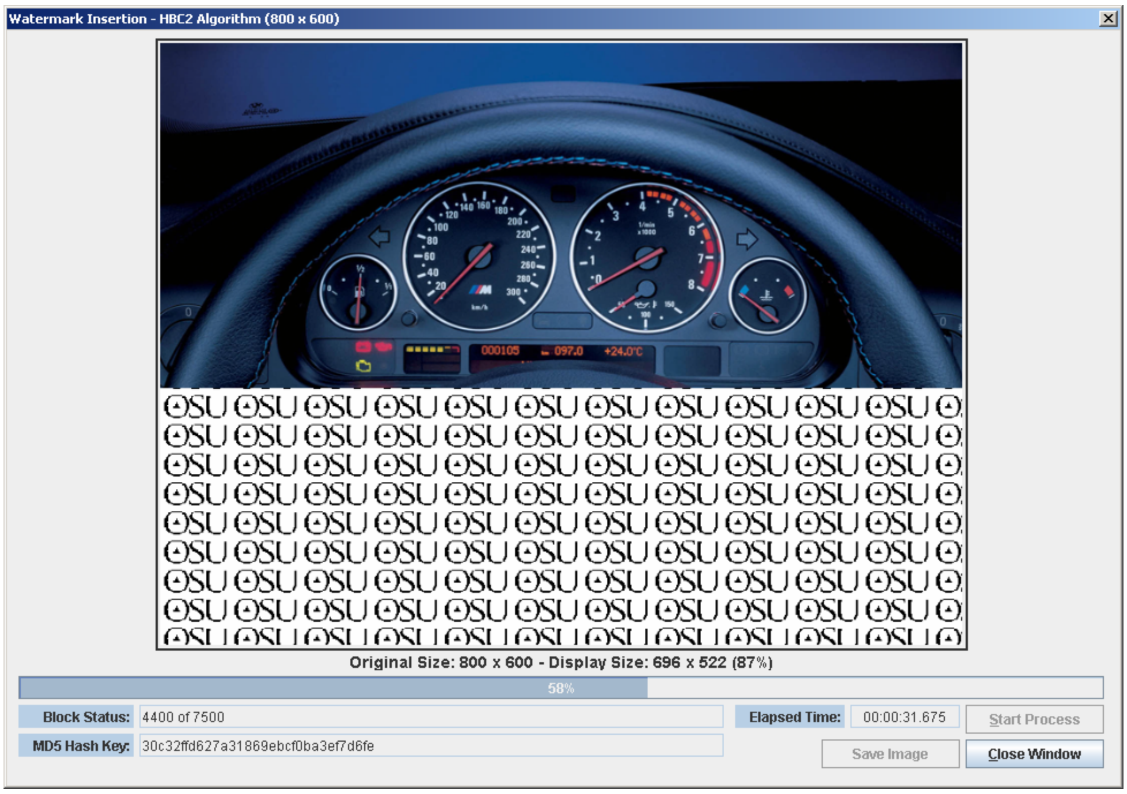
Cover Image [✓ Watermark]

Watermark Image - Original Size: 800 x 600 - Display Size: 576 x 432 (72%)

(7) After selecting the cover image and watermark image to be used, click on the insert button (#22 from the component layout). The following dialog will appear on the screen:


Watermark Insertion - HBC2 Algorithm (800 x 600)
Original Size: 800 x 600 - Display Size: 696 x 522 (87%)
0%
Block Status:
MD5 Hash Key:
Elapsed Time:
00:00:00.000
Start Process
Save Image
Close Window

(8) To start the watermark insertion process, click on "Start Process" button. The application will proceed to insert the watermark and update the center image once every row is completed as seen below:



Original Size: 800 x 600 - Display Size: 696 x 522 (87%)

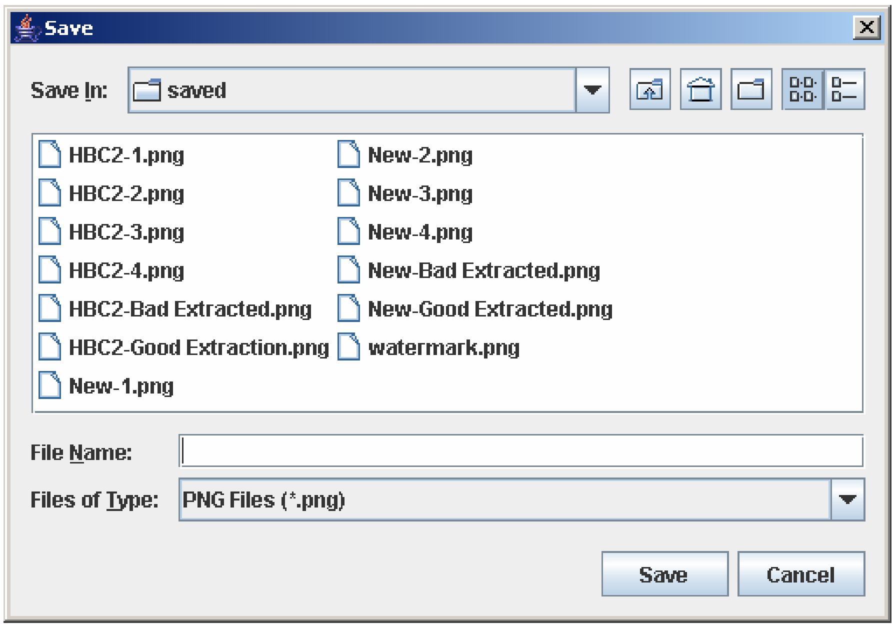
58%

Block Status: 4400 of 7500

Elapsed Time: 00:00:31.675

MD5 Hash Key: 30c32ffd627a31869ebcf0ba3ef7d6fe

(9) After the application finishes inserting the watermark image, users can save the watermarked image by clicking the save image button. The following dialog will appear on the screen:



Save In: saved

HBC2-1.png
HBC2-2.png
HBC2-3.png
HBC2-4.png
HBC2-Bad Extracted.png
HBC2-Good Extraction.png
New-1.png
New-2.png
New-3.png
New-4.png
New-Bad Extracted.png
New-Good Extracted.png
watermark.png

File Name:

Files of Type: PNG Files (*.png)

(10) Navigate to the desired save directory and type in the file name to be used as the watermarked image and click on save.

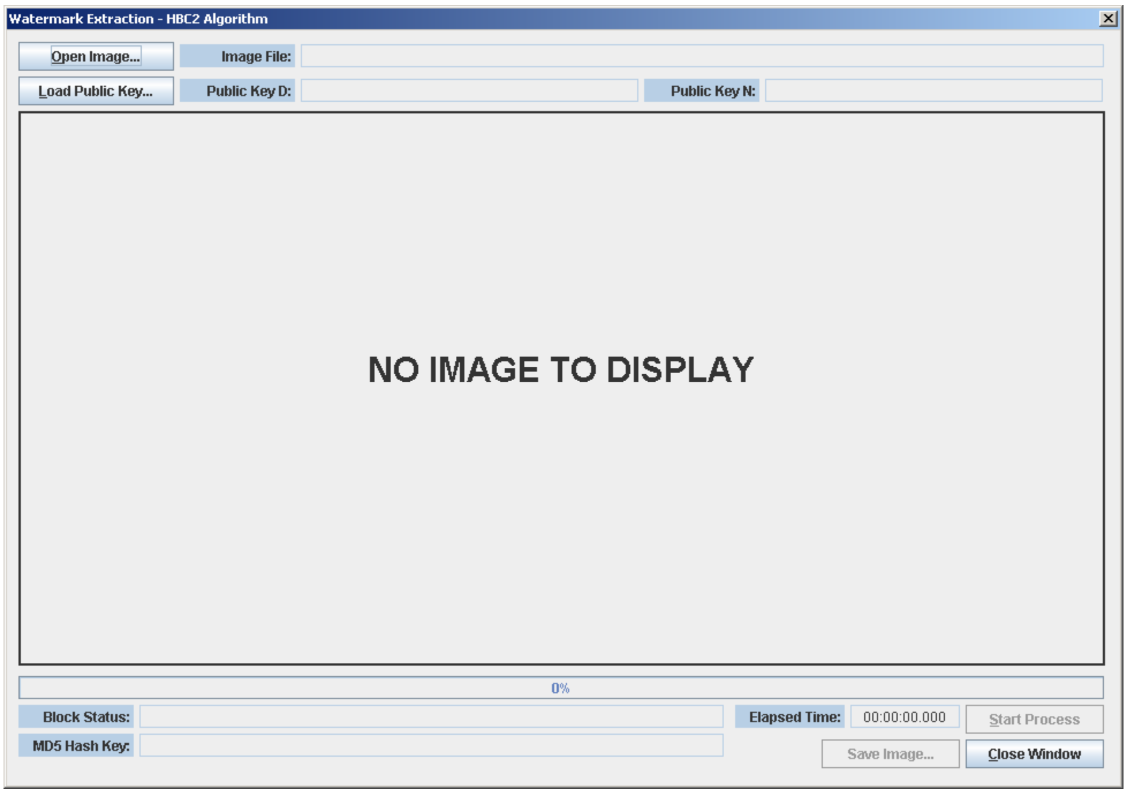
(11) Click on close window to close the watermark insertion dialog.

A.10 EXTRACTING A WATERMARK IMAGE FROM A WATERMARKED IMAGE

To extract the embedded watermark image from a watermarked image, follow the following steps:

(1) Select the appropriate watermarking algorithm from the algorithm selection tabs (#15 from the component layout).

(2) Click on the extract button (#21 from the component layout), the following dialog will appear on the screen:



(3) Click on “Open Image” and select the watermarked image and click on “Open”.

(4) Click on “Load Public Key” and load the public key-pair used in the watermark insertion process. After the watermarked image and its respective keys are loaded, the screen will look as follows:

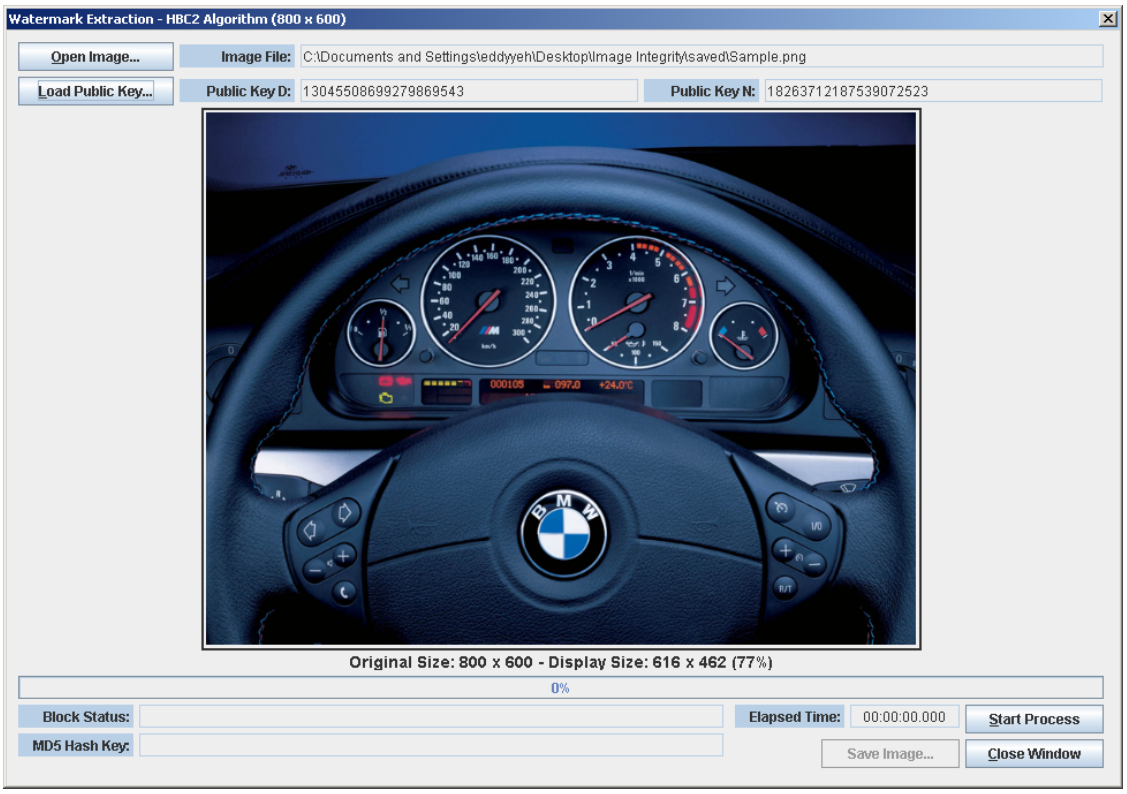

Watermark Extraction - HBC2 Algorithm (800 x 600)

Image File: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\saved\Sample.png

Public Key D: 13045508699279869543

Public Key N: 18263712187539072523

Original Size: 800 x 600 - Display Size: 616 x 462 (77%)

0%

Block Status:

MD5 Hash Key:

Elapsed Time: 00:00:00.000

(5) To start the watermark extraction process, click on “Start Process” button. The application will proceed to extract the watermark and update the center image once every row is completed as seen below:

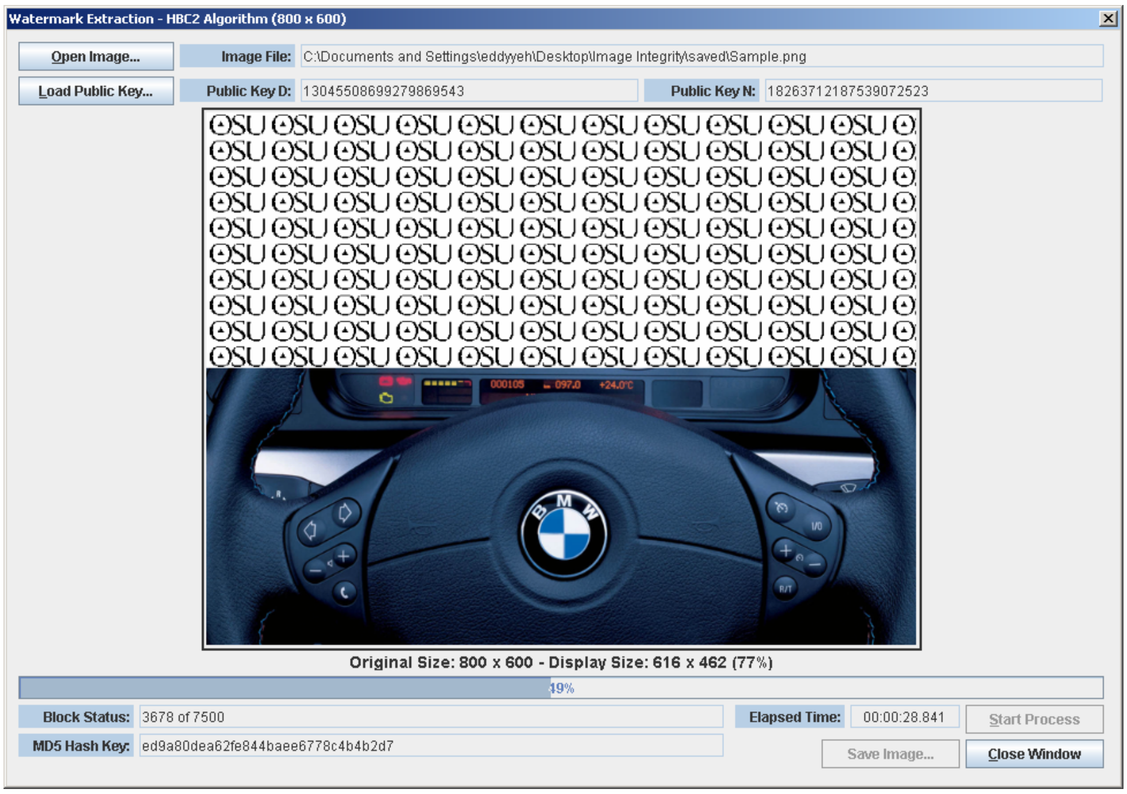

Watermark Extraction - HBC2 Algorithm (800 x 600)

Image File: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\saved\Sample.png

Public Key D: 13045508699279869543

Public Key N: 18263712187539072523

Original Size: 800 x 600 - Display Size: 616 x 462 (77%)

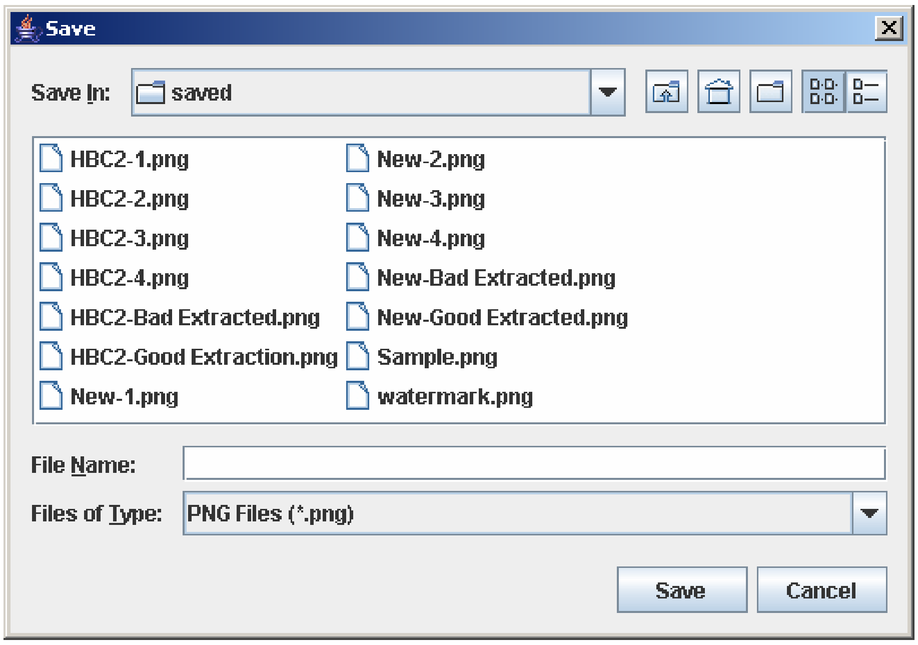
49%

Block Status: 3678 of 7500

Elapsed Time: 00:00:28.841

MD5 Hash Key: ed9a80dea62fe844baee6778c4b4b2d7

(6) After the application finishes extracting the watermark image, users can save the watermark by clicking the save image button. The following dialog will appear on the screen:


Save

Save In: saved

HBC2-1.png	New-2.png
HBC2-2.png	New-3.png
HBC2-3.png	New-4.png
HBC2-4.png	New-Bad Extracted.png
HBC2-Bad Extracted.png	New-Good Extracted.png
HBC2-Good Extraction.png	Sample.png
New-1.png	watermark.png

File Name:

Files of Type: PNG Files (*.png)

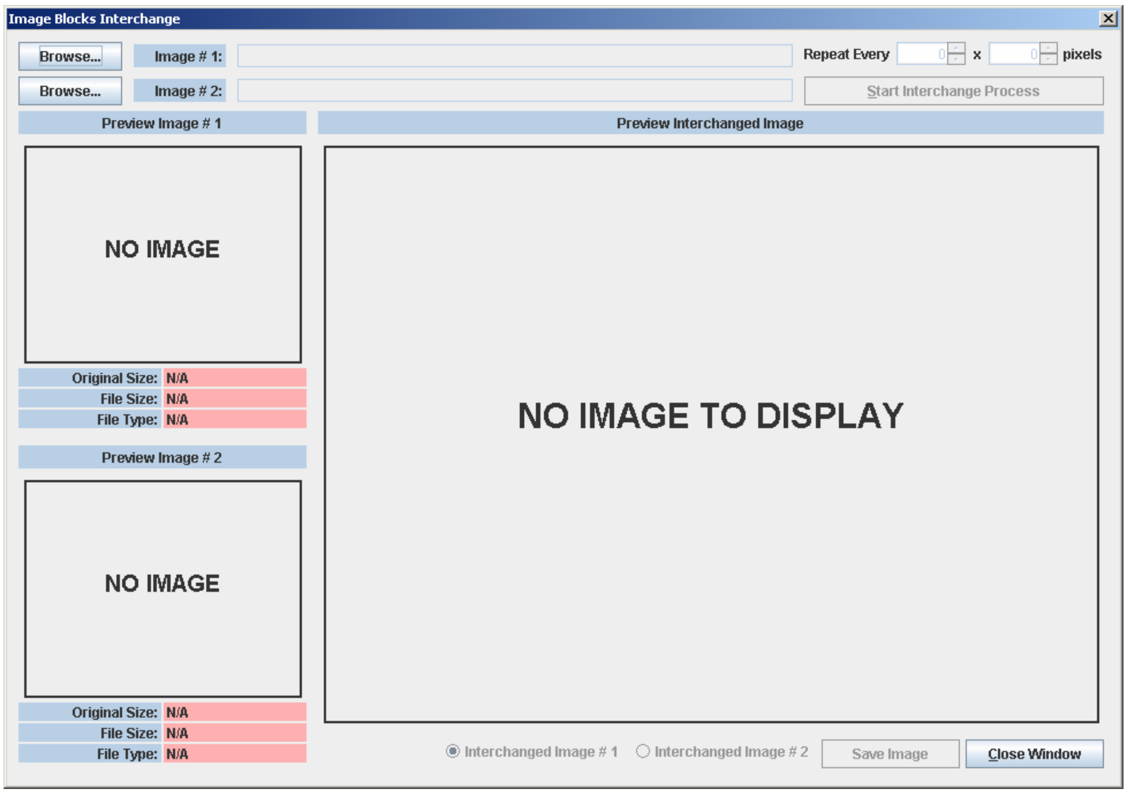
(7) Navigate to the desired save location, type the file name to be used to save the watermark image and clock on the “Save” button.

(8) Click on “Close Window” to close the watermark extraction dialog.

A.11 INTERCHANGING BLOCKS FROM TWO WATERMARKED IMAGES

To interchange blocks of a specific width and height, follow the steps below:

(1) From the menu bar, go to Extras → Interchange Blocks. The following dialog will appear on the screen:



(2) Use the “Browse” buttons to select the first and second watermarked images to be interchanged.

(3) Select the interchange block width and height from the spinner buttons located on the top-left portion of the dialog.

(4) Click on “Start Interchange Process” to start interchanging blocks as seen below:

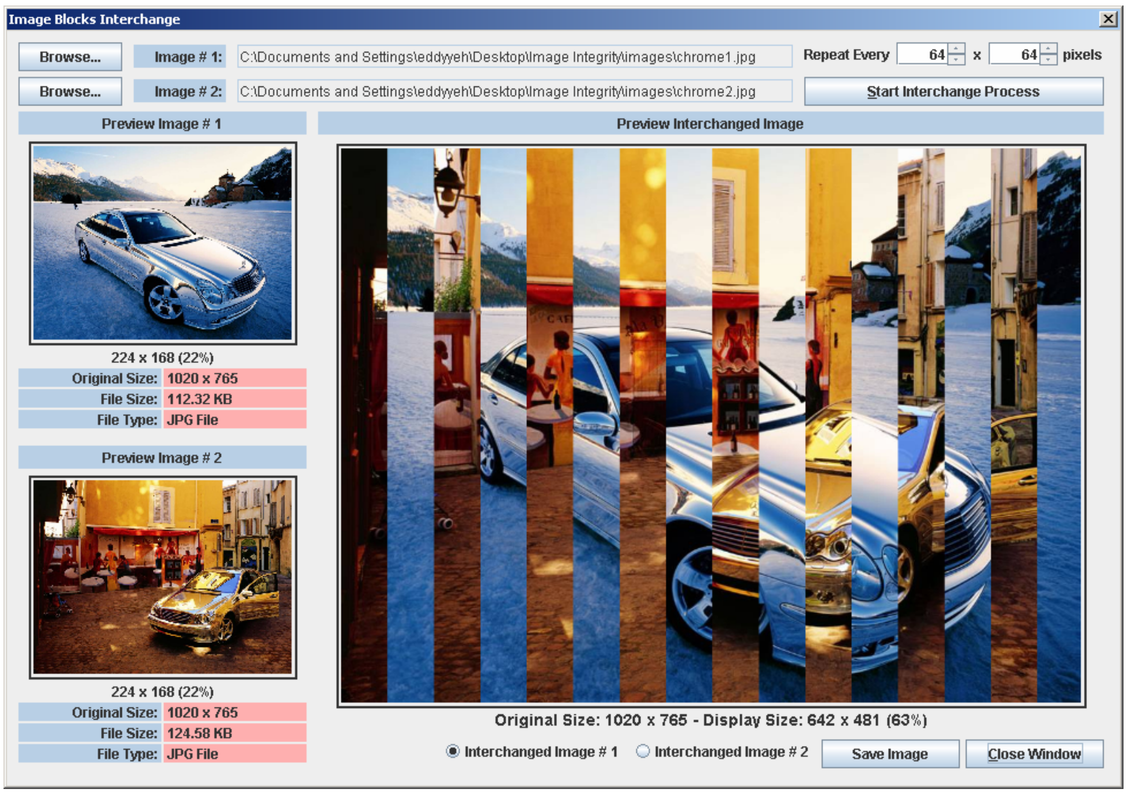


Image # 1: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\images\chrome1.jpg

Image # 2: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\images\chrome2.jpg

Repeat Every 64 x 64 pixels

224 x 168 (22%)

Original Size:	1020 x 765
File Size:	112.32 KB
File Type:	JPG File

224 x 168 (22%)

Original Size:	1020 x 765
File Size:	124.58 KB
File Type:	JPG File

Original Size: 1020 x 765 - Display Size: 642 x 481 (63%)

[✓ Interchanged Image # 1] Interchanged Image # 2

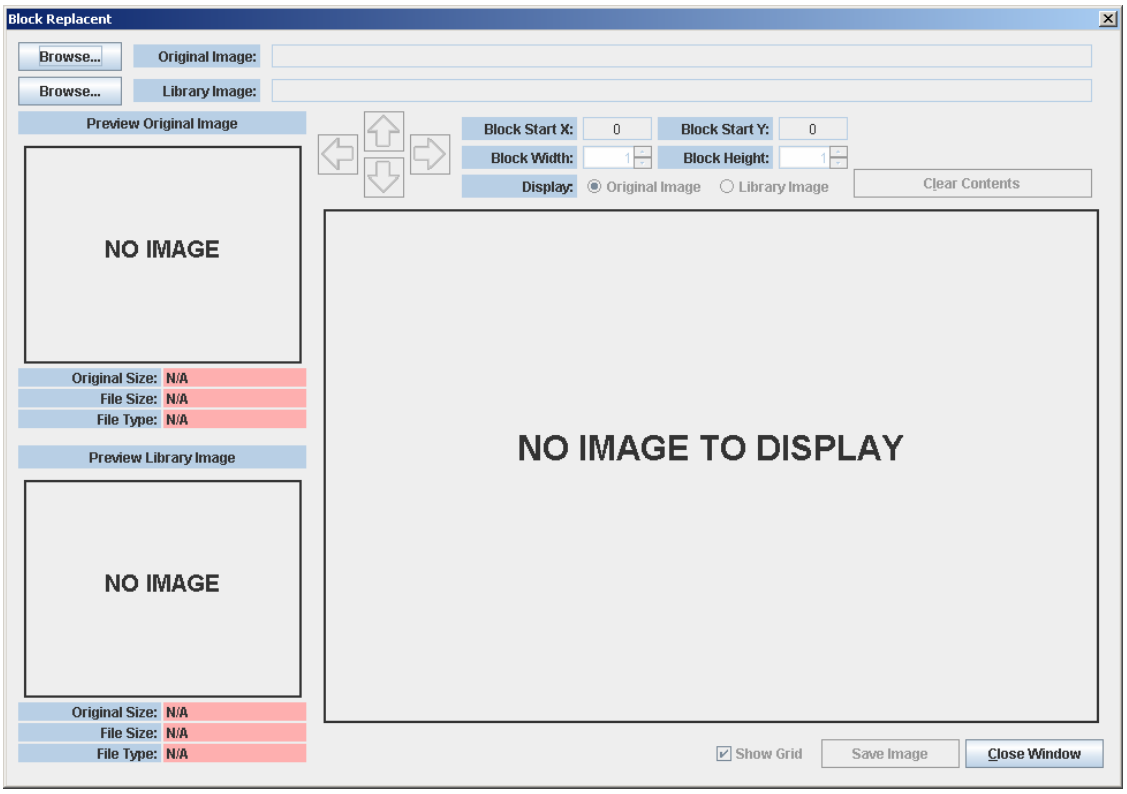
(5) After the application finishes interchanging the image blocks, choose the appropriate version of the image to save and click on “Save Image” button.

(6) Click on “Close Window” to close the block interchange dialog.

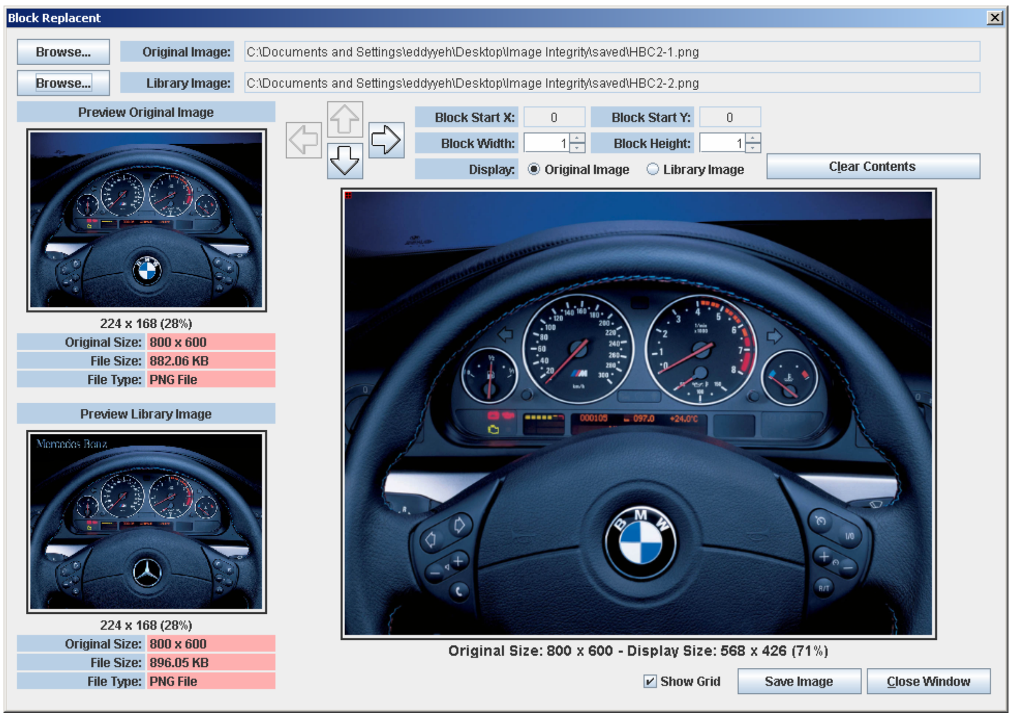
A.12 REPLACING A SPECIFIED IMAGE BLOCK FROM TWO WATERMARKED IMAGES

To replace a specified image block from two watermarked images, follow the steps below:

(1) From the menu bar, go to Extras → Block Replacement. The following dialog will appear on the screen:



(2) Use the "Browse" buttons to select the original and library image respectively. After selecting two different images, the dialog will look as follows:


Block Replacent

Browse... Original Image: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\saved\HBC2-1.png

Browse... Library Image: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\saved\HBC2-2.png

Preview Original Image

224 x 168 (28%)

Original Size:	800 x 600
File Size:	882.06 KB
File Type:	PNG File

Preview Library Image

224 x 168 (28%)

Original Size:	800 x 600
File Size:	896.05 KB
File Type:	PNG File

Block Start X: 0 Block Start Y: 0

Block Width: 1 Block Height: 1

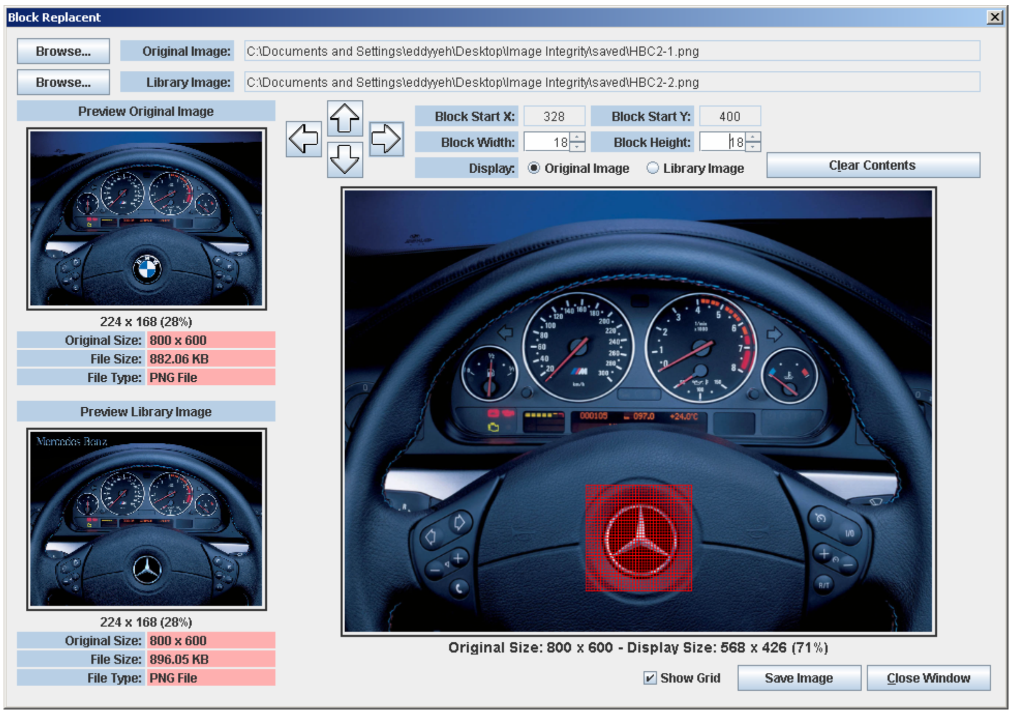
Display: [✓ Original Image] Library Image

Clear Contents

Original Size: 800 x 600 - Display Size: 568 x 426 (71%)

✓ Show Grid Save Image Close Window

(3) Select the block width and height to be replaced and use the up, down, left, and right buttons to select the location in which the block replacement will take place.


Block Replacent

Browse... Original Image: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\saved\HBC2-1.png

Browse... Library Image: C:\Documents and Settings\eddyyeh\Desktop\Image Integrity\saved\HBC2-2.png

Preview Original Image

224 x 168 (28%)

Original Size:	800 x 600
File Size:	882.06 KB
File Type:	PNG File

Preview Library Image

224 x 168 (28%)

Original Size:	800 x 600
File Size:	896.05 KB
File Type:	PNG File

Block Start X: 328 Block Start Y: 400

Block Width: 18 Block Height: 18

Display: [✓ Original Image] Library Image

Clear Contents

Original Size: 800 x 600 - Display Size: 568 x 426 (71%)

✓ Show Grid Save Image Close Window

(4) Once the block replacement region is selected, select the display image option and click on “Save Image” to save the block replaced image.

(5) Click on “Close Window” to close the block replacement dialog.

APPENDIX B

APPLICATION SOURCE CODE

B.1 ImageIntegrity.java

```
// ImageIntegrity.java
//
// ImageIntegrity is the main Graphical User Interface window for the
// application.

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.util.*;
import java.io.*;
import java.math.*;

public class ImageIntegrity extends JFrame
   implements ActionListener, ListSelectionListener, ChangeListener,
              MouseListener{

   private static final String DEFAULT_IMAGE_DIR = "images/";
   private static final String DEFAULT_WATERMARK_DIR = "watermarks/";
   private static final String DEFAULT_KEYS_DIR = "keys/";
   private static final String APP_ICON = "icons/app.gif";
   private static final String TAB_ICON = "icons/tab.gif";
   private static final String OFF_ICON = "icons/off.gif";

   protected JMenuItem exitMenuItem, addMenuItem, removeMenuItem,
                       clearListMenuItem, generateMenuItem, loadKeysMenuItem,
                       saveKeysMenuItem, clearKeysMenuItem, previewMenuItem,
                       interchangeMenuItem, blockReplacementMenuItem,
                       aboutMenuItem;
   protected JMenuItem pAddMenuItem, pRemoveMenuItem, pClearListMenuItem,
                       pGenerateMenuItem, pLoadKeysMenuItem, pSaveKeysMenuItem,
                       pClearKeysMenuItem, pPreviewMenuItem;
   protected JCheckBoxMenuItem imageListPanelMenuItem, watermarkPanelMenuItem,
                               keysPanelMenuItem;
   protected JCheckBoxMenuItem lImageListPanelMenuItem, lWatermarkPanelMenuItem,
                               lKeysPanelMenuItem;
   protected JCheckBoxMenuItem rImageListPanelMenuItem, rWatermarkPanelMenuItem,
                               rKeysPanelMenuItem;
   protected JPopupMenu imageListPanelPopupMenu, watermarkingPanelPopupMenu;
   protected JLabel imageCountLabel;
   protected JList imageList;
   protected JButton addButton, removeButton, clearButton, previewButton;
   protected JButton generateButton, loadKeyButton, saveKeyButton;
   protected JRadioButton tiledRadioButton, scaledRadioButton;
   protected JTextField dTextField, eTextField, nTextField;
   protected JComboBox watermarkComboBox;
   protected JPanel imageListPanel, watermarkPanel, keysPanel;
protected JTabbedPane insertionTabs;
protected ThumbPanel thumbPanel;
protected DisplayArea Wong, HBC1, HBC2, proposedAlgorithm;
protected BigInteger e, d, n;
protected Vector<ImageInfo> imageListVector;
protected Vector<ImageInfo> watermarkListVector;

public ImageIntegrity(){
   super("Image Integrity");
   GraphicsEnvironment env = GraphicsEnvironment.
      getLocalGraphicsEnvironment();

   Rectangle desktopSize = env.getMaximumWindowBounds();

   this.getContentPane().setLayout(new BorderLayout());
   this.setSize((int)desktopSize.getWidth(), (int)desktopSize.getHeight());
   this.setResizable(false);
   this.setIconImage(this.getIcon(APP_ICON).getImage());
   this.setJMenuBar(initializeMenuBar());
   this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

   imageListPanel = initializeImageListPanel();
   watermarkPanel = initializeWatermarkPanel();
   keysPanel = initializeKeyPanel();
   imageListPanelPopupMenu = initializeImageListPanelPopupMenu();
   watermarkingPanelPopupMenu = initializeWatermarkingPanelPopupMenu();

   Wong = new WongAlgorithm(this);
   HBC1 = new HBC1Algorithm(this);
   HBC2 = new HBC2Algorithm(this);
   proposedAlgorithm = new ProposedAlgorithm(this);

   insertionTabs = new JTabbedPane();
   insertionTabs.addTab("Wong's Algorithm", this.getIcon(TAB_ICON), Wong,
                     "Original Wong's Public/Private Key Watermarking " +
                     "scheme");
   insertionTabs.addTab("HBC1", this.getIcon(OFF_ICON), HBC1,
                     "Hash Block Chaining 1");
   insertionTabs.addTab("HBC2", this.getIcon(OFF_ICON), HBC2,
                     "Hash Block Chaining 2");
   insertionTabs.addTab("Proposed Algorithm", this.getIcon(OFF_ICON),
                     proposedAlgorithm, "Proposed Algorithm");
   insertionTabs.setBorder(new EmptyBorder(5, 5, 5, 5));

   JPanel topPanel = new JPanel();
   topPanel.setLayout(new BorderLayout(3, 3));
   topPanel.add(watermarkPanel, BorderLayout.NORTH);
   topPanel.add(keysPanel, BorderLayout.SOUTH);

   JPanel rightPanel = new JPanel();
   rightPanel.setLayout(new BorderLayout());
   rightPanel.add(topPanel, BorderLayout.NORTH);
   rightPanel.add(insertionTabs, BorderLayout.CENTER);

   this.getContentPane().add(imageListPanel, BorderLayout.WEST);
   this.getContentPane().add(rightPanel, BorderLayout.CENTER);

   if (imageListVector.size() != 0){
      imageList.setSelectedIndex(0);
      thumbPanel.setImageInfo((ImageInfo)imageList.getSelectedValue());
   } // End of if statement
   else{
      removeButton.setEnabled(false);
      clearButton.setEnabled(false);
      removeMenuItem.setEnabled(false);
      pRemoveMenuItem.setEnabled(false);
      clearListMenuItem.setEnabled(false);
      pClearListMenuItem.setEnabled(false);
   } // End of else statement

   if (this.canSelect()){
      Wong.enableSelectButton();
      HBC1.enableSelectButton();
      HBC2.enableSelectButton();
      proposedAlgorithm.enableSelectButton();
   } // End of if statement
   else{
      Wong.disableSelectButton();
      HBC1.disableSelectButton();
      HBC2.disableSelectButton();
      proposedAlgorithm.disableSelectButton();
} // End of else statement

if (watermarkListVector.size() == 0){
   watermarkComboBox.setEnabled(false);
   previewButton.setEnabled(false);
   previewMenuItem.setEnabled(false);
   pPreviewMenuItem.setEnabled(false);
   tiledRadioButton.setEnabled(false);
   scaledRadioButton.setEnabled(false);
} // End of if statement

saveKeyButton.setEnabled(false);
saveKeysMenuItem.setEnabled(false);
pSaveKeysMenuItem.setEnabled(false);
clearKeysMenuItem.setEnabled(false);
pClearKeysMenuItem.setEnabled(false);

// Registering component listeners
addButton.addActionListener(this);
removeButton.addActionListener(this);
clearButton.addActionListener(this);
imageList.addMouseListener(this);
imageList.addListSelectionListener(this);
thumbPanel.addMouseListener(this);
addButton.addMouseListener(this);
removeButton.addMouseListener(this);
clearButton.addMouseListener(this);
previewButton.addActionListener(this);
previewButton.addMouseListener(this);
watermarkComboBox.addMouseListener(this);
tiledRadioButton.addMouseListener(this);
scaledRadioButton.addMouseListener(this);
generateButton.addActionListener(this);
loadKeyButton.addActionListener(this);
saveKeyButton.addActionListener(this);
generateButton.addMouseListener(this);
loadKeyButton.addMouseListener(this);
saveKeyButton.addMouseListener(this);
dTextField.addMouseListener(this);
eTextField.addMouseListener(this);
nTextField.addMouseListener(this);
exitMenuItem.addActionListener(this);
addMenuItem.addActionListener(this);
removeMenuItem.addActionListener(this);
clearListMenuItem.addActionListener(this);
generateMenuItem.addActionListener(this);
loadKeysMenuItem.addActionListener(this);
saveKeysMenuItem.addActionListener(this);
clearKeysMenuItem.addActionListener(this);
previewMenuItem.addActionListener(this);
blockReplacementMenuItem.addActionListener(this);
interchangeMenuItem.addActionListener(this);
aboutMenuItem.addActionListener(this);
imageListPanelMenuItem.addActionListener(this);
watermarkPanelMenuItem.addActionListener(this);
keysPanelMenuItem.addActionListener(this);
pAddMenuItem.addActionListener(this);
pRemoveMenuItem.addActionListener(this);
pClearListMenuItem.addActionListener(this);
lImageListPanelMenuItem.addActionListener(this);
lWatermarkPanelMenuItem.addActionListener(this);
lKeysPanelMenuItem.addActionListener(this);
pGenerateMenuItem.addActionListener(this);
pLoadKeysMenuItem.addActionListener(this);
pSaveKeysMenuItem.addActionListener(this);
pClearKeysMenuItem.addActionListener(this);
pPreviewMenuItem.addActionListener(this);
rImageListPanelMenuItem.addActionListener(this);
rWatermarkPanelMenuItem.addActionListener(this);
rKeysPanelMenuItem.addActionListener(this);
insertionTabs.addChangeListener(this);
insertionTabs.addMouseListener(this);
imageListPanel.addMouseListener(this);
watermarkPanel.addMouseListener(this);
keysPanel.addMouseListener(this);
topPanel.addMouseListener(this);
rightPanel.addMouseListener(this);
Wong.addMouseListener(this);
HBC1.addMouseListener(this);
HBC2.addMouseListener(this);
proposedAlgorithm.addMouseListener(this);
```

```
   this.setVisible(true);        // Displaying window
} // End of public ImageIntegrity()

// Initializes the components in the image list panel
private JPanel initializeImageListPanel(){
   imageListVector = new Vector<ImageInfo>();

   // Reading images stored in the default images dir
   if (new File(DEFAULT_IMAGE_DIR).exists()){
      File imageFiles[] = new File(DEFAULT_IMAGE_DIR).listFiles();

      for (int i = 0; i < imageFiles.length; i++){
         if (ImageFilter.acceptFile(imageFiles[i])){
            ImageInfo tmp = new ImageInfo(imageFiles[i].getAbsolutePath());
            imageListVector.add(tmp);
         } // end of if statement
      } // End of for loop
   } // End of if statement

   imageCountLabel = new JLabel("Total Images On List: " +
                                imageListVector.size());
   imageCountLabel.setForeground(Color.red);
   // Image list settings
   imageList = new JList();
   imageList.setListData(imageListVector);
   imageList.setCellRenderer(new ImageListCellRenderer());
   imageList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
   imageList.setVisibleRowCount(20);
   JScrollPane imageListScroller = new JScrollPane(imageList);

   JPanel listPanel = new JPanel();
   listPanel.setLayout(new BorderLayout());
   listPanel.add(imageCountLabel, BorderLayout.NORTH);
   listPanel.add(imageListScroller, BorderLayout.SOUTH);

   // Thumbnail panel settings
   thumbPanel = new ThumbPanel();

   // Buttons settings
   addButton = new JButton("Add Image...");
   addButton.setMnemonic(KeyEvent.VK_A);
   addButton.setToolTipText("Adds an image to the list");

   removeButton = new JButton("Remove Image");
   removeButton.setMnemonic(KeyEvent.VK_R);
   removeButton.setToolTipText("Removes an image from the list");

   clearButton = new JButton("Clear List");
   clearButton.setMnemonic(KeyEvent.VK_C);
   clearButton.setToolTipText("Clear the contents of the image list");

   JPanel buttonsPanelTop = new JPanel();
   buttonsPanelTop.setLayout(new GridLayout(1, 2, 3, 3));
   buttonsPanelTop.add(addButton);
   buttonsPanelTop.add(removeButton);

   JPanel buttonsPanel = new JPanel();
   buttonsPanel.setLayout(new BorderLayout());
   buttonsPanel.add(buttonsPanelTop, BorderLayout.NORTH);
   buttonsPanel.add(clearButton, BorderLayout.SOUTH);

   JPanel imageListPanel = new JPanel();
   imageListPanel.setLayout(new BorderLayout(3, 3));
   imageListPanel.add(listPanel, BorderLayout.NORTH);
   imageListPanel.add(thumbPanel, BorderLayout.CENTER);
   imageListPanel.add(buttonsPanel, BorderLayout.SOUTH);
   imageListPanel.setBorder(new EmptyBorder(5, 5, 5, 5));

   return imageListPanel;
} // End of private JPanel initializeImageListPanel()

// Initialize the components of the watermark panel
private JPanel initializeWatermarkPanel(){
   JLabel waterLabel = new JLabel("Watermark File: ", JLabel.RIGHT);

   watermarkComboBox = new JComboBox();
   watermarkListVector = new Vector<ImageInfo>();

   // Reading watermarks stored in the default watermark dir
   if (new File(DEFAULT_WATERMARK_DIR).exists()){
      File imageFiles[] = new File(DEFAULT_WATERMARK_DIR).listFiles();

      for (int i = 0; i < imageFiles.length; i++){
         if (ImageFilter.acceptFile(imageFiles[i])){
            ImageInfo info = new ImageInfo(imageFiles[i].getAbsolutePath());
            watermarkListVector.add(info);
            watermarkComboBox.addItem(info.getFileName());
         } // end of if statement
      } // End of for loop
   } // End of if statement

   previewButton = new JButton("Preview...");
   previewButton.setMnemonic(KeyEvent.VK_P);
   previewButton.setToolTipText("Preview the current selected watermark");

   tiledRadioButton = new JRadioButton("Tile Watermark", true);
   tiledRadioButton.setToolTipText("Tiled watermark option");
   scaledRadioButton = new JRadioButton("Scale Watermark");
   scaledRadioButton.setToolTipText("Scaled watermark option");
   ButtonGroup group = new ButtonGroup();
   group.add(tiledRadioButton);
   group.add(scaledRadioButton);

   JPanel comboPanel = new JPanel();
   comboPanel.setLayout(new BorderLayout(3, 3));
   comboPanel.add(waterLabel, BorderLayout.WEST);
   comboPanel.add(watermarkComboBox, BorderLayout.CENTER);
   comboPanel.add(previewButton, BorderLayout.EAST);

   JPanel radioPanel = new JPanel();
   radioPanel.setLayout(new GridLayout(1, 2, 3, 3));
   radioPanel.add(tiledRadioButton);
   radioPanel.add(scaledRadioButton);

   JPanel watermarkPanel = new JPanel();
   watermarkPanel.setLayout(new BorderLayout(3, 3));
   watermarkPanel.add(comboPanel, BorderLayout.CENTER);
   watermarkPanel.add(radioPanel, BorderLayout.EAST);
   watermarkPanel.setBorder(new EmptyBorder(5, 5, 5, 5));

   return watermarkPanel;
} // End of private JPanel initializeWatermarkPanel()

// Initialize the components of the keys panel
private JPanel initializeKeyPanel(){
   generateButton = new JButton("Generate Keys");
   generateButton.setMnemonic(KeyEvent.VK_G);
   generateButton.setToolTipText("Generates 64-bit keys for encryption/" +
                                 "decryption");
   loadKeyButton = new JButton("Load Keys...");
   loadKeyButton.setToolTipText("Load keys from file");
   saveKeyButton = new JButton("Save Keys...");
   saveKeyButton.setToolTipText("Save Keys to file");

   JPanel buttonsPanel = new JPanel();
   buttonsPanel.setLayout(new GridLayout(3, 1, 3, 3));
   buttonsPanel.add(generateButton);
   buttonsPanel.add(loadKeyButton);
   buttonsPanel.add(saveKeyButton);

   JPanel labelsPanel = new JPanel();
   labelsPanel.setLayout(new GridLayout(3, 1, 3, 3));
   labelsPanel.add(new JLabel(" E:", JLabel.RIGHT));
   labelsPanel.add(new JLabel(" D:", JLabel.RIGHT));
   labelsPanel.add(new JLabel(" N:", JLabel.RIGHT));

   eTextField = new JTextField(20);
   eTextField.setEditable(false);
   dTextField = new JTextField(20);
   dTextField.setEditable(false);
   nTextField = new JTextField(20);
   nTextField.setEditable(false);

   JPanel textFieldPanel = new JPanel();
   textFieldPanel.setLayout(new GridLayout(3, 1, 3, 3));
   textFieldPanel.add(eTextField);
   textFieldPanel.add(dTextField);
   textFieldPanel.add(nTextField);

   JPanel labelFieldPanel = new JPanel();
   labelFieldPanel.setLayout(new BorderLayout(3, 3));
```

```
    labelFieldPanel.add(labelsPanel, BorderLayout.WEST);
    labelFieldPanel.add(textFieldPanel, BorderLayout.CENTER);

    JPanel keyPanel = new JPanel();
    keyPanel.setLayout(new BorderLayout(3, 3));
    keyPanel.add(buttonsPanel, BorderLayout.WEST);
    keyPanel.add(labelFieldPanel, BorderLayout.CENTER);
    keyPanel.setBorder(new EmptyBorder(5, 5, 5, 5));

    return keyPanel;
 } // End of private JPanel initializeKeyPanel()

 // Initialize the menu bar
 private JMenuBar initializeMenuBar(){
    JMenu fileMenu = new JMenu("File");
    fileMenu.setMnemonic(KeyEvent.VK_F);
    exitMenuItem = new JMenuItem("Exit");
    exitMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F4,
                                                    KeyEvent.ALT_MASK));
    exitMenuItem.setToolTipText("Exit Program");
    fileMenu.add(exitMenuItem);

    JMenu editMenu = new JMenu("Edit");
    editMenu.setMnemonic(KeyEvent.VK_E);
    addMenuItem = new JMenuItem("Add Image...");
    addMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A,
                                                   KeyEvent.CTRL_MASK));
    addMenuItem.setToolTipText("Insert an image to the image list");
    removeMenuItem = new JMenuItem("Remove Image");
    removeMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_R,
                                                      KeyEvent.CTRL_MASK));
    removeMenuItem.setToolTipText("Remove an image from the list");
    clearListMenuItem = new JMenuItem("Clear List");
    clearListMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C,
                                                      KeyEvent.CTRL_MASK));
    editMenu.add(addMenuItem);
    editMenu.add(removeMenuItem);
    editMenu.add(new JSeparator());
    editMenu.add(clearListMenuItem);

    JMenu viewMenu = new JMenu("View");
    viewMenu.setMnemonic(KeyEvent.VK_V);
    imageListPanelMenuItem = new JCheckBoxMenuItem("Image List Panel", true);
    imageListPanelMenuItem.setToolTipText("Show/Hide Image List Panel");
    watermarkPanelMenuItem = new JCheckBoxMenuItem("Watermark Panel", true);
    watermarkPanelMenuItem.setToolTipText("Show/Hide Watermark Panel");
    keysPanelMenuItem = new JCheckBoxMenuItem("Keys Panel", true);
    keysPanelMenuItem.setToolTipText("Show/Hide Keys Panel");
    viewMenu.add(imageListPanelMenuItem);
    viewMenu.add(watermarkPanelMenuItem);
    viewMenu.add(keysPanelMenuItem);

    JMenu keysMenu = new JMenu("Keys");
    keysMenu.setMnemonic(KeyEvent.VK_K);
    generateMenuItem = new JMenuItem("Generate Keys");
    generateMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_G,
                                                      KeyEvent.CTRL_MASK));
    generateMenuItem.setToolTipText("Generate 64-bit keys for insertion & " +
                                "extraction");
    loadKeysMenuItem = new JMenuItem("Load Keys...");
    loadKeysMenuItem.setToolTipText("Load previously saved private and "+
                                "public keys");
    saveKeysMenuItem = new JMenuItem("Save Keys...");
    saveKeysMenuItem.setToolTipText("Save private & public keys to file");
    clearKeysMenuItem = new JMenuItem("Clear Keys");
    clearKeysMenuItem.setToolTipText("Clears the current displayed keys");
    keysMenu.add(generateMenuItem);
    keysMenu.add(loadKeysMenuItem);
    keysMenu.add(saveKeysMenuItem);
    keysMenu.add(new JSeparator());
    keysMenu.add(clearKeysMenuItem);

    JMenu watermarkingMenu = new JMenu("Watermarking");
    watermarkingMenu.setMnemonic(KeyEvent.VK_W);
    previewMenuItem = new JMenuItem("Preview Watermark...");
    previewMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
                                                      KeyEvent.CTRL_MASK));
    previewMenuItem.setToolTipText("Preview current selected watermark");
    watermarkingMenu.add(previewMenuItem);

    JMenu extrasMenu = new JMenu("Extras");
```

```
   extrasMenu.setMnemonic(KeyEvent.VK_X);
   blockReplacementMenuItem = new JMenuItem("Block Replacement...");
   blockReplacementMenuItem.setAccelerator(KeyStroke.getKeyStroke(
                                      KeyEvent.VK_R, KeyEvent.CTRL_MASK));
   blockReplacementMenuItem.setToolTipText("Replaces specific blocks of " +
                             "tow images");
   interchangeMenuItem = new JMenuItem("Interchange Blocks...");
   interchangeMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_I,
                                                     KeyEvent.CTRL_MASK));
   interchangeMenuItem.setToolTipText("Interchange blocks of two images");
   extrasMenu.add(blockReplacementMenuItem);
   extrasMenu.add(interchangeMenuItem);

   JMenu helpMenu = new JMenu("Help");
   helpMenu.setMnemonic(KeyEvent.VK_H);
   aboutMenuItem = new JMenuItem("About Image Integrity");
   aboutMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F1, 0));
   aboutMenuItem.setToolTipText("About Image Integrity");
   helpMenu.add(aboutMenuItem);

   JMenuBar menuBar = new JMenuBar();
   menuBar.add(fileMenu);
   menuBar.add(editMenu);
   menuBar.add(viewMenu);
   menuBar.add(keysMenu);
   menuBar.add(watermarkingMenu);
   menuBar.add(extrasMenu);
   menuBar.add(helpMenu);

   return menuBar;
} // End of private JMenuBar initializeMenuBar()

// Creates the popup menu of the image list panel
private JPopupMenu initializeImageListPanelPopupMenu(){
   pAddMenuItem = new JMenuItem("Add Image...");
   pAddMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A,
                                                   KeyEvent.CTRL_MASK));
   pAddMenuItem.setToolTipText("Insert an image to the image list");
   pRemoveMenuItem = new JMenuItem("Remove Image");
   pRemoveMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_R,
                                                     KeyEvent.CTRL_MASK));
   pRemoveMenuItem.setToolTipText("Remove an image from the list");
   pClearListMenuItem = new JMenuItem("Clear List");
   pClearListMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C,
                                                     KeyEvent.CTRL_MASK));
   lImageListPanelMenuItem = new JCheckBoxMenuItem("Image List Panel", true);
   lImageListPanelMenuItem.setToolTipText("Show/Hide Image List Panel");
   lWatermarkPanelMenuItem = new JCheckBoxMenuItem("Watermark Panel", true);
   lWatermarkPanelMenuItem.setToolTipText("Show/Hide Watermark Panel");
   lKeysPanelMenuItem = new JCheckBoxMenuItem("Keys Panel", true);
   lKeysPanelMenuItem.setToolTipText("Show/Hide Keys Panel");

   JMenu viewMenu = new JMenu("View");
   viewMenu.setMnemonic(KeyEvent.VK_V);
   viewMenu.add(lImageListPanelMenuItem);
   viewMenu.add(lWatermarkPanelMenuItem);
   viewMenu.add(lKeysPanelMenuItem);

   JPopupMenu popup = new JPopupMenu();
   popup.add(pAddMenuItem);
   popup.add(pRemoveMenuItem);
   popup.add(pClearListMenuItem);
   popup.add(new JSeparator());
   popup.add(viewMenu);

   return popup;
} // End of private JPopupMenu initializeImageListPanelPopupMenu()

// Creates the popup menu for the watermark, keys, and
// display area panels
private JPopupMenu initializeWatermarkingPanelPopupMenu(){
   pGenerateMenuItem = new JMenuItem("Generate Keys");
   pGenerateMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_G,
                                                     KeyEvent.CTRL_MASK));
   pGenerateMenuItem.setToolTipText("Generate 64-bit keys for insertion" +
                                 " & extraction");
   pLoadKeysMenuItem = new JMenuItem("Load Keys...");
   pLoadKeysMenuItem.setToolTipText("Load previously saved private and "+
                                 "public keys");
   pSaveKeysMenuItem = new JMenuItem("Save Keys...");
   pSaveKeysMenuItem.setToolTipText("Save private & public keys to file");
```

```
      pClearKeysMenuItem = new JMenuItem("Clear Keys");
      pClearKeysMenuItem.setToolTipText("Clears the current displayed keys");
      pPreviewMenuItem = new JMenuItem("Preview Watermark...");
      pPreviewMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
                                                      KeyEvent.CTRL_MASK));
      pPreviewMenuItem.setToolTipText("Preview current selected watermark");
      rImageListPanelMenuItem = new JCheckBoxMenuItem("Image List Panel", true);
      rImageListPanelMenuItem.setToolTipText("Show/Hide Image List Panel");
      rWatermarkPanelMenuItem = new JCheckBoxMenuItem("Watermark Panel", true);
      rWatermarkPanelMenuItem.setToolTipText("Show/Hide Watermark Panel");
      rKeysPanelMenuItem = new JCheckBoxMenuItem("Keys Panel", true);
      rKeysPanelMenuItem.setToolTipText("Show/Hide Keys Panel");

      JMenu viewMenu = new JMenu("View");
      viewMenu.setMnemonic(KeyEvent.VK_V);
      viewMenu.add(rImageListPanelMenuItem);
      viewMenu.add(rWatermarkPanelMenuItem);
      viewMenu.add(rKeysPanelMenuItem);

      JPopupMenu popup = new JPopupMenu();
      popup.add(pPreviewMenuItem);
      popup.add(new JSeparator());
      popup.add(pGenerateMenuItem);
      popup.add(pLoadKeysMenuItem);
      popup.add(pSaveKeysMenuItem);
      popup.add(pClearKeysMenuItem);
      popup.add(new JSeparator());
      popup.add(viewMenu);

      return popup;
   } // End of private JPopupMenu initializeWatermarkingPanelPopupMenu()

   // Returns an Image icon constructed by the path if the path is valid,
   // null otherwise
   private ImageIcon getIcon(String path){
      ImageIcon icon = null;

      try{
         icon = new ImageIcon(path);
      } // End of try statement
      catch(Exception e){
         icon = null;
      } // Endof catch statement

      return icon;
   } // End of private ImageIcon getIcon(String path)

   // Returns true if the select button can be enabled, false otherwise
   public boolean canSelect(){
      if (imageList.getSelectedIndex() >= 0 &&
          watermarkComboBox.getSelectedIndex() >= 0) return true;
      else return false;
   } // end of public boolean canSelect()

   // Returns the buffered image file to be used as cover image
   public BufferedImage getImage(){
      int index = imageList.getSelectedIndex();
      if (index < 0) return null;
      else return ((ImageInfo)imageListVector.get(index)).getBufferedImage();
   } // End of public BufferedImage getImage()

   // Tiles/Scales the watermark file to fit the size of the cover image
   // according to the tiled and scaled options
   public BufferedImage getWatermark(){
      int comboIndex = watermarkComboBox.getSelectedIndex();
      int listIndex = imageList.getSelectedIndex();

      if (comboIndex < 0 || listIndex < 0){
         return null;
      } // End of if statement
      else{
         ImageInfo tmpInfo = (ImageInfo)imageListVector.get(listIndex);
         ImageInfo tmpWaterInfo = (ImageInfo)watermarkListVector.get(
                                                         comboIndex);
         BufferedImage water = new BufferedImage(tmpInfo.getWidth(),
                                                tmpInfo.getHeight(),
                                               BufferedImage.TYPE_BYTE_BINARY);
         Graphics2D graphics2D = water.createGraphics();
         graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                 RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      BufferedImage img = tmpWaterInfo.getBufferedImage();

      if (tiledRadioButton.isSelected()){
         for (int y = 0; y < tmpInfo.getHeight();
              y = y + tmpWaterInfo.getHeight()){
            for (int x = 0; x < tmpInfo.getWidth();
                 x = x + tmpWaterInfo.getWidth()){
               graphics2D.drawImage(img, x, y,
                                    tmpWaterInfo.getWidth(),
                                    tmpWaterInfo.getHeight(), null);
            } // End of for loop
         } // End of for loop
      } // End of if statement
      else{
         graphics2D.drawImage(img, 0, 0, tmpInfo.getWidth(),
                              tmpInfo.getHeight(), null);
      } // End of else statement
      img.flush();
      graphics2D.dispose();

      return water;
   } // End of else statement
} // End of public BufferedImage getWatermark

// Returns the 'e' key to be used in the RSA encryption
public BigInteger getE(){
   return e;
} // End of public BigInteger getE()

// Returns the 'd' key to be used in the RSA encryption
public BigInteger getD(){
   return d;
} // End of public BigInteger getD()

// Returns the 'n' key to be used in the RSA encryption
public BigInteger getN(){
   return n;
} // End of public BigInteger getN()

// Action Listeners event handler
public void actionPerformed(ActionEvent evt){
   if (evt.getSource() == addButton ||
       evt.getSource() == addMenuItem ||
       evt.getSource() == pAddMenuItem){

      JFileChooser fileChooser;

      try{
         fileChooser = new JFileChooser(DEFAULT_IMAGE_DIR);
      } // End of try statement
      catch(Exception e){
         fileChooser = new JFileChooser(System.getProperty("user.dir"));
      } // End of catch statement

      fileChooser.setDialogTitle("Add Image To List");
      fileChooser.setAcceptAllFileFilterUsed(false);
      fileChooser.setFileFilter(new ImageFilter());

      int result = fileChooser.showOpenDialog(this);

      if (result == JFileChooser.APPROVE_OPTION){
         ImageInfo newImage = new ImageInfo(fileChooser.getSelectedFile().
                                  getAbsolutePath());

         imageList.removeListSelectionListener(this);

         int index = imageListVector.size();
         boolean stop = false;
         boolean duplicate = false;

         while (index > 0 && !stop){
            ImageInfo tmp = (ImageInfo)imageListVector.get(index - 1);

            if (tmp.getPath().compareToIgnoreCase(newImage.getPath()) == 0){
               stop = true;
               duplicate = true;
            } // end of if statement
            else if (newImage.getFileName().compareToIgnoreCase(
                     tmp.getFileName()) > 0){
               stop = true;
            } // End of if statement
            else{
               index--;
            } // End of else statmenet
         } // End of while loop

         if (!duplicate){
            imageListVector.insertElementAt(newImage, index);
            imageList.setListData(imageListVector);
            imageList.addListSelectionListener(this);
            imageList.setSelectedValue(newImage, true);
            thumbPanel.setImageInfo((ImageInfo)imageList.getSelectedValue());
            imageCountLabel.setText("Total Images On List: " +
                                    imageListVector.size());
            removeButton.setEnabled(true);
            clearButton.setEnabled(true);
            removeMenuItem.setEnabled(true);
            pRemoveMenuItem.setEnabled(true);
            clearListMenuItem.setEnabled(true);
            pClearListMenuItem.setEnabled(true);
         } // End of if statement
         else{
            Toolkit.getDefaultToolkit().beep();
            JOptionPane.showMessageDialog(this, "The new image is already” +
                                          " in the list", "Duplicate",
                                          JOptionPane.INFORMATION_MESSAGE);
            imageList.addListSelectionListener(this);
         } // End of else statement
      } // End of if statment
   } // End of if statement
   else if (evt.getSource() == removeButton ||
            evt.getSource() == removeMenuItem ||
            evt.getSource() == pRemoveMenuItem){
      imageList.removeListSelectionListener(this);
      int index = imageList.getSelectedIndex();
      imageListVector.removeElementAt(index);
      imageList.setListData(imageListVector);
      imageCountLabel.setText("Total Images: " + imageListVector.size());
      thumbPanel.clear();

      removeButton.setEnabled(false);
      removeMenuItem.setEnabled(false);
      pRemoveMenuItem.setEnabled(false);

      if (imageListVector.size() == 0){
         clearButton.setEnabled(false);
         clearListMenuItem.setEnabled(false);
         pClearListMenuItem.setEnabled(false);
      } // End of if statement

      if (this.canSelect()){
         Wong.enableSelectButton();
         HBC1.enableSelectButton();
         HBC2.enableSelectButton();
         proposedAlgorithm.enableSelectButton();
      } // End of if statement
      else{
         Wong.disableSelectButton();
         HBC1.disableSelectButton();
         HBC2.disableSelectButton();
         proposedAlgorithm.disableSelectButton();
      } // End of else statement

      imageList.addListSelectionListener(this);
   } // End of else if statement
   else if (evt.getSource() == clearButton ||
            evt.getSource() == clearListMenuItem ||
            evt.getSource() == pClearListMenuItem){
      imageList.removeListSelectionListener(this);
      imageListVector.removeAllElements();
      imageList.setListData(imageListVector);
      imageCountLabel.setText("Total Images: " + imageListVector.size());
      thumbPanel.clear();

      removeButton.setEnabled(false);
      clearButton.setEnabled(false);
      removeMenuItem.setEnabled(false);
      pRemoveMenuItem.setEnabled(false);
      clearListMenuItem.setEnabled(false);
      pClearListMenuItem.setEnabled(false);

      if (this.canSelect()){
```

```
      Wong.enableSelectButton();
      HBC1.enableSelectButton();
      HBC2.enableSelectButton();
      proposedAlgorithm.enableSelectButton();
   } // End of if statement
   else{
      Wong.disableSelectButton();
      HBC1.disableSelectButton();
      HBC2.disableSelectButton();
      proposedAlgorithm.disableSelectButton();
   } // End of else statement

   imageList.addListSelectionListener(this);
} // End of if statement
else if (evt.getSource() == previewButton ||
         evt.getSource() == previewMenuItem ||
         evt.getSource() == pPreviewMenuItem){
   int index = watermarkComboBox.getSelectedIndex();
   new PreviewDialog(this, (ImageInfo)watermarkListVector.get(index));
} // End of else if statement
else if (evt.getSource() == imageListPanelMenuItem ||
    evt.getSource() == lImageListPanelMenuItem ||
    evt.getSource() == rImageListPanelMenuItem){
   imageListPanel.setVisible(!imageListPanel.isVisible());
   imageListPanelMenuItem.setSelected(imageListPanel.isVisible());
   lImageListPanelMenuItem.setSelected(imageListPanel.isVisible());
   rImageListPanelMenuItem.setSelected(imageListPanel.isVisible());
} // End of if statemnet
else if (evt.getSource() == watermarkPanelMenuItem ||
         evt.getSource() == lWatermarkPanelMenuItem ||
         evt.getSource() == rWatermarkPanelMenuItem){
   watermarkPanel.setVisible(!watermarkPanel.isVisible());
   watermarkPanelMenuItem.setSelected(watermarkPanel.isVisible());
   lWatermarkPanelMenuItem.setSelected(watermarkPanel.isVisible());
   rWatermarkPanelMenuItem.setSelected(watermarkPanel.isVisible());
} // End of else if statement
else if (evt.getSource() == keysPanelMenuItem ||
         evt.getSource() == lKeysPanelMenuItem ||
         evt.getSource() == rKeysPanelMenuItem){
   keysPanel.setVisible(!keysPanel.isVisible());
   keysPanelMenuItem.setSelected(keysPanel.isVisible());
   lKeysPanelMenuItem.setSelected(keysPanel.isVisible());
   rKeysPanelMenuItem.setSelected(keysPanel.isVisible());
} // End of else if statement
else if (evt.getSource() == generateButton ||
         evt.getSource() == generateMenuItem ||
         evt.getSource() == pGenerateMenuItem){
   this.setCursor(new Cursor(Cursor.WAIT_CURSOR));
   RSACryptoSystem RSA = new RSACryptoSystem(64);
   e = RSA.getE();
   d = RSA.getD();
   n = RSA.getN();
   eTextField.setText(e.toString());
   dTextField.setText(d.toString());
   nTextField.setText(n.toString());
   saveKeyButton.setEnabled(true);
   saveKeysMenuItem.setEnabled(true);
   pSaveKeysMenuItem.setEnabled(true);
   clearKeysMenuItem.setEnabled(true);
   pClearKeysMenuItem.setEnabled(true);
   this.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
} // End of else if statement
else if (evt.getSource() == loadKeyButton ||
         evt.getSource() == loadKeysMenuItem ||
         evt.getSource() == pLoadKeysMenuItem){
   JFileChooser fc;
   FileReader fileReader;
   BufferedReader reader;
   int result;

   try{
      fc = new JFileChooser(DEFAULT_KEYS_DIR);
   } // End of try statement
   catch(Exception e){
      fc = new JFileChooser(System.getProperty("user.dir"));
   } // End of catch statmenet

   fc.setDialogTitle("Load Private Key");
   fc.setAcceptAllFileFilterUsed(false);
   fc.setFileFilter(new PrivateFilter());
```

```
      result = fc.showOpenDialog(this);

      if (result == JFileChooser.APPROVE_OPTION){
         e = null;
         d = null;
         n = null;
         try{
            fileReader = new FileReader(fc.getSelectedFile());
            reader = new BufferedReader(fileReader);
            e = new BigInteger(reader.readLine());
            n = new BigInteger(reader.readLine());
            eTextField.setText(e.toString());
            dTextField.setText("N/A");
            nTextField.setText(n.toString());
            saveKeyButton.setEnabled(false);
            saveKeysMenuItem.setEnabled(false);
            pSaveKeysMenuItem.setEnabled(false);
            reader.close();
         } // End of try statement
         catch(Exception e){
            Toolkit.getDefaultToolkit().beep();
            JOptionPane.showMessageDialog(this,
               "An error occurred while reading the private key file.",
               "I/O Error", JOptionPane.ERROR_MESSAGE);
         } // end of catch statement
      } // End of if statemnet
   } // End of if statement
   else if (evt.getSource() == saveKeyButton ||
            evt.getSource() == saveKeysMenuItem ||
            evt.getSource() == pSaveKeysMenuItem){
      JFileChooser fc;
      FileWriter fileWriter;
      BufferedWriter writer;
      boolean save;
      int result;

      try{
         fc = new JFileChooser(DEFAULT_KEYS_DIR);
      } // End of try statement
      catch(Exception e){
         fc = new JFileChooser(System.getProperty("user.dir"));
      } // End of catch statement

      fc.setDialogTitle("Save Private Key");
      fc.setAcceptAllFileFilterUsed(false);
      fc.setFileFilter(new PrivateFilter());

      result = fc.showSaveDialog(this);

      if (result == JFileChooser.APPROVE_OPTION){
         File privateFile;
         if (!fc.getSelectedFile().getName().toLowerCase().endsWith(
            ".private")){
            privateFile = new File(fc.getSelectedFile().getAbsoluteFile() +
                                   ".private");
         } // End of if statement
         else{
            privateFile = fc.getSelectedFile();
         } // End of else statement

         save = true;

         if (privateFile.exists()){
            Toolkit.getDefaultToolkit().beep();
            result = JOptionPane.showConfirmDialog(this,
               "Would you like to replace the existing file?", "Replace",
               JOptionPane.YES_NO_OPTION);
            if (result == JOptionPane.NO_OPTION){
               save = false;
            } // End of if statement
         } // End of if statmenet

         if (save){
            try{
               fileWriter = new FileWriter(privateFile);
               writer = new BufferedWriter(fileWriter);
               writer.write(e.toString());
               writer.newLine();
               writer.write(n.toString());
               writer.close();
            } // End of try statement
         catch(Exception e){
            JOptionPane.showMessageDialog(this, "Error Private File");
            e.printStackTrace();
         } // End of catch statement
      } // End of if statement
   } // End of if statemment

   try{
      fc = new JFileChooser(DEFAULT_KEYS_DIR);
   } // End of try statement
   catch(Exception e){
      fc = new JFileChooser(System.getProperty("user.dir"));
   } // End of catch statement

   fc.setDialogTitle("Save Public Key");
   fc.setAcceptAllFileFilterUsed(false);
   fc.setFileFilter(new PublicFilter());

   result = fc.showSaveDialog(this);

   if (result == JFileChooser.APPROVE_OPTION){
      File publicFile;
      if (!fc.getSelectedFile().getName().toLowerCase().endsWith(
         ".public")){
         publicFile = new File(fc.getSelectedFile().getAbsoluteFile() +
                               ".public");
      } // End of if statement
      else{
         publicFile = fc.getSelectedFile();
      } // End of else statement

      save = true;

      if (publicFile.exists()){
         Toolkit.getDefaultToolkit().beep();
         result = JOptionPane.showConfirmDialog(this,
            "Would you like to replace the existing file?", "Replace",
            JOptionPane.YES_NO_OPTION);
         if (result == JOptionPane.NO_OPTION){
            save = false;
         } // End of if statement
      } // End of if statmenet

      if (save){
         try{
            fileWriter = new FileWriter(publicFile);
            writer = new BufferedWriter(fileWriter);
            writer.write(d.toString());
            writer.newLine();
            writer.write(n.toString());
            writer.close();
         } // End of try statement
         catch(Exception e){
            JOptionPane.showMessageDialog(this, "Error public file");
            e.printStackTrace();
         } // End of catch statement
      } // End of if statement
   } // End of if statemment
} // End of else statement
else if (evt.getSource() == clearKeysMenuItem ||
         evt.getSource() == pClearKeysMenuItem){
   e = null;
   d = null;
   n = null;
   eTextField.setText("");
   dTextField.setText("");
   nTextField.setText("");
   clearKeysMenuItem.setEnabled(false);
   pClearKeysMenuItem.setEnabled(false);
} // End of else if statement
else if (evt.getSource() == exitMenuItem){
   this.setVisible(false);
   this.dispose();
} // End of else if statement
else if (evt.getSource() == blockReplacementMenuItem){
   new BlockReplacementDialog(this);
} // End of else if statement
else if (evt.getSource() == interchangeMenuItem){
   new InterchangeDialog(this);
} // End of else if statement
else if (evt.getSource() == aboutMenuItem){
   } // End of else if statement
} // End of public void actionPerformed(ActionEvent evt)

// Change Listeners event handler
public void stateChanged(ChangeEvent evt){
   for (int i = 0; i < insertionTabs.getTabCount(); i++){
      if (i != insertionTabs.getSelectedIndex()){
         insertionTabs.setIconAt(i, this.getIcon(OFF_ICON));
      } // End of if statement
      else{
         insertionTabs.setIconAt(i, this.getIcon(TAB_ICON));
      } // End of else statement
   } // End of for loop
} // End of public void stateChanged(ChangeEvent evt)

// List Selection Listeners event handler
public void valueChanged(ListSelectionEvent evt){
   thumbPanel.setImageInfo((ImageInfo)imageList.getSelectedValue());
   removeButton.setEnabled(true);
   clearButton.setEnabled(true);
   removeMenuItem.setEnabled(true);
   pRemoveMenuItem.setEnabled(true);
   clearListMenuItem.setEnabled(true);
   pClearListMenuItem.setEnabled(true);

   if (this.canSelect()){
      Wong.enableSelectButton();
      HBC1.enableSelectButton();
      HBC2.enableSelectButton();
      proposedAlgorithm.enableSelectButton();
   } // End of if statement
   else{
      Wong.disableSelectButton();
      HBC1.disableSelectButton();
      HBC2.disableSelectButton();
      proposedAlgorithm.disableSelectButton();
   } // End of else statement
} // End of public void valueChanged(ListSelectionEvent evt)

public void mouseExited(MouseEvent e){ }

public void mouseEntered(MouseEvent e){ }

public void mouseClicked(MouseEvent e){ }

public void mousePressed(MouseEvent evt){
   if (evt.isPopupTrigger()){
      if (evt.getSource() == imageListPanel ||
          evt.getSource() == imageList ||
          evt.getSource() == addButton ||
          evt.getSource() == removeButton ||
          evt.getSource() == clearButton){
         imageListPanelPopupMenu.show(evt.getComponent(),
                                      evt.getX(), evt.getY());
      } // End of if statement
      else{
         watermarkingPanelPopupMenu.show(evt.getComponent(),
                                         evt.getX(), evt.getY());
      } // End of else statement
   } // End of if statement
} // End of public void mousePressed(MouseEvent evt)

public void mouseReleased(MouseEvent evt){
   if (evt.isPopupTrigger()){
      if (evt.getSource() == imageListPanel ||
          evt.getSource() == imageList ||
          evt.getSource() == addButton ||
          evt.getSource() == removeButton ||
          evt.getSource() == clearButton){
         imageListPanelPopupMenu.show(evt.getComponent(),
                                      evt.getX(), evt.getY());
      } // End of if statement
      else{
         watermarkingPanelPopupMenu.show(evt.getComponent(),
                                         evt.getX(), evt.getY());
      } // End of else statement
   } // End of if statement
} // End of public void mouseReleased(MouseEvent evt)

public static void main(String args[]){
      ImageIntegrity mainWindow = new ImageIntegrity();
   } // End of public static void main(String args[])
} // End of public class ImageIntegrity extends JFrame
```

B.2 BlockReplacementDialog.java

```
// BlockReplacementDialog.java

import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.io.*;
import javax.imageio.*;

public class BlockReplacementDialog extends JDialog implements
   ActionListener, ChangeListener{

   private static final int BLOCK_WIDTH = 8;
   private static final int BLOCK_HEIGHT = 8;
   private static final Color CAPTION_BACKGROUND = new Color(184, 207, 229);
   private static final double DISPLAY_PERCENTAGE = 0.95;
   private static final String DEFAULT_SAVE_DIR = "saved/";
   private static final String UP_ICON = "./icons/up.gif";
   private static final String DOWN_ICON = "./icons/down.gif";
   private static final String LEFT_ICON = "./icons/left.gif";
   private static final String RIGHT_ICON = "./icons/right.gif";

   protected JButton originalButton, libraryButton, clearButton;
   protected JButton up, down, right, left, saveButton, closeButton;
   protected JLabel originalLabel, libraryLabel, thumbCaption1, thumbCaption2;
   protected JLabel xPosLabel, yPosLabel, widthLabel, heightLabel, displayLabel;
   protected JTextField originalTextField, libraryTextField;
   protected JTextField xPosTextField, yPosTextField;
   protected JRadioButton originalRadioButton, libraryRadioButton;
   protected JCheckBox gridCheckBox;
   protected JSpinner widthSpinner, heightSpinner;
   protected SpinnerNumberModel widthModel, heightModel;
   protected ThumbPanel thumb1, thumb2;
   protected ImagePanel replacementPanel;
   protected BufferedImage image1 = null, image2 = null, displayImage = null,
                           replacementImage = null;
   protected Graphics2D graphics2D = null, replacementGraphics2D = null;
   protected int blockWidthCount, blockHeightCount;
   protected int currentBlockX = 0, currentBlockY = 0;

   public BlockReplacementDialog(ImageIntegrity pParent){
      super(pParent, true);

      Rectangle desktopSize = GraphicsEnvironment.
                  getLocalGraphicsEnvironment().getMaximumWindowBounds();

      this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
      this.setSize((int)(desktopSize.getWidth() * DISPLAY_PERCENTAGE),
                   (int)(desktopSize.getHeight() * DISPLAY_PERCENTAGE));
      this.setTitle("Block Replacent");

      originalButton = new JButton("Browse...");
      originalLabel = new JLabel("Original Image: ", JLabel.RIGHT);
      originalLabel.setOpaque(true);
      originalLabel.setBackground(CAPTION_BACKGROUND);
      originalTextField = new JTextField(20);
      originalTextField.setEditable(false);

      libraryButton = new JButton("Browse...");
      libraryLabel = new JLabel("Library Image: ", JLabel.RIGHT);
      libraryLabel.setOpaque(true);
      libraryLabel.setBackground(CAPTION_BACKGROUND);
      libraryTextField = new JTextField(20);
      libraryTextField.setEditable(false);

      thumbCaption1 = new JLabel("Preview Original Image", JLabel.CENTER);
      thumbCaption1.setOpaque(true);
      thumbCaption1.setBackground(CAPTION_BACKGROUND);
      thumbCaption2 = new JLabel("Preview Library Image", JLabel.CENTER);
```

```
thumbCaption2.setOpaque(true);
thumbCaption2.setBackground(CAPTION_BACKGROUND);

thumb1 = new ThumbPanel();
thumb2 = new ThumbPanel();

up = new JButton(new ImageIcon(UP_ICON));
down = new JButton(new ImageIcon(DOWN_ICON));
left = new JButton(new ImageIcon(LEFT_ICON));
right = new JButton(new ImageIcon(RIGHT_ICON));

xPosLabel = new JLabel("Block Start X: ", JLabel.RIGHT);
xPosLabel.setOpaque(true);
xPosLabel.setBackground(CAPTION_BACKGROUND);
xPosTextField = new JTextField("0", 5);
xPosTextField.setHorizontalAlignment(JTextField.CENTER);
xPosTextField.setEditable(false);

yPosLabel = new JLabel("Block Start Y: ", JLabel.RIGHT);
yPosLabel.setOpaque(true);
yPosLabel.setBackground(CAPTION_BACKGROUND);
yPosTextField = new JTextField("0", 5);
yPosTextField.setHorizontalAlignment(JTextField.CENTER);
yPosTextField.setEditable(false);

widthLabel = new JLabel("Block Width: ", JLabel.RIGHT);
widthLabel.setOpaque(true);
widthLabel.setBackground(CAPTION_BACKGROUND);
widthModel = new SpinnerNumberModel();
widthModel.setMinimum(1);
widthModel.setValue(1);
widthSpinner = new JSpinner(widthModel);
widthSpinner.setEnabled(false);

heightLabel = new JLabel("Block Height: ", JLabel.RIGHT);
heightLabel.setOpaque(true);
heightLabel.setBackground(CAPTION_BACKGROUND);
heightModel = new SpinnerNumberModel();
heightModel.setMinimum(1);
heightModel.setValue(1);
heightSpinner = new JSpinner(heightModel);
heightSpinner.setEnabled(false);

clearButton = new JButton("Clear Contents");
clearButton.setMnemonic(KeyEvent.VK_L);
clearButton.setToolTipText("Clear Contents");
clearButton.setEnabled(false);

displayLabel = new JLabel("Display: ", JLabel.RIGHT);
displayLabel.setOpaque(true);
displayLabel.setBackground(CAPTION_BACKGROUND);
originalRadioButton = new JRadioButton("Original Image", true);
originalRadioButton.setEnabled(false);
libraryRadioButton = new JRadioButton("Library Image");
libraryRadioButton.setEnabled(false);
ButtonGroup group = new ButtonGroup();
group.add(originalRadioButton);
group.add(libraryRadioButton);

replacementPanel = new ImagePanel(true);

gridCheckBox = new JCheckBox("Show Grid", true);
gridCheckBox.setEnabled(false);

saveButton = new JButton("Save Image");
saveButton.setEnabled(false);
saveButton.setToolTipText("Save the current displaying interchanged " +
                          "image");

closeButton = new JButton("Close Window");
closeButton.setMnemonic(KeyEvent.VK_C);
closeButton.setToolTipText("Close Window");

this.setLayout(null);
this.add(originalButton);
this.add(originalLabel);
this.add(originalTextField);
this.add(libraryButton);
this.add(libraryLabel);
this.add(libraryTextField);
this.add(thumbCaption1);
this.add(thumbCaption2);
this.add(thumb1);
this.add(thumb2);
this.add(up);
this.add(down);
this.add(left);
this.add(right);
this.add(xPosLabel);
this.add(xPosTextField);
this.add(yPosLabel);
this.add(yPosTextField);
this.add(widthLabel);
this.add(widthSpinner);
this.add(heightLabel);
this.add(heightSpinner);
this.add(displayLabel);
this.add(originalRadioButton);
this.add(libraryRadioButton);
this.add(clearButton);
this.add(replacementPanel);
this.add(gridCheckBox);
this.add(saveButton);
this.add(closeButton);

int width = this.getWidth() - 20;
int height = this.getHeight() - 40;

originalButton.setBounds(10, 10, 90, 25);
originalLabel.setBounds(110, 12, 110, 20);
originalTextField.setBounds(230, 12, width - 240, 20);
libraryButton.setBounds(10, 40, 90, 25);
libraryLabel.setBounds(110, 42, 110, 20);
libraryTextField.setBounds(230, 42, width - 240, 20);
thumbCaption1.setBounds(10, 70, 250, 20);
thumb1.setBounds(10, 95, 250, 250);
thumbCaption2.setBounds(10, 360, 250, 20);
thumb2.setBounds(10, 385, 250, 250);
up.setBounds(310, 70, 35, 35);
down.setBounds(310, 110, 35, 35);
left.setBounds(270, 90, 35, 35);
right.setBounds(350, 90, 35, 35);
xPosLabel.setBounds(395, 75, 100, 20);
xPosTextField.setBounds(500, 75, 60, 20);
yPosLabel.setBounds(565, 75, 100, 20);
yPosTextField.setBounds(670, 75, 60, 20);
widthLabel.setBounds(395, 100, 100, 20);
widthSpinner.setBounds(500, 100, 60, 20);
heightLabel.setBounds(565, 100, 100, 20);
heightSpinner.setBounds(670, 100, 60, 20);
displayLabel.setBounds(395, 125, 100, 20);
originalRadioButton.setBounds(500, 125, 110, 20);
libraryRadioButton.setBounds(615, 125, 110, 20);
clearButton.setBounds(735, 120, width - 745, 25);
replacementPanel.setBounds(270, 150, width - 270, height - 185);
gridCheckBox.setBounds(width - 120 - 120 - 90 - 5 - 5, height - 25,90,25);
saveButton.setBounds(width - 120 - 120 - 5, height - 25, 120, 25);
closeButton.setBounds(width - 120, height - 25, 120, 25);

//reset();    // Reset the contents

// Registering components
originalButton.addActionListener(this);
libraryButton.addActionListener(this);
clearButton.addActionListener(this);
closeButton.addActionListener(this);
originalRadioButton.addActionListener(this);
libraryRadioButton.addActionListener(this);
saveButton.addActionListener(this);
up.addActionListener(this);
down.addActionListener(this);
left.addActionListener(this);
right.addActionListener(this);
widthSpinner.addChangeListener(this);
heightSpinner.addChangeListener(this);
gridCheckBox.addActionListener(this);

reset();

this.center();
this.setResizable(false);
this.setVisible(true);
```

```
} // End of public BlockReplacementDialog(ImageIntegrity pParent)

// Centers the dialog respective to the desktop space
private void center(){
   Rectangle desktopSize = GraphicsEnvironment.
                     getLocalGraphicsEnvironment().getMaximumWindowBounds();
   this.setLocation((int)((desktopSize.getWidth() - this.getWidth()) / 2),
                   (int)((desktopSize.getHeight() - this.getHeight()) / 2));
} // End of private void center()

public void dispose(){
   if (image1 != null) image1.flush();
   if (image2 != null) image2.flush();
   if (replacementImage != null) replacementImage.flush();
   if (displayImage != null) displayImage.flush();
   if (replacementGraphics2D != null) replacementGraphics2D.dispose();
   if (graphics2D != null) graphics2D.dispose();
} // End of public void dispose()

private void reset(){
   widthSpinner.removeChangeListener(this);
   heightSpinner.removeChangeListener(this);

   if (image1 != null) image1.flush();
   if (image2 != null) image2.flush();
   if (displayImage != null) displayImage.flush();
   if (replacementImage != null) replacementImage.flush();

   image1 = null;
   image2 = null;
   displayImage = null;
   replacementImage = null;
   blockWidthCount = 0;
   blockHeightCount = 0;
   currentBlockX = 0;
   currentBlockY = 0;
   originalTextField.setText("");
   libraryTextField.setText("");
   thumb1.clear();
   thumb2.clear();
   replacementPanel.clear();
   up.setEnabled(false);
   down.setEnabled(false);
   left.setEnabled(false);
   right.setEnabled(false);
   xPosTextField.setText("0");
   yPosTextField.setText("0");
   widthModel.setValue(1);
   heightModel.setValue(1);
   originalRadioButton.setSelected(true);
   originalRadioButton.setEnabled(false);
   libraryRadioButton.setEnabled(false);
   widthSpinner.setEnabled(false);
   heightSpinner.setEnabled(false);
   clearButton.setEnabled(false);
   saveButton.setEnabled(false);
   gridCheckBox.setEnabled(false);
   widthSpinner.addChangeListener(this);
   heightSpinner.addChangeListener(this);
} // End of private void reset()

private void updateImage(){
   if (originalRadioButton.isSelected()){
      replacementGraphics2D.drawImage(image1, 0, 0, image1.getWidth(),
                                      image1.getHeight(), null);
      graphics2D.drawImage(image1, 0, 0, image1.getWidth(),
                           image1.getHeight(), null);
   } // End of if statement
   else{
      replacementGraphics2D.drawImage(image2, 0, 0, image2.getWidth(),
                                      image2.getHeight(), null);
      graphics2D.drawImage(image2, 0, 0, image2.getWidth(),
                           image2.getHeight(), null);
   } // End of else statement

   int width = 0, height = 0;

   if (currentBlockX + (Integer.parseInt(widthSpinner.getValue().toString())
          * BLOCK_WIDTH) >= image1.getWidth()){
      width = image1.getWidth() - currentBlockX;
   } // End of if statement
   else{
      width = Integer.parseInt(widthSpinner.getValue().toString()) *
              BLOCK_WIDTH;
   } // End of else statement

   if (currentBlockY + (Integer.parseInt(heightSpinner.getValue().toString())
          * BLOCK_HEIGHT) >= image1.getHeight()){
      height = image1.getHeight() - currentBlockY;
   } // End of if statement
   else{
      height = Integer.parseInt(heightSpinner.getValue().toString()) *
               BLOCK_HEIGHT;
   } // End of else statement

   BufferedImage blockImage = new BufferedImage(width, height,
                                             BufferedImage.TYPE_INT_RGB);

   int RGB = 0;
   for (int i = 0; i < blockImage.getWidth(); i++){
      for (int j = 0; j < blockImage.getHeight(); j++){
         if (originalRadioButton.isSelected())
            RGB = image2.getRGB(i + currentBlockX, j + currentBlockY);
         else
            RGB = image1.getRGB(i + currentBlockX, j + currentBlockY);

         blockImage.setRGB(i, j, RGB);
      } // End of for loop
   } // End of for loop

   replacementGraphics2D.drawImage(blockImage, currentBlockX, currentBlockY,
                                   blockImage.getWidth(),
                                   blockImage.getHeight(),
                                   null);
   graphics2D.drawImage(blockImage, currentBlockX, currentBlockY,
                        blockImage.getWidth(), blockImage.getHeight(), null);
   graphics2D.setColor(Color.RED);
   graphics2D.drawRect(currentBlockX, currentBlockY, blockImage.getWidth(),
                       blockImage.getHeight());

   if (gridCheckBox.isSelected()){
      for (int k1 = currentBlockX; k1 < currentBlockX + width; k1 = k1 + 4)
         graphics2D.drawLine(k1, currentBlockY, k1, currentBlockY + height);

      for (int k2 = currentBlockY; k2 < currentBlockY + height; k2 = k2 + 4)
         graphics2D.drawLine(currentBlockX, k2, currentBlockX + width, k2);
   } // End of if statement

   blockImage.flush();

   replacementPanel.setImage(displayImage);
} // end of private void updateImage()

public void actionPerformed(ActionEvent evt){
   if (evt.getSource() == originalButton){
      JFileChooser fileChooser;

      try{
         fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
      } // End of try statement
      catch(Exception e){
         fileChooser = new JFileChooser(System.getProperty("user.dir"));
      } // End of catch statement

      fileChooser.setDialogTitle("Open Image");
      fileChooser.setAcceptAllFileFilterUsed(false);
      //fileChooser.setFileFilter(new PNGFilter());
      fileChooser.setFileFilter(new ImageFilter());

      int result = fileChooser.showOpenDialog(this);

      if (result == JFileChooser.APPROVE_OPTION){
         String path = fileChooser.getSelectedFile().getAbsolutePath();
         ImageInfo info = new ImageInfo(path);
         originalTextField.setText(info.getPath());
         thumb1.setImageInfo(info);

         image1 = info.getBufferedImage();
      } // End of if statement

      if (image1 != null && image2 != null){
         if (image1.getWidth() == image2.getWidth() &&
             image1.getHeight() == image2.getHeight()){
            blockWidthCount = image1.getWidth() / BLOCK_WIDTH;
            blockHeightCount = image1.getHeight() / BLOCK_HEIGHT;

            if (image1.getWidth() % BLOCK_WIDTH != 0) blockWidthCount++;
            if (image1.getHeight() % BLOCK_HEIGHT != 0) blockHeightCount++;

            widthSpinner.removeChangeListener(this);
            heightSpinner.removeChangeListener(this);

            widthModel.setMaximum(blockWidthCount);
            heightModel.setMaximum(blockHeightCount);

            down.setEnabled(true);
            right.setEnabled(true);
            widthSpinner.setEnabled(true);
            heightSpinner.setEnabled(true);
            originalRadioButton.setEnabled(true);
            libraryRadioButton.setEnabled(true);
            clearButton.setEnabled(true);
            saveButton.setEnabled(true);
            gridCheckBox.setEnabled(true);

            replacementImage = new BufferedImage(image1.getWidth(),
                                                 image1.getHeight(),
                                                 BufferedImage.TYPE_INT_RGB);
            replacementGraphics2D = replacementImage.createGraphics();
            replacementGraphics2D.setRenderingHint(
                               RenderingHints.KEY_INTERPOLATION,
                               RenderingHints.VALUE_INTERPOLATION_BILINEAR);
            displayImage = new BufferedImage(image1.getWidth(),
                                             image1.getHeight(),
                                             BufferedImage.TYPE_INT_RGB);
            graphics2D = displayImage.createGraphics();
            graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                RenderingHints.VALUE_INTERPOLATION_BILINEAR);
            replacementGraphics2D.drawImage(image1, 0, 0, image1.getWidth(),
                                            image1.getHeight(), null);
            graphics2D.drawImage(image1, 0, 0, image1.getWidth(),
                                 image1.getHeight(), null);

            widthSpinner.addChangeListener(this);
            heightSpinner.addChangeListener(this);

            updateImage();
         } // End of if statement
         else{
            Toolkit.getDefaultToolkit().beep();
            JOptionPane.showMessageDialog(this, "The width and height of " +
               "images does not match.", "Dimension Mismatch",
               JOptionPane.WARNING_MESSAGE);
            reset();
         } // End of else statement
      } // End of if statement
   } // End of if stamtent
   else if (evt.getSource() == libraryButton){
      JFileChooser fileChooser;

      try{
         fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
      } // End of try statement
      catch(Exception e){
         fileChooser = new JFileChooser(System.getProperty("user.dir"));
      } // End of catch statement

      fileChooser.setDialogTitle("Open Image");
      fileChooser.setAcceptAllFileFilterUsed(false);
      //fileChooser.setFileFilter(new PNGFilter());
      fileChooser.setFileFilter(new ImageFilter());

      int result = fileChooser.showOpenDialog(this);

      if (result == JFileChooser.APPROVE_OPTION){
         String path = fileChooser.getSelectedFile().getAbsolutePath();
         ImageInfo info = new ImageInfo(path);
         libraryTextField.setText(info.getPath());
         thumb2.setImageInfo(info);

         image2 = info.getBufferedImage();
      } // End of if statement
      if (image1 != null && image2 != null){
         if (image1.getWidth() == image2.getWidth() &&
             image1.getHeight() == image2.getHeight()){
            blockWidthCount = image1.getWidth() / BLOCK_WIDTH;
            blockHeightCount = image1.getHeight() / BLOCK_HEIGHT;

            if (image1.getWidth() % BLOCK_WIDTH != 0) blockWidthCount++;
            if (image1.getHeight() % BLOCK_HEIGHT != 0) blockHeightCount++;

            widthSpinner.removeChangeListener(this);
            heightSpinner.removeChangeListener(this);

            widthModel.setMaximum(blockWidthCount);
            heightModel.setMaximum(blockHeightCount);

            down.setEnabled(true);
            right.setEnabled(true);
            widthSpinner.setEnabled(true);
            heightSpinner.setEnabled(true);
            originalRadioButton.setEnabled(true);
            libraryRadioButton.setEnabled(true);
            clearButton.setEnabled(true);
            saveButton.setEnabled(true);
            gridCheckBox.setEnabled(true);

            replacementImage = new BufferedImage(image2.getWidth(),
                                                 image2.getHeight(),
                                                 BufferedImage.TYPE_INT_RGB);
            replacementGraphics2D = replacementImage.createGraphics();
            replacementGraphics2D.setRenderingHint(
                               RenderingHints.KEY_INTERPOLATION,
                               RenderingHints.VALUE_INTERPOLATION_BILINEAR);
            displayImage = new BufferedImage(image2.getWidth(),
                                             image2.getHeight(),
                                             BufferedImage.TYPE_INT_RGB);
            graphics2D = displayImage.createGraphics();
            graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                RenderingHints.VALUE_INTERPOLATION_BILINEAR);
            replacementGraphics2D.drawImage(image1, 0, 0, image2.getWidth(),
                                            image2.getHeight(), null);
            graphics2D.drawImage(image2, 0, 0, image2.getWidth(),
                                 image2.getHeight(), null);

            widthSpinner.addChangeListener(this);
            heightSpinner.addChangeListener(this);
            updateImage();
         } // end of if statement
         else{
            Toolkit.getDefaultToolkit().beep();
            JOptionPane.showMessageDialog(this, "The width and height of " +
               "images does not match.", "Dimension Mismatch",
               JOptionPane.WARNING_MESSAGE);
            reset();
         } // End of else statement
      } // End of if statement
   } // End of else if statement
   else if (evt.getSource() == up){
      currentBlockY -= BLOCK_HEIGHT;
      yPosTextField.setText(String.valueOf(currentBlockY));
      if (currentBlockY <= 0) up.setEnabled(false);
      down.setEnabled(true);

      int count = (image1.getHeight() - currentBlockY) / BLOCK_HEIGHT;
      if ((image1.getHeight() - currentBlockY) % BLOCK_HEIGHT != 0) count++;
      heightModel.setMaximum(count);

      if (Integer.parseInt(heightSpinner.getValue().toString()) > count)
         heightModel.setValue(count);
   } // End of else statement
   else if (evt.getSource() == down){
      currentBlockY += BLOCK_HEIGHT;
      yPosTextField.setText(String.valueOf(currentBlockY));
      int height = Integer.parseInt(heightSpinner.getValue().toString()) *
                   BLOCK_HEIGHT;
      if (currentBlockY+height >= image1.getHeight()) down.setEnabled(false);
      up.setEnabled(true);

      int count = (image1.getHeight() - currentBlockY) / BLOCK_HEIGHT;
      if ((image1.getHeight() - currentBlockY) % BLOCK_HEIGHT != 0) count++;
      heightModel.setMaximum(count);
      if (Integer.parseInt(heightSpinner.getValue().toString()) > count)
         heightModel.setValue(count);
   } // end of else if statement
   else if (evt.getSource() == left){
      currentBlockX -= BLOCK_WIDTH;
      xPosTextField.setText(String.valueOf(currentBlockX));
      if (currentBlockX <= 0) left.setEnabled(false);
      right.setEnabled(true);

      int count = (image1.getWidth() - currentBlockX) / BLOCK_WIDTH;
      if ((image1.getWidth() - currentBlockX) % BLOCK_WIDTH != 0) count++;
      widthModel.setMaximum(count);

      if (Integer.parseInt(widthSpinner.getValue().toString()) > count)
         widthModel.setValue(count);
   } // End of else if statement
   else if (evt.getSource() == right){
      currentBlockX += BLOCK_WIDTH;
      xPosTextField.setText(String.valueOf(currentBlockX));
      int width = Integer.parseInt(widthSpinner.getValue().toString()) *
                  BLOCK_WIDTH;

      if (currentBlockX+width >= image1.getWidth()) right.setEnabled(false);
      left.setEnabled(true);

      int count = (image1.getWidth() - currentBlockX) / BLOCK_WIDTH;
      if ((image1.getWidth() - currentBlockX) % BLOCK_WIDTH != 0) count++;
      widthModel.setMaximum(count);

      if (Integer.parseInt(widthSpinner.getValue().toString()) > count)
         widthModel.setValue(count);
   } // End of else if statement
   else if (evt.getSource() == originalRadioButton){
      replacementImage = new BufferedImage(image1.getWidth(),
                                           image1.getHeight(),
                                           BufferedImage.TYPE_INT_RGB);
      replacementGraphics2D = replacementImage.createGraphics();
      replacementGraphics2D.setRenderingHint(
                      RenderingHints.KEY_INTERPOLATION,
                      RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      displayImage = new BufferedImage(image1.getWidth(),
                                       image1.getHeight(),
                                       BufferedImage.TYPE_INT_RGB);
      graphics2D = displayImage.createGraphics();
      graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                             RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      replacementGraphics2D.drawImage(image1, 0, 0, image1.getWidth(),
                                      image1.getHeight(), null);
      graphics2D.drawImage(image1, 0, 0, image1.getWidth(),
                        image1.getHeight(), null);

      updateImage();
   } // End of if statement
   else if (evt.getSource() == libraryRadioButton){
      replacementImage = new BufferedImage(image2.getWidth(),
                                           image2.getHeight(),
                                           BufferedImage.TYPE_INT_RGB);
      replacementGraphics2D = replacementImage.createGraphics();
      replacementGraphics2D.setRenderingHint(
                      RenderingHints.KEY_INTERPOLATION,
                      RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      displayImage = new BufferedImage(image2.getWidth(),
                                       image2.getHeight(),
                                       BufferedImage.TYPE_INT_RGB);
      graphics2D = displayImage.createGraphics();
      graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                             RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      replacementGraphics2D.drawImage(image1, 0, 0, image2.getWidth(),
                                      image2.getHeight(), null);
      graphics2D.drawImage(image2, 0, 0, image2.getWidth(),
                        image2.getHeight(), null);

      updateImage();
   } // End of else statement
   else if (evt.getSource() == gridCheckBox){
      updateImage();
   } // end of else if statement
   else if (evt.getSource() == clearButton){
      reset();
   } // end of else if statement
   else if (evt.getSource() == saveButton){
```

```
         JFileChooser fileChooser = null;

         try{
            fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
         } // End of try statement
         catch(Exception e){
             fileChooser = new JFileChooser(System.getProperty("user.dir"));
         } // End of catch statement

         fileChooser.setAcceptAllFileFilterUsed(false);
         fileChooser.setFileFilter(new PNGFilter());

         int result = fileChooser.showSaveDialog(null);

         if (result == JFileChooser.APPROVE_OPTION){
            String filePath = fileChooser.getSelectedFile().getAbsolutePath();

            if (!filePath.toLowerCase().endsWith(".png")) filePath += ".png";

            result = JOptionPane.YES_OPTION;

            if (new File(filePath).exists()){
               Toolkit.getDefaultToolkit().beep();
               result = JOptionPane.showConfirmDialog(null, "Do you want to " +
                            "replace the existing file?", "File exists",
                            JOptionPane.YES_NO_OPTION);
            } // End of if statement

            if (result == JOptionPane.YES_OPTION){
               fileChooser.setCursor(new Cursor(Cursor.WAIT_CURSOR));

               try{
                  ImageIO.write(replacementImage, "PNG",
                                new File(filePath));
               } // End of try statement
               catch(Exception ex){
                  Toolkit.getDefaultToolkit().beep();
                  JOptionPane.showMessageDialog(null, "An error occurred " +
                            "while saving the requested image.", "I/O Error",
                            JOptionPane.ERROR_MESSAGE);
               } // end of catch statement
               finally{
                  fileChooser.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
               } // End of finally
            } // End of if statement
         } // End of if statement
      } // End of else if statement
      else if (evt.getSource() == closeButton){
         this.setVisible(false);
         this.dispose();
      } // End of else if statement
   } // End of public void actionPerformed(ActionEvent evt)

   public void stateChanged(ChangeEvent evt){
      updateImage();
   } // End of public void stateChanged(ChangeEvent evt)
} // End of public class BlockReplacementDialog extends JDialog implements
  //     ActionListener, ChangeListener
```

B.3 DisplayArea.java

```
// DisplayArea.java
//
// DisplayArea provides the visual representation of the components that
// are located inside each of the tabs.

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;
import java.math.BigInteger;

public abstract class DisplayArea extends JPanel implements ActionListener
{
   protected String ALGORITHM_TEXT = "NO SELECTED IMAGE";
protected JButton selectButton, enlargeButton, clearButton,
                  insertionButton, extractionButton;
protected JRadioButton imageRadioButton, watermarkRadioButton;
protected JPanel buttonsPanel;
protected BufferedImage image, watermark;
protected ImageIntegrity parent;

public DisplayArea(ImageIntegrity pParent){
   parent = pParent;
   image = null;
   watermark = null;

   // Initializing components
   selectButton = new JButton("Select");
   selectButton.setMnemonic(KeyEvent.VK_S);
   selectButton.setToolTipText("Select the current image and watermark " +
                               "for insertion");
   selectButton.setEnabled(false);
   enlargeButton = new JButton("Enlarge...");
   enlargeButton.setToolTipText("Enlarge the current displayed image");
   enlargeButton.setEnabled(false);
   clearButton = new JButton("Clear Area");
   clearButton.setToolTipText("Clear the contents of the display area");
   clearButton.setEnabled(false);
   imageRadioButton = new JRadioButton("Cover Image", true);
   imageRadioButton.setToolTipText("Displays the cover image");
   imageRadioButton.setEnabled(false);
   watermarkRadioButton = new JRadioButton("Watermark");
   watermarkRadioButton.setToolTipText("Display the watermark image");
   watermarkRadioButton.setEnabled(false);
   ButtonGroup buttonGroup = new ButtonGroup();
   buttonGroup.add(imageRadioButton);
   buttonGroup.add(watermarkRadioButton);
   insertionButton = new JButton("Insert...");
   insertionButton.setMnemonic(KeyEvent.VK_I);
   insertionButton.setToolTipText("Starts watermark insertion process");
   insertionButton.setEnabled(false);
   extractionButton = new JButton("Extract...");
   extractionButton.setMnemonic(KeyEvent.VK_E);
   extractionButton.setToolTipText("Starts watermark extraction process");

   buttonsPanel = new JPanel();
   buttonsPanel.setLayout(new GridLayout(1, 7, 3, 3));
   buttonsPanel.add(selectButton);
   buttonsPanel.add(imageRadioButton);
   buttonsPanel.add(watermarkRadioButton);
   buttonsPanel.add(enlargeButton);
   buttonsPanel.add(clearButton);
   buttonsPanel.add(extractionButton);
   buttonsPanel.add(insertionButton);
   buttonsPanel.setBorder(new EmptyBorder(5, 5, 5, 5));

   selectButton.addActionListener(this);
   imageRadioButton.addActionListener(this);
   watermarkRadioButton.addActionListener(this);
   enlargeButton.addActionListener(this);
   clearButton.addActionListener(this);
   extractionButton.addActionListener(this);
   insertionButton.addActionListener(this);

   this.setLayout(new BorderLayout(3, 3));
   this.add(buttonsPanel, BorderLayout.NORTH);
} // End of public DisplayArea(ImageIntegrity pParent)

// Enables the select button
public void enableSelectButton(){
   selectButton.setEnabled(true);
} // End of public void enableSelectButton()

// Disables the select button
public void disableSelectButton(){
   selectButton.setEnabled(false);
} // End of public void disableSelectButton

// Paint method, calls the paint method of the base class. Updates
// and scales the images in the display area adding borders and captions
public void paint(Graphics g){
   super.paint(g);

   this.setCursor(new Cursor(Cursor.WAIT_CURSOR));
   int width = this.getWidth() - 10;
   int height = this.getHeight() - buttonsPanel.getHeight() - 25;

   if (image == null){   // No Image to Display
      g.drawRect(5, 35, width, height + 20);
      g.drawRect(6, 36, width - 2, height + 20 - 2);
      Font font = new Font("Arial", Font.BOLD, 30);
      FontMetrics fontMetrics = this.getFontMetrics(font);
      int fontWidth = fontMetrics.stringWidth(ALGORITHM_TEXT);

      if (height - 30 > fontMetrics.getHeight()){
         g.setFont(font);
         g.drawString(ALGORITHM_TEXT, (width / 2) - (fontWidth / 2) + 5,
                     (height / 2) - fontMetrics.getHeight() + 35 + 20);
         fontWidth = fontMetrics.stringWidth("NO SELECTED IMAGE");
         g.drawString("NO SELECTED IMAGE", (width / 2) - (fontWidth / 2) + 5,
                     (height / 2) + 35 + 20);
      } // End of if statement
   } // End of if statement
   else{
      int startX = 10, startY = 45;
      float percentage = 1.0f;

      while ((int)(image.getWidth() * percentage) > width)
         percentage -= 0.01;
      while ((int)(image.getHeight() * percentage) > height)
         percentage -= 0.01;

      BufferedImage scaled = new BufferedImage(
           (int)(image.getWidth() * percentage),
           (int)(image.getHeight() * percentage),
           BufferedImage.TYPE_INT_RGB);
      Graphics2D graphics2D = scaled.createGraphics();
      graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                             RenderingHints.VALUE_INTERPOLATION_BILINEAR);

      if (imageRadioButton.isSelected())
         graphics2D.drawImage(image, 0, 0, scaled.getWidth(),
                             scaled.getHeight(), null);
      else
         graphics2D.drawImage(watermark, 0, 0, scaled.getWidth(),
                             scaled.getHeight(), null);
      graphics2D.dispose();

      // Calculating starting point x and y to draw image
      startX += (width / 2) - (scaled.getWidth() / 2) - 5;
      startY += (height / 2) - (scaled.getHeight() / 2) - 5;

      g.drawImage(new ImageIcon(scaled).getImage(), startX, startY,
                  scaled.getWidth(), scaled.getHeight(), null);

      // Draw borders of the image
      g.drawRect(startX - 4, startY - 4, scaled.getWidth() + 8,
                 scaled.getHeight() + 8);
      g.drawRect(startX - 3, startY - 3, scaled.getWidth() + 6,
                 scaled.getHeight() + 6);

      Font font = new Font("Arial", Font.BOLD, 12);
      FontMetrics fontMetrics = this.getFontMetrics(font);
      String imgText;

      if (imageRadioButton.isSelected()) imgText = "Original Image";
      else imgText = "Watermark Image";

      imgText += " - Original Size: " + image.getWidth() + " x " +
                 image.getHeight() + " - Display Size: " +
                 scaled.getWidth() + " x " + scaled.getHeight() +
                 " (" + (int)(percentage * 100) + "%)";
      int fontWidth = fontMetrics.stringWidth(imgText);

      // Displaying image size info
      g.setFont(font);
      g.drawString(imgText, (width / 2) - (fontWidth / 2) + 5,
                   startY + scaled.getHeight() + 18);
   } // End of else statement
   this.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
} // End of public void paint(Graphics g)

// Action listeners event handler
public void actionPerformed(ActionEvent evt){
   if (evt.getSource() == selectButton){
      this.setCursor(new Cursor(Cursor.WAIT_CURSOR));
      BufferedImage tmpImage = null;
      Graphics2D graphics2D = null;

      if (image != null) image.flush();
      if (watermark != null) watermark.flush();

      tmpImage = parent.getImage();
      image = new BufferedImage(tmpImage.getWidth(), tmpImage.getHeight(),
                                BufferedImage.TYPE_INT_RGB);
      graphics2D = image.createGraphics();
      graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      graphics2D.drawImage(tmpImage, 0, 0, tmpImage.getWidth(),
                           tmpImage.getHeight(), null);

      tmpImage = parent.getWatermark();
      watermark = new BufferedImage(tmpImage.getWidth(),
                                    tmpImage.getHeight(),
                                    BufferedImage.TYPE_INT_RGB);
      graphics2D = watermark.createGraphics();
      graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      graphics2D.drawImage(tmpImage, 0, 0, tmpImage.getWidth(),
                           tmpImage.getHeight(), null);
      tmpImage.flush();
      graphics2D.dispose();

      imageRadioButton.setEnabled(true);
      watermarkRadioButton.setEnabled(true);
      enlargeButton.setEnabled(true);
      clearButton.setEnabled(true);
      insertionButton.setEnabled(true);
      this.repaint();

      this.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
   } // End of if statement
   else if (evt.getSource() == imageRadioButton){
      this.repaint();
   } // End of else statement
   else if (evt.getSource() == watermarkRadioButton){
      this.repaint();
   } // End of else statement
   else if (evt.getSource() == enlargeButton){
      this.setCursor(new Cursor(Cursor.WAIT_CURSOR));

      if (imageRadioButton.isSelected())
         new EnlargeDialog(parent, image, "Original Image");
      else
         new EnlargeDialog(parent, watermark, "Watermark Image");

      this.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
   } // End of else if statement
   else if (evt.getSource() == clearButton){
      if (image != null) image.flush();
      if (watermark != null) watermark.flush();

      image = null;
      watermark = null;
      if (parent.canSelect()) selectButton.setEnabled(true);
      else selectButton.setEnabled(false);

      imageRadioButton.setEnabled(false);
      watermarkRadioButton.setEnabled(false);
      enlargeButton.setEnabled(false);
      clearButton.setEnabled(false);
      insertionButton.setEnabled(false);

      this.repaint();
   } // End of else if statement
   else if (evt.getSource() == extractionButton){
      extractWatermark();
   } // End of if statement
   else if (evt.getSource() == insertionButton){
      insertWatermark();
   } // End of else if statement
} // End of public void actionPerformed(ActionEvent evt)

// Abstract method for watermark insertion, must be implemented by
// the inherited class
public abstract void insertWatermark();
```

```
   // Abstract method for watermark extraction, must be implemented by
   // the inherited class
   public abstract void extractWatermark();
} // End of public abstract class DisplayArea extends JPanel implements
  // ActionListener
```

B.4 ElapsedTimerTask.java

```
// ElapsedTimerTask.java
//
// Elapsed Timer Task class is used to calculate the total elapsed time
// showing in 00:00:00.000 (hours, minutes, seconds, milliseconds)

import javax.swing.JTextField;
import java.util.TimerTask;

public class ElapsedTimerTask extends TimerTask{
   protected JTextField textField;
   protected long startTime = 0;

   // Constructor: Initialize the JTextField component where the update
   // will take place
   public ElapsedTimerTask(JTextField pTextField){
      textField = pTextField;
   } // End of public ElapsedTimerTask(JTextField pTextField)

   public void run(){
      if (startTime == 0) startTime = System.currentTimeMillis();
      long elapsed = System.currentTimeMillis() - startTime;

      int hours = (int)((elapsed / 1000) / 3600);
      int minutes = (int)(((elapsed / 1000) % 3600) / 60);
      int seconds = (int)((((elapsed / 1000) % 3600) % 60));
      int millis = (int)(elapsed % 1000);

      String elapsedString = new String();

      if (hours < 10) elapsedString += "0" + hours + ":";
      else elapsedString += hours + ":";

      if (minutes < 10) elapsedString += "0" + minutes + ":";
      else elapsedString += minutes + ":";

      if (seconds < 10) elapsedString += "0" + seconds + ".";
      else elapsedString += seconds + ".";

      for (int i = String.valueOf(millis).length(); i < 3; i++)
         elapsedString += "0";
      elapsedString += millis;

      textField.setText(elapsedString);
   } // End of public void run()
} // End of public class ElpasedTimerTask extends TimerTask
```

B.5 EnlargeDialog.java

```
// EnlargeDialog.java
//
// Enlarge Dialog is used to display the enlarged version of the image to
// be displayed providing means of seen the subdivision of the blocks and
// scaling the image to the desired size.

import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import javax.imageio.*;
import java.io.*;

public class EnlargeDialog extends JDialog implements ActionListener,
                                                      ChangeListener{
   private static final String DEFAULT_SAVE_DIR = "saved/";
   private static final double DISPLAY_PERCENTAGE = 0.95;
   private static final int BLOCK_WIDTH = 8;
   private static final int BLOCK_HEIGHT = 8;

   private JSlider slider;
   private JCheckBox dashedCheckBox, blockCheckBox;
   private JLabel colorLabel, imageLabel;
   private JButton changeColorButton, saveButton, closeButton;
   private JPanel bottomPanel;
   private ImagePanel imagePanel;
   private BufferedImage image;
   private JFrame parent;
   private static Color blockColor = Color.RED;

   public EnlargeDialog(JFrame pParent, BufferedImage pImage, String title){
      super(pParent, true);
      parent = pParent;
      image = pImage;

      this.setTitle(title + " (" + image.getWidth() + " x " +
                    image.getHeight() + ")");

      GraphicsEnvironment env;
      env = GraphicsEnvironment.getLocalGraphicsEnvironment();
      Rectangle desktopSize = env.getMaximumWindowBounds();

      this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
      this.setSize((int)(desktopSize.getWidth() * DISPLAY_PERCENTAGE),
                   (int)(desktopSize.getHeight() * DISPLAY_PERCENTAGE));

      imagePanel = new ImagePanel(true);

      slider = new JSlider(JSlider.HORIZONTAL);
      slider.setMajorTickSpacing(5);
      slider.setMinorTickSpacing(1);
      slider.setPaintTicks(true);

      blockCheckBox = new JCheckBox("Show Block Partitions", true);
      dashedCheckBox = new JCheckBox("Dashed Lines", true);
      colorLabel = new JLabel("               ");
      colorLabel.setBackground(blockColor);
      colorLabel.setOpaque(true);
      colorLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 2));

      changeColorButton = new JButton("Change Color");
      saveButton = new JButton("Save Image");
      saveButton.setMnemonic(KeyEvent.VK_S);
      closeButton = new JButton("Close Window");
      closeButton.setMnemonic(KeyEvent.VK_C);

      JPanel buttonsPanel = new JPanel();
      buttonsPanel.setLayout(new FlowLayout());
      buttonsPanel.add(blockCheckBox);
      buttonsPanel.add(dashedCheckBox);
      buttonsPanel.add(colorLabel);
      buttonsPanel.add(changeColorButton);
      buttonsPanel.add(saveButton);
      buttonsPanel.add(closeButton);

      bottomPanel = new JPanel();
      bottomPanel.setLayout(new BorderLayout());
      bottomPanel.add(slider, BorderLayout.NORTH);
      bottomPanel.add(buttonsPanel, BorderLayout.SOUTH);

      this.setLayout(null);
      this.add(imagePanel);
      this.add(bottomPanel);

      int bHeight = (int)bottomPanel.getPreferredSize().getHeight();

      imagePanel.setBounds(5, 5, this.getWidth() - 20,
                           this.getHeight() - bHeight - 45);
      bottomPanel.setBounds(5, this.getHeight() - bHeight - 35,
                            this.getWidth() - 20, bHeight);

      float percentage = imagePanel.calculateMaximumPercentage(image);
      slider.setMaximum((int)(percentage * 100));
      slider.setMinimum(1);
      slider.setValue((int)(percentage * 100));

      imagePanel.setImage(this.getBlockedImage(image));

      // Registering listeners
      slider.addChangeListener(this);
      blockCheckBox.addActionListener(this);
      dashedCheckBox.addActionListener(this);
      changeColorButton.addActionListener(this);
      saveButton.addActionListener(this);
      closeButton.addActionListener(this);

      this.center();
      this.setResizable(false);
      this.setVisible(true);
   } // End of public EnlargeDialog(JFrame pParent, BufferedImage pImage,
     //                             String title)

   // Centers this dialog respective to the desktop space
   private void center(){
      Rectangle desktopSize = GraphicsEnvironment.getLocalGraphicsEnvironment().
                               getMaximumWindowBounds();
      this.setLocation((int)((desktopSize.getWidth() - this.getWidth()) / 2),
                       (int)((desktopSize.getHeight() - this.getHeight()) / 2));
   } // End of private void center()

   public void dispose(){
      super.dispose();
      if (image != null) image.flush();
   } // End of public void dispose()

   // Returns a BufferedImage divided into blocks
   private BufferedImage getBlockedImage(BufferedImage source){
      BufferedImage img = new BufferedImage(source.getWidth(),
                                            source.getHeight(),
                                            BufferedImage.TYPE_INT_RGB);
      Graphics2D graphics2D = img.createGraphics();
      graphics2D.drawImage(source, 0, 0, null);
      graphics2D.setColor(blockColor);

      for (int i = 0; i < source.getWidth(); i = i + BLOCK_WIDTH){
         if (dashedCheckBox.isSelected())
            this.drawDashedLine(graphics2D, i, 0, i, source.getHeight(), 3, 3);
         else
            graphics2D.drawLine(i, 0, i, source.getHeight());
      } // End of for loop

      for (int j = 0; j < source.getHeight(); j = j + BLOCK_HEIGHT){
         if (dashedCheckBox.isSelected())
            this.drawDashedLine(graphics2D, 0, j, source.getWidth(), j, 3, 3);
         else
            graphics2D.drawLine(0, j, source.getWidth(), j);
      } // End of for loop

      graphics2D.dispose();
      return img;
   } // End of public static BufferedImage getBlockedImage(BufferedImage source)

   // Uses the graphics object to draw dash lines
   private void drawDashedLine(Graphics2D g,int x1,int y1,int x2,int y2,
                               double dashlength, double spacelength){
      if ((x1 == x2) && (y1 == y2)){
         g.drawLine(x1, y1, x2, y2);
         return;
      } // End of if statement

      double linelength = Math.sqrt((x2 - x1) * (x2 - x1) +
                                    (y2 - y1) * (y2 - y1));
      double yincrement = (y2 - y1) / (linelength /
                          (dashlength + spacelength));
      double xincdashspace = (x2 - x1) / (linelength /
                             (dashlength + spacelength));
      double yincdashspace = (y2 - y1) / (linelength /
                             (dashlength + spacelength));
      double xincdash = (x2 - x1) / (linelength / (dashlength));
      double yincdash = (y2 - y1) / (linelength / (dashlength));

      int counter=0;
      for (double i = 0;
           i < linelength - dashlength;
           i += dashlength + spacelength){
         g.drawLine((int)(x1 + xincdashspace * counter),
                    (int)(y1 + yincdashspace * counter),
                    (int)(x1 + xincdashspace * counter + xincdash),
                    (int)(y1 + yincdashspace * counter + yincdash));
         counter++;
      } // End of for loop

      if ((dashlength + spacelength) * counter <= linelength){
         g.drawLine((int)(x1 + xincdashspace * counter),
                    (int)(y1 + yincdashspace * counter),
                     x2, y2);
      } // End of if statement
   } // End of public static void drawDashedLine(Graphics g, int x1, int y1,
     //                                      int x2, int y2, double dl,
     //                                      double sl)

   // Action Listener event handlers
   public void actionPerformed(ActionEvent evt){
      if (evt.getSource() == blockCheckBox){
         if (blockCheckBox.isSelected()){
            imagePanel.setImage(this.getBlockedImage(image),
                               (float)(slider.getValue() / 100.0));
            dashedCheckBox.setEnabled(true);
            colorLabel.setEnabled(true);
            changeColorButton.setEnabled(true);
         } // End of if statement
         else{
            imagePanel.setImage(image, (float)(slider.getValue() / 100.0));
            dashedCheckBox.setEnabled(false);
            colorLabel.setEnabled(false);
            changeColorButton.setEnabled(false);
         } // End of else statement
      } // End of if statement
      else if (evt.getSource() == dashedCheckBox){
         imagePanel.setImage(this.getBlockedImage(image),
                            (float)(slider.getValue() / 100.0));
      } // End of if statement
      else if (evt.getSource() == changeColorButton){
         Color newColor = JColorChooser.showDialog(this,
                          "Choose Line Color", blockColor);
        if (newColor != null){
            blockColor = newColor;
            colorLabel.setBackground(blockColor);

            imagePanel.setImage(this.getBlockedImage(image),
                                (float)(slider.getValue() / 100.0));
        } // End of if statement
      } // End of else if statement
      else if (evt.getSource() == saveButton){
         JFileChooser fileChooser = null;

         try{
            fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
         } // End of try statement
         catch(Exception e){
             fileChooser = new JFileChooser(System.getProperty("user.dir"));
         } // End of catch statement

         fileChooser.setAcceptAllFileFilterUsed(false);
         fileChooser.setFileFilter(new PNGFilter());

         int result = fileChooser.showSaveDialog(null);

         if (result == JFileChooser.APPROVE_OPTION){
            String filePath = fileChooser.getSelectedFile().getAbsolutePath();

            if (!filePath.toLowerCase().endsWith(".png")) filePath += ".png";

            result = JOptionPane.YES_OPTION;

            if (new File(filePath).exists()){
               Toolkit.getDefaultToolkit().beep();
               result = JOptionPane.showConfirmDialog(null, "Do you want to " +
                            "replace the existing file?", "File exists",
                            JOptionPane.YES_NO_OPTION);
            } // End of if statement

            if (result == JOptionPane.YES_OPTION){
               fileChooser.setCursor(new Cursor(Cursor.WAIT_CURSOR));
```

```
               try{
                  ImageIO.write(imagePanel.getImage(), "PNG",
                                new File(filePath));
               } // End of try statement
               catch(Exception ex){
                  Toolkit.getDefaultToolkit().beep();
                  JOptionPane.showMessageDialog(null, "An error occurred " +
                            "while saving the requested image.", "I/O Error",
                            JOptionPane.ERROR_MESSAGE);
               } // end of catch statement
               finally{
                  fileChooser.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
               } // End of finally
            } // End of if statement
         } // End of if statement
      } // End of else if statement
      else if (evt.getSource() == closeButton){
         this.setVisible(false);
         this.dispose();
      } // end of else statement
   } // End of public void actionPerformed(ActionEvent evt)

   // Change Listeners event handlers
   public void stateChanged(ChangeEvent evt){
      if (evt.getSource() == slider){
         JSlider source = (JSlider)evt.getSource();
         if (!source.getValueIsAdjusting()){
            if (blockCheckBox.isSelected()){
               imagePanel.setImage(this.getBlockedImage(image),
                                  (float)(source.getValue() / 100.0));
            } // End of if statement
            else{
               imagePanel.setImage(image, (float)(source.getValue() / 100.0));
            } // End of else statement
         } // End of if statement
      } // End of if statement
   } // End of public void stateChanged(ChangeEvent evt)
} // End of public class EnlargeDialog extends JDialog
```

B.6 ExtractionDialog.java

```
// ExtractionDialog.java
//
// Extraction Dialog is the dialog used to represent the extraction process
// of the watermarked image.

import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.math.*;
import java.io.*;
import javax.imageio.*;

public class ExtractionDialog extends JDialog implements ActionListener{
   public static final int BLOCK_WIDTH = 8;
   public static final int BLOCK_HEIGHT = 8;
   private static final Color CAPTION_BACKGROUND = new Color(184, 207, 229);
   private static final double DISPLAY_PERCENTAGE = 0.95;
   private static final String DEFAULT_SAVE_DIR = "saved/";
   private static final String DEFAULT_KEYS_DIR = "keys/";

   protected JLabel imageLabel, dLabel, nLabel, blockLabel, hashLabel,
                    elapsedLabel;
   protected JTextField imageTextField, dTextField, nTextField;
   public JTextField blockTextField, hashTextField, elapsedTextField;
   protected JButton imageButton, keyButton, saveButton, startButton,
                     closeButton;
   public JProgressBar progress;
   protected ImagePanel imagePanel;
   protected ImageIntegrity parent;
   protected String algorithm;
   public BufferedImage image;
   protected ImageInfo info = null;
protected BigInteger d = BigInteger.ZERO, n = BigInteger.ZERO;
protected int totalBlocksCount, blockWidthCount, blockHeightCount;
protected Thread extractionThread = null;

public ExtractionDialog(ImageIntegrity pParent, String alg){
   super(pParent, true);
   parent = pParent;
   algorithm = alg;

   Rectangle desktopSize = GraphicsEnvironment.
                  getLocalGraphicsEnvironment().getMaximumWindowBounds();

   this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
   this.setSize((int)(desktopSize.getWidth() * DISPLAY_PERCENTAGE),
             (int)(desktopSize.getHeight() * DISPLAY_PERCENTAGE));
   this.setTitle("Watermark Extraction - " + algorithm);

   imagePanel = new ImagePanel(true);

   imageLabel = new JLabel("Image File: ", JLabel.RIGHT);
   imageLabel.setBackground(CAPTION_BACKGROUND);
   imageLabel.setOpaque(true);
   imageTextField = new JTextField(20);
   imageTextField.setEditable(false);

   dLabel = new JLabel("Public Key D: ", JLabel.RIGHT);
   dLabel.setBackground(CAPTION_BACKGROUND);
   dLabel.setOpaque(true);
   dTextField = new JTextField(10);
   dTextField.setEditable(false);

   nLabel = new JLabel("Public Key N: ", JLabel.RIGHT);
   nLabel.setBackground(CAPTION_BACKGROUND);
   nLabel.setOpaque(true);
   nTextField = new JTextField(10);
   nTextField.setEditable(false);

   progress = new JProgressBar();
   progress.setStringPainted(true);

   blockLabel = new JLabel("Block Status: ", JLabel.RIGHT);
   blockLabel.setOpaque(true);
   blockLabel.setBackground(CAPTION_BACKGROUND);
   blockTextField = new JTextField(20);
   blockTextField.setEditable(false);

   hashLabel = new JLabel("MD5 Hash Key: ", JLabel.RIGHT);
   hashLabel.setOpaque(true);
   hashLabel.setBackground(CAPTION_BACKGROUND);
   hashTextField = new JTextField(20);
   hashTextField.setEditable(false);

   elapsedLabel = new JLabel("Elapsed Time: ", JLabel.RIGHT);
   elapsedLabel.setOpaque(true);
   elapsedLabel.setBackground(CAPTION_BACKGROUND);
   elapsedTextField = new JTextField("00:00:00.000");
   elapsedTextField.setHorizontalAlignment(JTextField.CENTER);
   elapsedTextField.setEditable(false);

   imageButton = new JButton("Open Image...");
   imageButton.setMnemonic(KeyEvent.VK_O);

   keyButton = new JButton("Load Public Key...");
   keyButton.setMnemonic(KeyEvent.VK_L);

   saveButton = new JButton("Save Image...");
   saveButton.setEnabled(false);

   startButton = new JButton("Start Process");
   startButton.setMnemonic(KeyEvent.VK_S);
   startButton.setEnabled(false);

   closeButton = new JButton("Close Window");
   closeButton.setMnemonic(KeyEvent.VK_C);

   this.setLayout(null);
   this.add(imagePanel);
   this.add(imageLabel);
   this.add(imageTextField);
   this.add(dLabel);
   this.add(dTextField);
   this.add(nLabel);
   this.add(nTextField);
   this.add(progress);
   this.add(blockLabel);
   this.add(blockTextField);
   this.add(hashLabel);
   this.add(hashTextField);
   this.add(elapsedLabel);
   this.add(elapsedTextField);
   this.add(imageButton);
   this.add(keyButton);
   this.add(saveButton);
   this.add(startButton);
   this.add(closeButton);

   int width = this.getWidth() - 20;
   int height = this.getHeight() - 40;

   imageButton.setBounds(10, 10, 135, 25);
   imageLabel.setBounds(150, 12, 100, 20);
   imageTextField.setBounds(255, 12, width - 255, 20);
   keyButton.setBounds(10, 40, 135, 25);
   dLabel.setBounds(150, 42, 100, 20);
   dTextField.setBounds(255, 42, (width - 255 - 100) / 2 - 5, 20);
   nLabel.setBounds((width - 255 - 100) / 2 + 255, 42, 100, 20);
   nTextField.setBounds((width - 255 - 100) / 2 + 360, 42,
                        (width - 255 - 100) / 2 - 5, 20);
   imagePanel.setBounds(5, 65, width, height - 30 - 120);
   progress.setBounds(10, height - 80, width - 10, 20);
   blockLabel.setBounds(10, height - 55, 100, 20);
   blockTextField.setBounds(115, height - 55, width - 445, 20);
   hashLabel.setBounds(10, height - 30, 100, 20);
   hashTextField.setBounds(115, height - 30, width - 445, 20);
   elapsedLabel.setBounds(width - 320, height - 55, 95, 20);
   elapsedTextField.setBounds(width - 220, height - 55, 95, 20);
   saveButton.setBounds(width - 120 - 120 - 5, height - 25, 120, 25);
   startButton.setBounds(width - 120, height - 55, 120, 25);
   closeButton.setBounds(width - 120, height - 25, 120, 25);

   // Registering listeners
   imageButton.addActionListener(this);
   keyButton.addActionListener(this);
   saveButton.addActionListener(this);
   startButton.addActionListener(this);
   closeButton.addActionListener(this);

   this.center();
   this.setResizable(false);
   this.setVisible(true);
} // End of public ExtractionDialog(ImageIntegrity pParent, String alg)

// Centers the dialog respective to the desktop space
private void center(){
   Rectangle desktopSize = GraphicsEnvironment.getLocalGraphicsEnvironment().
                            getMaximumWindowBounds();
   this.setLocation((int)((desktopSize.getWidth() - this.getWidth()) / 2),
                    (int)((desktopSize.getHeight() - this.getHeight()) / 2));
} // End of private void center()

public void dispose(){
   super.dispose();
   if (extractionThread != null) extractionThread.interrupt();
   if (image != null) image.flush();
} // End of public void dispose()

// Returns the number of total blocks in the image to be watermarked
public int getTotalBlocksCount(){
   return totalBlocksCount;
} // End of public int getTotalBlocksCount()

// Returns the number of blocks calculated from the width of the image
public int getBlockWidthCount(){
   return blockWidthCount;
} // End of public int getBlockWidthCount()

// Returns the number of blocks calculated from the height of the image
public int getBlockHeightCount(){
   return blockHeightCount;
} // End of public int getBlockHeightCount

// Updates the image to be displayed
```

```
public void updateImage(){
   imagePanel.setImage(image);
} // end of public void updateImage()

// Activates the save button
public void activateSaveButton(){
   saveButton.setEnabled(true);
   startButton.setEnabled(true);
} // end of public void activateSaveButton()

// Action Listeners event handlers
public void actionPerformed(ActionEvent evt){
   if (evt.getSource() == imageButton){
      JFileChooser fileChooser;

      try{
         fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
      } // End of try statement
      catch(Exception e){
         fileChooser = new JFileChooser(System.getProperty("user.dir"));
      } // End of catch statement

      fileChooser.setDialogTitle("Open Watermarked Image");
      fileChooser.setAcceptAllFileFilterUsed(false);
      fileChooser.setFileFilter(new PNGFilter());

      int result = fileChooser.showOpenDialog(this);

      if (result == JFileChooser.APPROVE_OPTION){
         String path = fileChooser.getSelectedFile().getAbsolutePath();
         info = new ImageInfo(path);
         image = info.getBufferedImage();

         this.setTitle("Watermark Extraction - " + algorithm + " (" +
                       image.getWidth() + " x " + image.getHeight() + ")");
         imageTextField.setText(info.getPath());
         imagePanel.setImage(image);

         blockWidthCount = image.getWidth() / BLOCK_WIDTH;
         blockHeightCount = image.getHeight() / BLOCK_HEIGHT;

         if (image.getWidth() % BLOCK_WIDTH != 0) blockWidthCount++;
         if (image.getHeight() % BLOCK_HEIGHT != 0) blockHeightCount++;
         totalBlocksCount = blockWidthCount * blockHeightCount;

         progress.setMinimum(0);
         progress.setMaximum(totalBlocksCount);
         progress.setValue(0);
      } // End of if statement

      if (imagePanel.getImage() != null && !d.equals(BigInteger.ZERO) &&
          !n.equals(BigInteger.ZERO)){
         startButton.setEnabled(true);
      } // End of if statmenet
      else{
         startButton.setEnabled(false);
      } // End of else statement
   } // End of if statement
   else if (evt.getSource() == keyButton){
      JFileChooser fc;
      FileReader fileReader;
      BufferedReader reader;
      int result;

      try{
         fc = new JFileChooser(DEFAULT_KEYS_DIR);
      } // End of try statement
      catch(Exception e){
         fc = new JFileChooser(System.getProperty("user.dir"));
      } // End of catch statmenet

      fc.setDialogTitle("Load Public Key");
      fc.setAcceptAllFileFilterUsed(false);
      fc.setFileFilter(new PublicFilter());

      result = fc.showOpenDialog(this);

      if (result == JFileChooser.APPROVE_OPTION){
         d = BigInteger.ZERO;
         n = BigInteger.ZERO;
         try{
```

```
         fileReader = new FileReader(fc.getSelectedFile());
         reader = new BufferedReader(fileReader);
         d = new BigInteger(reader.readLine());
         n = new BigInteger(reader.readLine());
         dTextField.setText(d.toString());
         nTextField.setText(n.toString());
         reader.close();
      } // End of try statement
      catch(Exception e){
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(this,
            "An error occurred while reading the public key file.",
            "I/O Error", JOptionPane.ERROR_MESSAGE);
      } // End of catch statement
   } // End of if statemnet

   if (imagePanel.getImage() != null && !d.equals(BigInteger.ZERO) &&
       !n.equals(BigInteger.ZERO)){
      startButton.setEnabled(true);
   } // End of if statmenet
   else{
      startButton.setEnabled(false);
   } // End of else statement
} // end of else if statement
else if (evt.getSource() == saveButton){
   JFileChooser fileChooser = null;

   try{
      fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
   } // End of try statement
   catch(Exception e){
       fileChooser = new JFileChooser(System.getProperty("user.dir"));
   } // End of catch statement

   fileChooser.setAcceptAllFileFilterUsed(false);
   fileChooser.setFileFilter(new PNGFilter());

   int result = fileChooser.showSaveDialog(null);

   if (result == JFileChooser.APPROVE_OPTION){
      String filePath = fileChooser.getSelectedFile().getAbsolutePath();

      if (!filePath.toLowerCase().endsWith(".png")) filePath += ".png";

      result = JOptionPane.YES_OPTION;

      if (new File(filePath).exists()){
         Toolkit.getDefaultToolkit().beep();
         result = JOptionPane.showConfirmDialog(null, "Do you want to " +
                     "replace the existing file?", "File exists",
                     JOptionPane.YES_NO_OPTION);
      } // End of if statement

      if (result == JOptionPane.YES_OPTION){
         fileChooser.setCursor(new Cursor(Cursor.WAIT_CURSOR));

         try{
            ImageIO.write(image, "PNG", new File(filePath));
         } // End of try statement
         catch(Exception ex){
            Toolkit.getDefaultToolkit().beep();
            JOptionPane.showMessageDialog(null, "An error occurred " +
                     "while saving the requested image.", "I/O Error",
                     JOptionPane.ERROR_MESSAGE);
         } // end of catch statement
         finally{
            fileChooser.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
         } // End of finally
      } // End of if statement
   } // End of if statement
} // end of else if statement
else if (evt.getSource() == startButton){
   image = info.getBufferedImage();
   progress.setValue(0);
   saveButton.setEnabled(false);
   this.updateImage();

   if (algorithm.equals("Wong's Algorithm")){
      extractionThread = new WongExtractionThread(this, d, n);
      extractionThread.setPriority(Thread.NORM_PRIORITY);
      extractionThread.start();
```

```
         } // End of if statement
         else if (algorithm.equals("HBC1 Algorithm")){
            extractionThread = new HBC1ExtractionThread(this, d, n);
            extractionThread.setPriority(Thread.NORM_PRIORITY);
            extractionThread.start();
         } // End of else if statement
         else if (algorithm.equals("HBC2 Algorithm")){
            extractionThread = new HBC2ExtractionThread(this, d, n);
            extractionThread.setPriority(Thread.NORM_PRIORITY);
            extractionThread.start();
         } // End of else if statement
         else if (algorithm.equals("Proposed Algorithm")){
            extractionThread = new ProposedExtractionThread(this, d, n);
            extractionThread.setPriority(Thread.NORM_PRIORITY);
            extractionThread.start();
         } // End of else if statement
         else{
            JOptionPane.showMessageDialog(parent, "Invalid Extraction " +
                 "Algorithm", "Algorithm Error", JOptionPane.ERROR_MESSAGE);
         } // End of else statement

         startButton.setEnabled(false);
      } // End of else if statemetn
      else if (evt.getSource() == closeButton){
         if (extractionThread != null) extractionThread.interrupt();

         this.setVisible(false);
         this.dispose();
      } // End of else if statement
   } // End of public void actionPerformed(ActionEvent evt)
} // End of public class ExtractionDialog extends JDialog
```

B.7 HBC1Algorithm.java

```
// HBC1Algorithm.java
//
// Base class DisplayArea, provides the visual interface of the display area.
// In addition, insertion and extraction will be done using the Hash Block
// Chaining 1 (HBC1) algorithm

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.math.*;

public class HBC1Algorithm extends DisplayArea{
   public HBC1Algorithm(ImageIntegrity pParent){
      super(pParent);    // call base class constructor
      ALGORITHM_TEXT = "HASH BLOCK CHAINING 1";
   } // End of public HBC1Algorithm(ImageIntegrity pParent)

   // Insers the watermark
   public void insertWatermark(){
      BigInteger e = parent.getE();
      BigInteger n = parent.getN();

      if (e == null || n == null){
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(parent, "Insertion process cannot be " +
                "performed because private key is missing", "Missing Key",
                JOptionPane.ERROR_MESSAGE);
      } // End of if statement
      else{
         new InsertionDialog(parent, image, watermark, e, n, "HBC1 Algorithm");
      } // End of else statmenet
   } // End of public void insertWatermark()

   // Extracts the watermark from the watermarked image
   public void extractWatermark(){
      new ExtractionDialog(parent, "HBC1 Algorithm");
   } // end of public void extractWatermark()
} // End of public class HBC1Algorithm extends DisplayArea
```

B.8 HBC1ExtractionThread.java

```
// HBC1ExtractionThread.java
//
// Thread to extract the watermark from the watermarked image using the HBC1
// extraction algorithm

import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.math.BigInteger;
import java.util.Timer;
import java.io.*;

public class HBC1ExtractionThread extends Thread{
   protected ExtractionDialog dialog;
   protected RSACryptoSystem RSA;
   protected Timer elapsedTimer = new Timer();

   public HBC1ExtractionThread(ExtractionDialog pDialog, BigInteger pD,
                               BigInteger pN){
      dialog = pDialog;
      RSA = new RSACryptoSystem();
      RSA.setPublicKey(pD, pN);
   } // End of public HBC1ExtractionThread(ExtractionDialog pDialog,
     //                                    BigInteger pD, BigInteger pN)

   public void run(){
      int imgWidth = dialog.image.getWidth();
      int imgHeight = dialog.image.getHeight();
      int blockWidth = 0, blockHeight = 0, totalBlocks = 0;
      int imageRGB, red, green, blue, blockIndex = 0;
      String zeroedString, zeroedStringPrev, hashString, encodedString,
             binaryDigest, decodedBinary, xorString;
      BigInteger msg = null, dec = null, xor = null;

      elapsedTimer.schedule(new ElapsedTimerTask(dialog.elapsedTextField),
                            0, 1);

      // Obtaining the previous block for the first block
      int rowStart_prev = imgWidth - (imgWidth % dialog.BLOCK_WIDTH);
      int colStart_prev = imgHeight - (imgHeight % dialog.BLOCK_HEIGHT);

      if (rowStart_prev == imgWidth) rowStart_prev -= dialog.BLOCK_WIDTH;
      if (colStart_prev == imgHeight) colStart_prev -= dialog.BLOCK_HEIGHT;

      zeroedStringPrev = new String();
      for (int y_prev = colStart_prev; y_prev < imgHeight; y_prev++){
         for (int x_prev = rowStart_prev; x_prev < imgWidth; x_prev++){
            imageRGB = dialog.image.getRGB(x_prev, y_prev);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            if (blue % 2 == 1) blue--;    // Zero blue LSB

            zeroedStringPrev += String.valueOf(red) +
                                String.valueOf(green) +
                                String.valueOf(blue);
         } // End of for loop
      } // End of for loop

      for (int y = 0; y < imgHeight; y = y + dialog.BLOCK_HEIGHT){
         for (int x = 0; x < imgWidth && !this.isInterrupted();
              x = x + dialog.BLOCK_WIDTH){
            blockIndex++;
            dialog.blockTextField.setText(blockIndex + " of " +
                                          dialog.getTotalBlocksCount());

            blockWidth = Math.min(dialog.BLOCK_WIDTH, imgWidth - x);
            blockHeight = Math.min(dialog.BLOCK_HEIGHT, imgHeight - y);
            totalBlocks = blockWidth * blockHeight;

            zeroedString = new String();
            encodedString = new String();

            for (int j = 0; j < blockHeight; j++){
               for (int i = 0; i < blockWidth; i++){
```

```
            imageRGB = dialog.image.getRGB(x + i, y + j);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            if (blue % 2 == 1){
               blue--;
               encodedString += "1";
            } // End of if statement
            else{
               encodedString += "0";
            } // End of else statement

            zeroedString += String.valueOf(red) +
                            String.valueOf(green) +
                            String.valueOf(blue);
         } // End of for loop
      } // End of for loop

      hashString = imgWidth + imgHeight + zeroedString +
                   zeroedStringPrev + blockIndex;
      zeroedStringPrev = zeroedString;    // Saving current zeroed string
                                          // for next iteration
      dialog.hashTextField.setText(MD5Digest.getHexDigest(hashString));

      binaryDigest = MD5Digest.getBinDigest(hashString);
      binaryDigest = binaryDigest.substring(0, totalBlocks);
      msg = new BigInteger(encodedString, 2);

      if (totalBlocks < (dialog.BLOCK_WIDTH * dialog.BLOCK_HEIGHT) ||
          msg.compareTo(RSA.getN()) > 0){
         decodedBinary = encodedString;
      } // End of if statement
      else{ // Decode message
         dec = RSA.decrypt(msg);
         decodedBinary = dec.toString(2);

         // Padding with leading zeroes
         for (int k = decodedBinary.length(); k < totalBlocks; k++)
            decodedBinary = "0" + decodedBinary;
      } // End of if statement

      xor = new BigInteger(binaryDigest, 2).xor(
                                    new BigInteger(decodedBinary, 2));
      xorString = xor.toString(2);

      // Pad with leading zeroes
      for (int k = xorString.length(); k < totalBlocks; k++)
         xorString = "0" + xorString;

      int index = 0;
      for (int jj = 0; jj < blockHeight; jj++){
         for (int ii = 0; ii < blockWidth; ii++){
            char c = xorString.charAt(index);

            if (c == '1')
               dialog.image.setRGB(x + ii, y + jj, Color.WHITE.getRGB());
            else
               dialog.image.setRGB(x + ii, y + jj, Color.BLACK.getRGB());
            index++;
         } // End of for loop
      } // End of for loop

      dialog.progress.setValue(dialog.progress.getValue() + 1);

      if (blockIndex % dialog.getBlockWidthCount() == 0){
         dialog.updateImage();
      } // End of if statement
   } // End of for loop
} // End of for loop

elapsedTimer.cancel();

if (!this.isInterrupted()){
   dialog.activateSaveButton();
   Toolkit.getDefaultToolkit().beep();
   JOptionPane.showMessageDialog(dialog, "HBC1 watermark extraction " +
                              "process was completed successfully.",
                              "Success", JOptionPane.INFORMATION_MESSAGE);
} // End of if statement
```

```
   } // end of public void run()
} // End of public class HBC1ExtractionThread extends Thread
```

B.9 HBC1InsertionThread.java

```
// HBC1InsertionThread.java
//
// Thread to insert the watermark into the cover image using the HBC1
// insertion algorithm

import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.math.BigInteger;
import java.util.Timer;
import java.io.*;

public class HBC1InsertionThread extends Thread{
   protected InsertionDialog dialog;
   protected RSACryptoSystem RSA;
   protected Timer elapsedTimer = new Timer();

   public HBC1InsertionThread(InsertionDialog pDialog, BigInteger pE,
                              BigInteger pN){
      dialog = pDialog;
      RSA = new RSACryptoSystem();
      RSA.setPrivateKey(pE, pN);
   } // End of public HBC1InsertionThread(InsertionDialog pDialog,
     //                                  BigInteger pE, BigInteger pN)

   public void run(){
      int imgWidth = dialog.image.getWidth();
      int imgHeight = dialog.image.getHeight();
      int blockWidth = 0, blockHeight = 0, totalBlocks = 0;
      int red, green, blue, blockIndex = 0;
      int imageRGB, waterRGB;
      String zeroedString, zeroedStringPrev, waterString, binaryDigest,
             encodedBinary, hashString;
      BigInteger msg = null, enc = null;
      int count = 0;

      elapsedTimer.schedule(new ElapsedTimerTask(dialog.elapsedTextField),
                            0, 1);

      // Obtaining the previous block for the first block
      int rowStart_prev = imgWidth - (imgWidth % dialog.BLOCK_WIDTH);
      int colStart_prev = imgHeight - (imgHeight % dialog.BLOCK_HEIGHT);

      if (rowStart_prev == imgWidth) rowStart_prev -= dialog.BLOCK_WIDTH;
      if (colStart_prev == imgHeight) colStart_prev -= dialog.BLOCK_HEIGHT;

      zeroedStringPrev = new String();
      for (int y_prev = colStart_prev; y_prev < imgHeight; y_prev++){
         for (int x_prev = rowStart_prev; x_prev < imgWidth; x_prev++){
            imageRGB = dialog.image.getRGB(x_prev, y_prev);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            if (blue % 2 == 1) blue--;    // Zero blue LSB

            zeroedStringPrev += String.valueOf(red) +
                                String.valueOf(green) +
                                String.valueOf(blue);
         } // End of for loop
      } // End of for loop

      for (int y = 0; y < imgHeight; y = y + dialog.BLOCK_HEIGHT){
         for (int x = 0; x < imgWidth && !this.isInterrupted();
              x = x + dialog.BLOCK_WIDTH){
            blockIndex++;
            dialog.blockTextField.setText(blockIndex + " of " +
                                          dialog.getTotalBlocksCount());

            blockWidth = Math.min(dialog.BLOCK_WIDTH, imgWidth - x);
```

```
blockHeight = Math.min(dialog.BLOCK_HEIGHT, imgHeight - y);
totalBlocks = blockWidth * blockHeight;

zeroedString = new String();
waterString = new String();

for (int j = 0; j < blockHeight; j++){
   for (int i = 0; i < blockWidth; i++){
      imageRGB = dialog.image.getRGB(x + i, y + j);
      waterRGB = dialog.watermark.getRGB(x + i, y + j);

      red = (imageRGB >> 16) & 0xFF;
      green = (imageRGB >> 8) & 0xFF;
      blue = (imageRGB >> 0) & 0xFF;

      if (blue % 2 == 1) blue--;    // Zero blue LSB

      zeroedString += String.valueOf(red) +
                      String.valueOf(green) +
                      String.valueOf(blue);

      if (waterRGB == Color.BLACK.getRGB())
         waterString += "0";
      else if (waterRGB == Color.WHITE.getRGB())
         waterString += "1";
      else
         JOptionPane.showMessageDialog(null, "ERROR WATER COLOR");
   } // End of for loop
} // End of for loop

hashString = imgWidth + imgHeight + zeroedString +
             zeroedStringPrev + blockIndex;
zeroedStringPrev = zeroedString;    // Saving current zeroed string
                                    // for next iteration
dialog.hashTextField.setText(MD5Digest.getHexDigest(hashString));

binaryDigest = MD5Digest.getBinDigest(hashString);
binaryDigest = binaryDigest.substring(0, totalBlocks);

msg = (new BigInteger(binaryDigest, 2)).xor(
                                    new BigInteger(waterString, 2));

if (totalBlocks < (dialog.BLOCK_WIDTH * dialog.BLOCK_HEIGHT) ||
    msg.compareTo(RSA.getN()) > 0){

   encodedBinary = msg.toString(2);

   // Pad with leading zeroes
   for (int k = encodedBinary.length(); k < totalBlocks; k++)
      encodedBinary = "0" + encodedBinary;
} // End of if statement
else{ // Encode message
   enc = RSA.encrypt(msg);
   encodedBinary = enc.toString(2);

   // Pad with leading zeroes
   for (int k = encodedBinary.length(); k < totalBlocks; k++)
      encodedBinary = "0" + encodedBinary;
} // End of if statement

int index = 0;
for (int jj = 0; jj < blockHeight; jj++){
   for (int ii = 0; ii < blockWidth; ii++){
      imageRGB = dialog.image.getRGB(x + ii, y + jj);

      red = (imageRGB >> 16) & 0xFF;
      green = (imageRGB >> 8) & 0xFF;
      blue = (imageRGB >> 0) & 0xFF;

      char c = encodedBinary.charAt(index);
      if (blue % 2 == 0 && c == '1') blue++;
      else if (blue % 2 == 1 && c == '0') blue--;
      dialog.displayImage.setRGB(x + ii, y + jj,
                          new Color(red, green, blue).getRGB());
      index++;
   } // End of for loop
} // End of for loop

dialog.progress.setValue(dialog.progress.getValue() + 1);

if (blockIndex % dialog.getBlockWidthCount() == 0){
```

```
               dialog.updateImage();
            } // End of if statement
         } // End of for loop
      } // End of for loop

      elapsedTimer.cancel();

      if (!this.isInterrupted()){
         dialog.activateSaveButton();
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(dialog, "HBC1 watermark insertion " +
                                "process was completed successfully.",
                                "Success", JOptionPane.INFORMATION_MESSAGE);
      } // End of if statement
   } // End of public void run()
} // End of public class HBC1InsertionThread extends Thread
```

B.10 HBC2Algorithm.java

```
// HBC2Algorithm.java
//
// Base class DisplayArea, provides the visual interface of the display area.
// In addition, insertion and extraction will be done using the Hash Block
// Chaining 2 (HBC2) algorithm

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.math.*;

public class HBC2Algorithm extends DisplayArea{
   public HBC2Algorithm(ImageIntegrity pParent){
      super(pParent);    // call base class constructor
      ALGORITHM_TEXT = "HASH BLOCK CHAINING 2";
   } // End of public HBC2Algorithm(ImageIntegrity pParent)

   // Insers the watermark
   public void insertWatermark(){
      BigInteger e = parent.getE();
      BigInteger n = parent.getN();

      if (e == null || n == null){
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(parent, "Insertion process cannot be " +
                "performed because private key is missing", "Missing Key",
                JOptionPane.ERROR_MESSAGE);
      } // End of if statement
      else{
         new InsertionDialog(parent, image, watermark, e, n, "HBC2 Algorithm");
      } // End of else statmenet
   } // End of public void insertWatermark()

   // Extracts the watermark from the watermarked image
   public void extractWatermark(){
      new ExtractionDialog(parent, "HBC2 Algorithm");
   } // end of public void extractWatermark()
} // End of public class HBC2Algorithm extends DisplayArea
```

B.11 HBC2ExtractionThread.java

```
// HBC2ExtractionThread.java
//
// Thread to extract the watermark from the watermarked image using the HBC2
// extraction algorithm

import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.math.BigInteger;
import java.util.Timer;
import java.io.*;

public class HBC2ExtractionThread extends Thread{
   protected ExtractionDialog dialog;
   protected RSACryptoSystem RSA;
   protected Timer elapsedTimer = new Timer();

   public HBC2ExtractionThread(ExtractionDialog pDialog, BigInteger pD,
                               BigInteger pN){
      dialog = pDialog;
      RSA = new RSACryptoSystem();
      RSA.setPublicKey(pD, pN);
   } // End of public HBC2ExtractionThread(ExtractionDialog pDialog,
     //                                  BigInteger pD, BigInteger pN)

   public void run(){
      int imgWidth = dialog.image.getWidth();
      int imgHeight = dialog.image.getHeight();
      int blockWidth = 0, blockHeight = 0, totalBlocks = 0;
      int imageRGB, red, green, blue, blockIndex = 0;
      String zeroedString, zeroedStringPrev, hashString, encodedString,
             binaryDigest, decodedBinary, xorString, nonDetString, tmpDetString;
      BigInteger msg = null, dec = null, xor = null;

      elapsedTimer.schedule(new ElapsedTimerTask(dialog.elapsedTextField),
                            0, 1);

      // Obtaining the previous block for the first block
      int rowStart_prev = imgWidth - (imgWidth % dialog.BLOCK_WIDTH);
      int colStart_prev = imgHeight - (imgHeight % dialog.BLOCK_HEIGHT);

      if (rowStart_prev == imgWidth) rowStart_prev -= dialog.BLOCK_WIDTH;
      if (colStart_prev == imgHeight) colStart_prev -= dialog.BLOCK_HEIGHT;

      zeroedStringPrev = new String();
      nonDetString = new String();

      for (int y_prev = colStart_prev; y_prev < imgHeight; y_prev++){
         for (int x_prev = rowStart_prev; x_prev < imgWidth; x_prev++){
            imageRGB = dialog.image.getRGB(x_prev, y_prev);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            if (blue % 2 == 1) blue--;    // Zero blue LSB

            zeroedStringPrev += String.valueOf(red) +
                                String.valueOf(green) +
                                String.valueOf(blue);
         } // End of for loop
      } // End of for loop

      for (int y = 0; y < imgHeight; y = y + dialog.BLOCK_HEIGHT){
         for (int x = 0; x < imgWidth && !this.isInterrupted();
               x = x + dialog.BLOCK_WIDTH){
            blockIndex++;
            dialog.blockTextField.setText(blockIndex + " of " +
                                          dialog.getTotalBlocksCount());

            blockWidth = Math.min(dialog.BLOCK_WIDTH, imgWidth - x);
            blockHeight = Math.min(dialog.BLOCK_HEIGHT, imgHeight - y);
            totalBlocks = blockWidth * blockHeight;

            zeroedString = new String();
            encodedString = new String();
            tmpDetString = new String();

            for (int j = 0; j < blockHeight; j++){
               for (int i = 0; i < blockWidth; i++){
                  imageRGB = dialog.image.getRGB(x + i, y + j);

                  red = (imageRGB >> 16) & 0xFF;
                  green = (imageRGB >> 8) & 0xFF;
                  blue = (imageRGB >> 0) & 0xFF;

                  tmpDetString += String.valueOf(red) +
                                  String.valueOf(green) +
                                  String.valueOf(blue);

                  if (blue % 2 == 1){
                     blue--;
                     encodedString += "1";
                  } // End of if statement
                  else{
                     encodedString += "0";
                  } // End of else statement

                  zeroedString += String.valueOf(red) +
                                  String.valueOf(green) +
                                  String.valueOf(blue);
               } // End of for loop
            } // End of for loop

            hashString = imgWidth + imgHeight + zeroedString +
                         zeroedStringPrev + blockIndex + nonDetString;
            zeroedStringPrev = zeroedString;    // Saving current zeroed string
                                                // for next iteration
            dialog.hashTextField.setText(MD5Digest.getHexDigest(hashString));

            binaryDigest = MD5Digest.getBinDigest(hashString);
            binaryDigest = binaryDigest.substring(0, totalBlocks);
            //nonDetString = binaryDigest + tmpDetString;
            nonDetString = encodedString;

            msg = new BigInteger(encodedString, 2);

            if (totalBlocks < (dialog.BLOCK_WIDTH * dialog.BLOCK_HEIGHT) ||
                msg.compareTo(RSA.getN()) > 0){
               decodedBinary = encodedString;
            } // End of if statement
            else{ // Decode message
               dec = RSA.decrypt(msg);
               decodedBinary = dec.toString(2);

               // Padding with leading zeroes
               for (int k = decodedBinary.length(); k < totalBlocks; k++)
                  decodedBinary = "0" + decodedBinary;
            } // End of if statement

            xor = new BigInteger(binaryDigest, 2).xor(
                                              new BigInteger(decodedBinary, 2));
            xorString = xor.toString(2);

            // Pad with leading zeroes
            for (int k = xorString.length(); k < totalBlocks; k++)
               xorString = "0" + xorString;

            int index = 0;
            for (int jj = 0; jj < blockHeight; jj++){
               for (int ii = 0; ii < blockWidth; ii++){
                  char c = xorString.charAt(index);

                  if (c == '1')
                     dialog.image.setRGB(x + ii, y + jj, Color.WHITE.getRGB());
                  else
                     dialog.image.setRGB(x + ii, y + jj, Color.BLACK.getRGB());
                  index++;
               } // End of for loop
            } // End of for loop

            dialog.progress.setValue(dialog.progress.getValue() + 1);

            if (blockIndex % dialog.getBlockWidthCount() == 0){
               dialog.updateImage();
            } // End of if statement
         } // End of for loop
      } // End of for loop

      elapsedTimer.cancel();

      if (!this.isInterrupted()){
         dialog.activateSaveButton();
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(dialog, "HBC2 watermark extraction " +
                                    "process was completed successfully.",
                                    "Success", JOptionPane.INFORMATION_MESSAGE);
      } // End of if statement
   } // end of public void run()
} // End of public class HBC2ExtractionThread extends Thread
```

B.12 HBC2InsertionThread.java

```
// HBC2InsertionThread.java
//
// Thread to insert the watermark into the cover image using the HBC2
// insertion algorithm

import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.math.BigInteger;
import java.util.Timer;
import java.io.*;

public class HBC2InsertionThread extends Thread{
   protected InsertionDialog dialog;
   protected RSACryptoSystem RSA;
   protected Timer elapsedTimer = new Timer();

   public HBC2InsertionThread(InsertionDialog pDialog, BigInteger pE,
                              BigInteger pN){
      dialog = pDialog;
      RSA = new RSACryptoSystem();
      RSA.setPrivateKey(pE, pN);
   } // End of public HBC2InsertionThread(InsertionDialog pDialog,
     //                                    BigInteger pE, BigInteger pN)

   public void run(){
      int imgWidth = dialog.image.getWidth();
      int imgHeight = dialog.image.getHeight();
      int blockWidth = 0, blockHeight = 0, totalBlocks = 0;
      int red, green, blue, blockIndex = 0;
      int imageRGB, waterRGB;
      String zeroedString, zeroedStringPrev, waterString, binaryDigest,
             encodedBinary, hashString, nonDetString;
      BigInteger msg = null, enc = null;
      int count = 0;

      elapsedTimer.schedule(new ElapsedTimerTask(dialog.elapsedTextField),
                            0, 1);

      // Obtaining the previous block for the first block
      int rowStart_prev = imgWidth - (imgWidth % dialog.BLOCK_WIDTH);
      int colStart_prev = imgHeight - (imgHeight % dialog.BLOCK_HEIGHT);

      if (rowStart_prev == imgWidth) rowStart_prev -= dialog.BLOCK_WIDTH;
      if (colStart_prev == imgHeight) colStart_prev -= dialog.BLOCK_HEIGHT;

      zeroedStringPrev = new String();
      nonDetString = new String();
      for (int y_prev = colStart_prev; y_prev < imgHeight; y_prev++){
         for (int x_prev = rowStart_prev; x_prev < imgWidth; x_prev++){
            imageRGB = dialog.image.getRGB(x_prev, y_prev);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            if (blue % 2 == 1) blue--;    // Zero blue LSB

            zeroedStringPrev += String.valueOf(red) +
                                String.valueOf(green) +
                                String.valueOf(blue);
         } // End of for loop
      } // End of for loop

      for (int y = 0; y < imgHeight; y = y + dialog.BLOCK_HEIGHT){
         for (int x = 0; x < imgWidth && !this.isInterrupted();
              x = x + dialog.BLOCK_WIDTH){
            blockIndex++;
            dialog.blockTextField.setText(blockIndex + " of " +
                                          dialog.getTotalBlocksCount());

            blockWidth = Math.min(dialog.BLOCK_WIDTH, imgWidth - x);
            blockHeight = Math.min(dialog.BLOCK_HEIGHT, imgHeight - y);
            totalBlocks = blockWidth * blockHeight;

            zeroedString = new String();
            waterString = new String();
```

```
for (int j = 0; j < blockHeight; j++){
   for (int i = 0; i < blockWidth; i++){
      imageRGB = dialog.image.getRGB(x + i, y + j);
      waterRGB = dialog.watermark.getRGB(x + i, y + j);

      red = (imageRGB >> 16) & 0xFF;
      green = (imageRGB >> 8) & 0xFF;
      blue = (imageRGB >> 0) & 0xFF;

      if (blue % 2 == 1) blue--;    // Zero blue LSB

      zeroedString += String.valueOf(red) +
                      String.valueOf(green) +
                      String.valueOf(blue);

      if (waterRGB == Color.BLACK.getRGB())
         waterString += "0";
      else if (waterRGB == Color.WHITE.getRGB())
         waterString += "1";
      else
         JOptionPane.showMessageDialog(null, "ERROR WATER COLOR");
   } // End of for loop
} // End of for loop

hashString = imgWidth + imgHeight + zeroedString +
             zeroedStringPrev + blockIndex + nonDetString;
zeroedStringPrev = zeroedString;    // Saving current zeroed string
                                    // for next iteration
dialog.hashTextField.setText(MD5Digest.getHexDigest(hashString));

binaryDigest = MD5Digest.getBinDigest(hashString);
binaryDigest = binaryDigest.substring(0, totalBlocks);
//nonDetString = binaryDigest;

msg = (new BigInteger(binaryDigest, 2)).xor(
                                    new BigInteger(waterString, 2));

if (totalBlocks < (dialog.BLOCK_WIDTH * dialog.BLOCK_HEIGHT) ||
    msg.compareTo(RSA.getN()) > 0){

   encodedBinary = msg.toString(2);

   // Pad with leading zeroes
   for (int k = encodedBinary.length(); k < totalBlocks; k++)
      encodedBinary = "0" + encodedBinary;
} // End of if statement
else{ // Encode message
   enc = RSA.encrypt(msg);
   encodedBinary = enc.toString(2);

   // Pad with leading zeroes
   for (int k = encodedBinary.length(); k < totalBlocks; k++)
      encodedBinary = "0" + encodedBinary;
} // End of if statement

nonDetString = encodedBinary;

int index = 0;
for (int jj = 0; jj < blockHeight; jj++){
   for (int ii = 0; ii < blockWidth; ii++){
      imageRGB = dialog.image.getRGB(x + ii, y + jj);

      red = (imageRGB >> 16) & 0xFF;
      green = (imageRGB >> 8) & 0xFF;
      blue = (imageRGB >> 0) & 0xFF;

      char c = encodedBinary.charAt(index);
      if (blue % 2 == 0 && c == '1') blue++;
      else if (blue % 2 == 1 && c == '0') blue--;

      //nonDetString += String.valueOf(red) +
      //                String.valueOf(green) +
      //                String.valueOf(blue);
      dialog.displayImage.setRGB(x + ii, y + jj,
                         new Color(red, green, blue).getRGB());
      index++;
   } // End of for loop
} // End of for loop

dialog.progress.setValue(dialog.progress.getValue() + 1);
```

```
            if (blockIndex % dialog.getBlockWidthCount() == 0){
               dialog.updateImage();
            } // End of if statement
         } // End of for loop
      } // End of for loop

      elapsedTimer.cancel();

      if (!this.isInterrupted()){
         dialog.activateSaveButton();
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(dialog, "HBC2 watermark insertion " +
                                    "process was completed successfully.",
                                    "Success", JOptionPane.INFORMATION_MESSAGE);
      } // End of if statement
   } // End of public void run()
} // End of public class HBC2InsertionThread extends Thread
```

B.13 ImageFilter.java

```
// ImageFilter.java
//
// Image Filter that used in the JFileChooser to filter only the acceptable
// image types.

import java.io.File;
import javax.swing.filechooser.FileFilter;

public class ImageFilter extends FileFilter{
        private static final String ACCEPTED_EXTENSIONS[] = {".jpg", ".jpeg",
                                                              ".jpe", ".png",
                                                              ".gif"};

   // Return true if the file is a valid file determined by its file
   // extension, false otherwise
        public boolean accept(File file){
        if (file.isDirectory()) return true;

        if (acceptFile(file)) return true;
        else return false;
    } // End of public boolean accept(File f)

   // Returns the description of this image filter
   public String getDescription(){
        return "All Image Files (*.gif, *.jpg, *.png)";
   } // End of public String getDescription()

   // Returns true if the file is a valid file with the given extension,
   // false otherwise.
   public static boolean acceptFile(File file){
      boolean found = false;

      if (file.isDirectory()) return false;

        String extension = file.getName().substring(file.getName().
                                              lastIndexOf(".")).toLowerCase();

        for (int i = 0; i < ACCEPTED_EXTENSIONS.length && !found; i++){
           if (ACCEPTED_EXTENSIONS[i].equals(extension)) found = true;
        } // End of for loop

        return found;
   } // End of public static boolean acceptFile(String path)
} // End of public class ImageFilter extends FileFilter
```

B.14 ImageInfo.java

```java
// ImageInfo.java
//
// ImageInfo.java is the placeholder to keep the information such as path
// name, file size, and file type of the image file.

import java.awt.*;
import java.awt.image.*;
import javax.swing.*;
import java.io.*;

public class ImageInfo{
   protected String path;              // Complete path of the image
   protected String fileType;          // File type of the image file
   protected long fileSize;            // File size in bytes of the image
   protected int width;                // The width of the image
   protected int height;               // The height of the image

   // Constructor: Initialize the path to the parametric object and
   // calculate the remaining values
   public ImageInfo(String imgPath){
      path = imgPath;

      ImageIcon icon = new ImageIcon(path);
      width = icon.getIconWidth();
      height = icon.getIconHeight();

      File file = new File(path);
      fileSize = file.length();

      fileType = "UNKNOWN File";

      String ext = file.getName().substring(file.getName().lastIndexOf("."));

      if (ext.toLowerCase().equals(".jpg") ||
          ext.toLowerCase().equals(".jpeg") ||
          ext.toLowerCase().equals(".jpe")) fileType = "JPG File";
      else if (ext.toLowerCase().equals(".png")) fileType = "PNG File";
      else if (ext.toLowerCase().equals(".gif")) fileType = "GIF File";
   } // End of public ImageInfo(String imgPath)

   // Returns the complete path of the image file associated with this object
   public String getPath(){
      return path;
   } // End of public String getPath()

   // Returns the file name of the image file
   public String getFileName(){
      File tmp = new File(path);
      return tmp.getName();
   } // End of public String getFileName()

   // Returns the formatted name of the file without the extension part
   public String getName(){
      File tmp = new File(path);
      return tmp.getName().substring(0, tmp.getName().lastIndexOf("."));
   } // End of public getName()

   // Returns the image object of the file path
   public Image getImage(){
      return new ImageIcon(path).getImage();
   } // End of public Image getImage()

   // Returns the buffered image object of the file path
   public BufferedImage getBufferedImage(){
      Image img = new ImageIcon(path).getImage();
      BufferedImage toReturn = new BufferedImage(this.getWidth(),
                                                 this.getHeight(),
                                                 BufferedImage.TYPE_INT_RGB);
      Graphics2D graphics2D = toReturn.createGraphics();
      graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                  RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      graphics2D.drawImage(img, 0, 0, this.getWidth(), this.getHeight(), null);
      graphics2D.dispose();
      img.flush();

      return toReturn;
   } // End of public BufferedImage getBufferedImage()
   // Returns the width of the image
   public int getWidth(){
      return width;
   } // End of public int getWidth()

   // Returns the height of the image
   public int getHeight(){
      return height;
   } // End ofp ublic int getHeight()

   // Returns the file size in bytes of the image file
   public long getFileSize(){
      return fileSize;
   } // End of public long getFileSize()

   // Returns the file type string of the image file
   public String getFileType(){
      return fileType;
   } // End of public String getFileType()
} // End of public class ImageInfo
```

B.15 ImageListCellRenderer.java

```
// ImageListCellRenderer.java
//
// Image List Cell Renderer is used to render the objects stored in the
// image list.

import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

public class ImageListCellRenderer extends JLabel implements ListCellRenderer{
        public ImageListCellRenderer(){
                this.setOpaque(true);
        } // End of ImageListCellRenderer()

        public Component getListCellRendererComponent(
          JList list, Object value, int index, boolean isSelected,
          boolean cellHasFocus)
        {
                //this.setIcon(new ImageIcon(LIST_ICON));
                this.setText(((ImageInfo)value).getName());
                this.setToolTipText(((ImageInfo)value).getPath());

      this.setBackground(isSelected ? Color.RED : Color.WHITE);
      this.setForeground(isSelected ? Color.WHITE : Color.BLACK);
      return this;
   } // End of public Component getListCellRendererComponent(KJList list,
     //       Object value, int index, boolean isSelected, boolean cellHasFocus)
} // End of public public ImageListCellRenderer extends ListCellRenderer
```

B.16 ImagePanel.java

```
// ImagePanel.java
//
// Image Panel is used to display a specific image by scaling proportionally
// until it fits the width and height of this image panel.

import java.awt.*;
import java.awt.image.*;
import javax.swing.*;

public class ImagePanel extends JPanel{
   BufferedImage image;
   BufferedImage toPaint;
   float percentage = 1.0f;
   boolean displayNoImageText;
```

```
public ImagePanel(boolean hasNoImageText){
   image = null;
   toPaint = null;
   displayNoImageText = hasNoImageText;
} // end of public ImagePanel()

// Scales the image proportionally and displays the image contents
// in the appropriate area
public void setImage(BufferedImage img){
   image = new BufferedImage(img.getWidth(), img.getHeight(),
                             BufferedImage.TYPE_INT_RGB);
   Graphics2D graphics2D = image.createGraphics();
   graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                               RenderingHints.VALUE_INTERPOLATION_BILINEAR);
   graphics2D.drawImage(img, 0, 0, img.getWidth(), img.getHeight(), null);

   percentage = this.calculateMaximumPercentage(img);
   int imgWidth = (int)(img.getWidth() * percentage);
   int imgHeight = (int)(img.getHeight() * percentage);

   // Resizing image to paint
   toPaint = new BufferedImage(imgWidth, imgHeight,
                               BufferedImage.TYPE_INT_RGB);
   graphics2D = toPaint.createGraphics();
   graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                               RenderingHints.VALUE_INTERPOLATION_BILINEAR);
   graphics2D.drawImage(image, 0, 0, imgWidth, imgHeight, null);
   graphics2D.dispose();

   this.repaint();
} // End of public void setImage(BufferedImage img)

public void clear(){
   if (image != null) image.flush();
   if (toPaint != null) toPaint.flush();
   image = null;
   toPaint = null;
   this.repaint();
} // End of public void clear()

// Sets the specified image by scaling to the specified percentage
// and displays in the appropriate area
public void setImage(BufferedImage img, float newPercentage){
   image = img;
   percentage = Math.min(newPercentage,
                         this.calculateMaximumPercentage(img));
   int imgWidth = (int)(img.getWidth() * percentage);
   int imgHeight = (int)(img.getHeight() * percentage);

   // Resizing image to paint
   toPaint = new BufferedImage(imgWidth, imgHeight,
                               BufferedImage.TYPE_INT_RGB);
   Graphics2D graphics2D = toPaint.createGraphics();
   graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                               RenderingHints.VALUE_INTERPOLATION_BILINEAR);
   graphics2D.drawImage(image, 0, 0, imgWidth, imgHeight, null);
   graphics2D.dispose();

   this.repaint();
} // End of public void setImage(BufferedImage img, int percentage)

// Returns the buffered image of this object that is currently displaying
public BufferedImage getImage(){
   return image;
} // End of public BufferedImage getImage()

// Calculates and returns the maximum percentage that the image can
// fit in the display area according to the width and height of this
// image panel.
public float calculateMaximumPercentage(BufferedImage img){
   int width = this.getWidth() - 10;
   int height = this.getHeight() - 25;

   float calcPercentage = 1.0f;

   while ((int)(img.getWidth() * calcPercentage) > width)
      calcPercentage -= 0.01;
   while ((int)(img.getHeight() * calcPercentage) > height)
      calcPercentage -= 0.01;

   return calcPercentage;
   } // End of public float calculateMaximumPercentage(BufferedImage img)

   // Paints the image into the appropriate area
   public void paint(Graphics g){
      super.paint(g);

      this.setCursor(new Cursor(Cursor.WAIT_CURSOR));

      int width = this.getWidth() - 10;
      int height = this.getHeight() - 25;    // -10 for border, -15 for text

      if (toPaint == null){         // no image info to display
         g.drawRect(5, 5, width, height + 15);
         g.drawRect(6, 6, width - 2, height + 15 - 2);

         if (displayNoImageText){
            Font font = new Font("Arial", Font.BOLD, 30);
            FontMetrics fontMetrics = this.getFontMetrics(font);
            String imgText = "NO IMAGE TO DISPLAY";
            int fontWidth = fontMetrics.stringWidth(imgText);

            if (height - 30 > fontMetrics.getHeight()){
               g.setFont(font);
               g.drawString(imgText, (width / 2) - (fontWidth / 2) + 5,
                        (height / 2) - (fontMetrics.getHeight() / 2) + 10 + 15);
            } // End of if statement
         } // End of if statement
      } // End of if statement
      else{
         int startX = 10, startY = 10;

         // Calculating starting point x and y to draw image
         startX += (width / 2) - (toPaint.getWidth() / 2) - 5;
         startY += (height / 2) - (toPaint.getHeight() / 2) - 5;
         g.drawImage(Toolkit.getDefaultToolkit().createImage(
                     toPaint.getSource()),
                     startX, startY,
                     toPaint.getWidth(), toPaint.getHeight(), null);

         // Draw borders of the image
         g.drawRect(startX - 4, startY - 4, toPaint.getWidth() + 8,
                    toPaint.getHeight() + 8);
         g.drawRect(startX - 3, startY - 3, toPaint.getWidth() + 6,
                    toPaint.getHeight() + 6);

         Font font = new Font("Arial", Font.BOLD, 14);
         FontMetrics fontMetrics = this.getFontMetrics(font);
         String imgText = "Original Size: " + image.getWidth() + " x " +
                          image.getHeight() + " - Display Size: " +
                          toPaint.getWidth() + " x " + toPaint.getHeight() +
                          " (" + (int)(percentage * 100) + "%)";

         int fontWidth = fontMetrics.stringWidth(imgText);

         // Displaying image size info
         g.setFont(font);
         g.drawString(imgText, (width / 2) - (fontWidth / 2) + 5,
                      startY + toPaint.getHeight() + 20);
      } // End of else statement

      this.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
   } // End of public void paint(Graphics graphics)
} // End of public class ImagePane extends JPanel
```

B.17 InsertionDialog.java

```
// InsertionDialog.java
//
// Insertion dialog is used to display the image during the insertion process

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import java.util.*;
import java.math.*;
import java.io.*;
import javax.imageio.*;

public class InsertionDialog extends JDialog implements ActionListener{
   public static final int BLOCK_WIDTH = 8;
   public static final int BLOCK_HEIGHT = 8;
   private static final Color CAPTION_BACKGROUND = new Color(184, 207, 229);
   private static final double DISPLAY_PERCENTAGE = 0.95;
   private static final String DEFAULT_SAVE_DIR = "saved/";

   protected JLabel blockLabel, hashLabel, elapsedLabel;
   public JTextField blockTextField, hashTextField, elapsedTextField;
   public JProgressBar progress;
   protected ImagePanel imagePanel;
   protected JButton saveButton, startButton, closeButton;
   protected JFrame parent;
   public BufferedImage image, watermark, displayImage;
   protected String algorithm;
   protected int totalBlocksCount, blockWidthCount, blockHeightCount;
   protected BigInteger e, n;
   protected Thread insertionThread = null;    // Used for insertion

   public InsertionDialog(JFrame pParent, BufferedImage pImage,
                          BufferedImage pWatermark, BigInteger pE,
                          BigInteger pN, String alg){
      super(pParent, true);
      parent = pParent;
      image = pImage;
      watermark = pWatermark;
      e = pE;
      n = pN;
      algorithm = alg;

      displayImage = new BufferedImage(watermark.getWidth(),
                                       watermark.getHeight(),
                                       BufferedImage.TYPE_INT_RGB);
      Graphics2D graphics2D = displayImage.createGraphics();
      graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                  RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      graphics2D.drawImage(watermark, 0, 0, watermark.getWidth(),
                           watermark.getHeight(), null);
      graphics2D.dispose();

      Rectangle desktopSize = GraphicsEnvironment.
                       getLocalGraphicsEnvironment().getMaximumWindowBounds();

      this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
      this.setSize((int)(desktopSize.getWidth() * DISPLAY_PERCENTAGE),
                   (int)(desktopSize.getHeight() * DISPLAY_PERCENTAGE));
      this.setTitle("Watermark Insertion - " + algorithm + " (" +
                    image.getWidth() + " x " + image.getHeight() + ")");

      imagePanel = new ImagePanel(true);

      blockWidthCount = image.getWidth() / BLOCK_WIDTH;
      blockHeightCount = image.getHeight() / BLOCK_HEIGHT;

      if (image.getWidth() % BLOCK_WIDTH != 0) blockWidthCount++;
      if (image.getHeight() % BLOCK_HEIGHT != 0) blockHeightCount++;
      totalBlocksCount = blockWidthCount * blockHeightCount;

      progress = new JProgressBar(0, totalBlocksCount);
      progress.setValue(0);
      progress.setStringPainted(true);

      blockLabel = new JLabel("Block Status: ", JLabel.RIGHT);
      blockLabel.setOpaque(true);
      blockLabel.setBackground(CAPTION_BACKGROUND);
      blockTextField = new JTextField(20);
      blockTextField.setEditable(false);

      hashLabel = new JLabel("MD5 Hash Key: ", JLabel.RIGHT);
      hashLabel.setOpaque(true);
      hashLabel.setBackground(CAPTION_BACKGROUND);
      hashTextField = new JTextField(20);
      hashTextField.setEditable(false);

      elapsedLabel = new JLabel("Elapsed Time: ", JLabel.RIGHT);
      elapsedLabel.setOpaque(true);
      elapsedLabel.setBackground(CAPTION_BACKGROUND);
      elapsedTextField = new JTextField("00:00:00.000");
      elapsedTextField.setHorizontalAlignment(JTextField.CENTER);
      elapsedTextField.setEditable(false);

      saveButton = new JButton("Save Image");
      saveButton.setEnabled(false);

      startButton = new JButton("Start Process");
      startButton.setMnemonic(KeyEvent.VK_S);

      closeButton = new JButton("Close Window");
      closeButton.setMnemonic(KeyEvent.VK_C);

      this.setLayout(null);
      this.add(imagePanel);
      this.add(progress);
      this.add(blockLabel);
      this.add(blockTextField);
      this.add(hashLabel);
      this.add(hashTextField);
      this.add(elapsedLabel);
      this.add(elapsedTextField);
      this.add(saveButton);
      this.add(startButton);
      this.add(closeButton);

      int width = this.getWidth() - 20;
      int height = this.getHeight() - 40;

      imagePanel.setBounds(5, 5, width, height - 90);
      progress.setBounds(10, height - 80, width - 10, 20);
      blockLabel.setBounds(10, height - 55, 100, 20);
      blockTextField.setBounds(115, height - 55, width - 445, 20);
      hashLabel.setBounds(10, height - 30, 100, 20);
      hashTextField.setBounds(115, height - 30, width - 445, 20);
      elapsedLabel.setBounds(width - 320, height - 55, 95, 20);
      elapsedTextField.setBounds(width - 220, height - 55, 95, 20);
      saveButton.setBounds(width - 120 - 120 - 5, height - 25, 120, 25);
      startButton.setBounds(width - 120, height - 55, 120, 25);
      closeButton.setBounds(width - 120, height - 25, 120, 25);

      saveButton.addActionListener(this);
      startButton.addActionListener(this);
      closeButton.addActionListener(this);

      this.updateImage();
      this.center();
      this.setResizable(false);
      this.setVisible(true);
   } // End of public InsertionDialog(JFrame pParent, BufferedImage pImage,
     //                              BufferedImage pWatermark, String alg)

   // Centers the dialog repective to the desktop area
   private void center(){
      Rectangle desktopSize = GraphicsEnvironment.
                         getLocalGraphicsEnvironment().getMaximumWindowBounds();
      this.setLocation((int)((desktopSize.getWidth() - this.getWidth()) / 2),
                       (int)((desktopSize.getHeight() - this.getHeight()) / 2));
   } // End of private void center()

   public void dispose(){
      super.dispose();
      if (insertionThread != null) insertionThread.interrupt();
      if (image != null) image.flush();
      if (watermark != null) watermark.flush();
      if (displayImage != null) displayImage.flush();
   } // End of public void dispose()

   // Returns the number of total blocks in the image to be watermarked
   public int getTotalBlocksCount(){
      return totalBlocksCount;
   } // End of public int getTotalBlocksCount()

   // Returns the number of blocks calculated from the width of the image
   public int getBlockWidthCount(){
      return blockWidthCount;
   } // End of public int getBlockWidthCount()

   // Returns the number of blocks calculated from the height of the image
   public int getBlockHeightCount(){
      return blockHeightCount;
   } // End of public int getBlockHeightCount
```

```
// Updates the image to be displayed
public void updateImage(){
   imagePanel.setImage(displayImage);
} // end of public void updateImage()

// Activates the save button
public void activateSaveButton(){
   saveButton.setEnabled(true);
   startButton.setEnabled(true);
} // end of public void activateSaveButton()

// Action Listener event handler
public void actionPerformed(ActionEvent evt){
   if (evt.getSource() == saveButton){
      JFileChooser fileChooser = null;

      try{
         fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
      } // End of try statement
      catch(Exception e){
          fileChooser = new JFileChooser(System.getProperty("user.dir"));
      } // End of catch statement

      fileChooser.setAcceptAllFileFilterUsed(false);
      fileChooser.setFileFilter(new PNGFilter());

      int result = fileChooser.showSaveDialog(null);

      if (result == JFileChooser.APPROVE_OPTION){
         String filePath = fileChooser.getSelectedFile().getAbsolutePath();

         if (!filePath.toLowerCase().endsWith(".png")) filePath += ".png";

         result = JOptionPane.YES_OPTION;

         if (new File(filePath).exists()){
            Toolkit.getDefaultToolkit().beep();
            result = JOptionPane.showConfirmDialog(null, "Do you want to " +
                         "replace the existing file?", "File exists",
                         JOptionPane.YES_NO_OPTION);
         } // End of if statement

         if (result == JOptionPane.YES_OPTION){
            fileChooser.setCursor(new Cursor(Cursor.WAIT_CURSOR));

            try{
               ImageIO.write(displayImage, "PNG", new File(filePath));
            } // End of try statement
            catch(Exception ex){
               Toolkit.getDefaultToolkit().beep();
               JOptionPane.showMessageDialog(null, "An error occurred " +
                         "while saving the requested image.", "I/O Error",
                         JOptionPane.ERROR_MESSAGE);
            } // end of catch statement
            finally{
               fileChooser.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
            } // End of finally
         } // End of if statement
      } // End of if statement
   } // End of if statement
   else if (evt.getSource() == startButton){
      displayImage = new BufferedImage(watermark.getWidth(),
                                       watermark.getHeight(),
                                       BufferedImage.TYPE_INT_RGB);
      Graphics2D graphics2D = displayImage.createGraphics();
      graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                              RenderingHints.VALUE_INTERPOLATION_BILINEAR);
      graphics2D.drawImage(watermark, 0, 0, watermark.getWidth(),
                        watermark.getHeight(), null);
      graphics2D.dispose();
      progress.setValue(0);
      saveButton.setEnabled(false);
      this.updateImage();

      if (algorithm.equals("Wong's Algorithm")){
         insertionThread = new WongInsertionThread(this, e, n);
         insertionThread.setPriority(Thread.NORM_PRIORITY);
         startButton.setEnabled(false);
         insertionThread.start();
      } // End of if statement
```

```
         else if (algorithm.equals("HBC1 Algorithm")){
            insertionThread = new HBC1InsertionThread(this, e, n);
            insertionThread.setPriority(Thread.NORM_PRIORITY);
            startButton.setEnabled(false);
            insertionThread.start();
         } // End of else if statement
         else if (algorithm.equals("HBC2 Algorithm")){
            insertionThread = new HBC2InsertionThread(this, e, n);
            insertionThread.setPriority(Thread.NORM_PRIORITY);
            startButton.setEnabled(false);
            insertionThread.start();
         } // End of else if statement
         else if (algorithm.equals("Proposed Algorithm")){
            insertionThread = new ProposedInsertionThread(this, e, n);
            insertionThread.setPriority(Thread.NORM_PRIORITY);
            startButton.setEnabled(false);
            insertionThread.start();
         } // End of else if statement
         else{
            JOptionPane.showMessageDialog(parent, "Invalid Insertion Algorithm",
                  "Algorithm Error", JOptionPane.ERROR_MESSAGE);
            startButton.setEnabled(false);
         } // End of else statement
      } // End of else if statement
      else if (evt.getSource() == closeButton){
         this.setVisible(false);
         this.dispose();
      } // End of else if statement
   } // End of public void actionPerformed(ActionEvent evt)
} // End of public class InsertionDialog extends Dialog
```

B.18 InterchangeDialog.java

```
// InterchangeDialog.java

import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import javax.imageio.*;

public class InterchangeDialog extends JDialog implements ActionListener{
   private static final Color CAPTION_BACKGROUND = new Color(184, 207, 229);
   private static final double DISPLAY_PERCENTAGE = 0.95;
   private static final String DEFAULT_SAVE_DIR = "saved/";

   protected JLabel image1Label, image2Label, repeatLabel, xLabel, pixelLabel;
   protected JLabel caption1Label, caption2Label, interchangedCaption;
   protected JTextField image1TextField, image2TextField;
   protected JButton image1Button, image2Button, startButton, saveButton,
                     closeButton;
   protected JRadioButton version1RadioButton, version2RadioButton;
   protected JSpinner widthSpinner, heightSpinner;
   protected ImagePanel interchangedPanel;
   protected ThumbPanel thumb1, thumb2;
   protected BufferedImage image1 = null, image2 = null,
             interchangedImage1 = null, interchangedImage2 = null;
   protected Thread interchangeThread = null;

   public InterchangeDialog(ImageIntegrity pParent){
      super(pParent, true);

      Rectangle desktopSize = GraphicsEnvironment.
                        getLocalGraphicsEnvironment().getMaximumWindowBounds();

      this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
      this.setSize((int)(desktopSize.getWidth() * DISPLAY_PERCENTAGE),
                   (int)(desktopSize.getHeight() * DISPLAY_PERCENTAGE));
      this.setTitle("Image Blocks Interchange");

      image1Label = new JLabel("Image # 1: ", JLabel.RIGHT);
      image1Label.setOpaque(true);
      image1Label.setBackground(CAPTION_BACKGROUND);
      image1TextField = new JTextField(20);
      image1TextField.setEditable(false);
```

```
image1Button = new JButton("Browse...");
image2Label = new JLabel("Image # 2: ", JLabel.RIGHT);
image2Label.setOpaque(true);
image2Label.setBackground(CAPTION_BACKGROUND);
image2TextField = new JTextField(20);
image2TextField.setEditable(false);
image2Button = new JButton("Browse...");

repeatLabel = new JLabel("Repeat Every ");
xLabel = new JLabel("x", JLabel.CENTER);
pixelLabel = new JLabel("pixels");
widthSpinner = new JSpinner();
widthSpinner.setEnabled(false);
heightSpinner = new JSpinner();
heightSpinner.setEnabled(false);

caption1Label = new JLabel("Preview Image # 1", JLabel.CENTER);
caption1Label.setOpaque(true);
caption1Label.setBackground(CAPTION_BACKGROUND);
caption2Label = new JLabel("Preview Image # 2", JLabel.CENTER);
caption2Label.setOpaque(true);
caption2Label.setBackground(CAPTION_BACKGROUND);
interchangedCaption = new JLabel("Preview Interchanged Image",
                                 JLabel.CENTER);
interchangedCaption.setOpaque(true);
interchangedCaption.setBackground(CAPTION_BACKGROUND);
thumb1 = new ThumbPanel();
thumb2 = new ThumbPanel();
interchangedPanel = new ImagePanel(true);

version1RadioButton = new JRadioButton("Interchanged Image # 1",
                                       true);
version1RadioButton.setEnabled(false);
version2RadioButton = new JRadioButton("Interchanged Image # 2");
version2RadioButton.setEnabled(false);
ButtonGroup group = new ButtonGroup();
group.add(version1RadioButton);
group.add(version2RadioButton);

startButton = new JButton("Start Interchange Process");
startButton.setMnemonic(KeyEvent.VK_S);
startButton.setEnabled(false);
startButton.setToolTipText("Start Block Interchange Process");

saveButton = new JButton("Save Image");
saveButton.setEnabled(false);
saveButton.setToolTipText("Save the current displaying interchanged " +
                          "image");

closeButton = new JButton("Close Window");
closeButton.setMnemonic(KeyEvent.VK_C);
closeButton.setToolTipText("Close Window");

this.setLayout(null);
this.add(image1Label);
this.add(image1TextField);
this.add(image1Button);
this.add(image2Label);
this.add(image2TextField);
this.add(image2Button);
this.add(repeatLabel);
this.add(widthSpinner);
this.add(xLabel);
this.add(heightSpinner);
this.add(pixelLabel);
this.add(caption1Label);
this.add(caption2Label);
this.add(interchangedCaption);
this.add(thumb1);
this.add(thumb2);
this.add(interchangedPanel);
this.add(version1RadioButton);
this.add(version2RadioButton);
this.add(startButton);
this.add(saveButton);
this.add(closeButton);

int width = this.getWidth() - 20;
int height = this.getHeight() - 40;

image1Button.setBounds(10, 10, 90, 25);
```

```
   image1Label.setBounds(110, 12, 80, 20);
   image1TextField.setBounds(200, 12, width - 470, 20);
   image2Button.setBounds(10, 40, 90, 25);
   image2Label.setBounds(110, 42, 80, 20);
   image2TextField.setBounds(200, 42, width - 470, 20);
   repeatLabel.setBounds(width - 260, 10, 100, 20);
   widthSpinner.setBounds(width - 180, 10, 60, 20);
   xLabel.setBounds(width - 115, 10, 10, 20);
   heightSpinner.setBounds(width - 100, 10, 60, 20);
   pixelLabel.setBounds(width - 35, 10, 50, 20);
   caption1Label.setBounds(10, 70, 250, 20);
   thumb1.setBounds(10, 95, 250, 250);
   caption2Label.setBounds(10, 360, 250, 20);
   thumb2.setBounds(10, 385, 250, 250);
   startButton.setBounds(width - 260, 40, 260, 25);
   interchangedCaption.setBounds(270, 70, width - 270, 20);
   interchangedPanel.setBounds(270, 95, width - 270, height - 130);
   version1RadioButton.setBounds(width - 120 - 120 - 165 - 165 - 5,
                              height - 25, 165, 20);
   version2RadioButton.setBounds(width - 120 - 120 - 165 - 5, height - 25,
                              165, 20);
   saveButton.setBounds(width - 120 - 120 - 5, height - 25, 120, 25);
   closeButton.setBounds(width - 120, height - 25, 120, 25);

   // Registering listeners
   image1Button.addActionListener(this);
   image2Button.addActionListener(this);
   version1RadioButton.addActionListener(this);
   version2RadioButton.addActionListener(this);
   startButton.addActionListener(this);
   saveButton.addActionListener(this);
   closeButton.addActionListener(this);

   this.center();
   this.setResizable(false);
   this.setVisible(true);
} // End of public InterchangeDialog(ImageIntegrity pParent)

// Centers the dialog respective to the desktop space
private void center(){
   Rectangle desktopSize = GraphicsEnvironment.
                   getLocalGraphicsEnvironment().getMaximumWindowBounds();
   this.setLocation((int)((desktopSize.getWidth() - this.getWidth()) / 2),
                 (int)((desktopSize.getHeight() - this.getHeight()) / 2));
} // End of private void center()

public void dispose(){
   super.dispose();
   if (interchangeThread != null) interchangeThread.interrupt();
   if (image1 != null) image1.flush();
   if (image2 != null) image2.flush();
   if (interchangedImage1 != null) interchangedImage1.flush();
   if (interchangedImage2 != null) interchangedImage2.flush();
} // End of public void dispose()

public void actionPerformed(ActionEvent evt){
   if (evt.getSource() == image1Button){
      JFileChooser fileChooser;

      try{
         fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
      } // End of try statement
      catch(Exception e){
         fileChooser = new JFileChooser(System.getProperty("user.dir"));
      } // End of catch statement

      fileChooser.setDialogTitle("Open Image");
      fileChooser.setAcceptAllFileFilterUsed(false);
      //fileChooser.setFileFilter(new PNGFilter());
      fileChooser.setFileFilter(new ImageFilter());

      int result = fileChooser.showOpenDialog(this);

      if (result == JFileChooser.APPROVE_OPTION){
         interchangedImage1 = null;
         interchangedImage2 = null;
         version1RadioButton.setEnabled(false);
         version2RadioButton.setEnabled(false);
         widthSpinner.setEnabled(false);
         heightSpinner.setEnabled(false);
         startButton.setEnabled(false);
```

```
            saveButton.setEnabled(false);
            String path = fileChooser.getSelectedFile().getAbsolutePath();
            ImageInfo info = new ImageInfo(path);
            image1TextField.setText(info.getPath());
            thumb1.setImageInfo(info);

            image1 = info.getBufferedImage();
         } // End of if statement

         if (image1 != null && image2 != null){
            interchangedPanel.clear();
            SpinnerNumberModel widthModel = new SpinnerNumberModel();
            widthModel.setValue(8);
            widthModel.setMinimum(1);
            widthModel.setMaximum(image1.getWidth());
            widthModel.setStepSize(8);

            SpinnerNumberModel heightModel = new SpinnerNumberModel();
            heightModel.setValue(8);
            heightModel.setMinimum(1);
            heightModel.setMaximum(image1.getHeight());
            heightModel.setStepSize(8);

            widthSpinner.setModel(widthModel);
            heightSpinner.setModel(heightModel);
            widthSpinner.setEnabled(true);
            heightSpinner.setEnabled(true);
            startButton.setEnabled(true);
         } // End of if statement
      } // End of if stamtent
      else if (evt.getSource() == image2Button){
         JFileChooser fileChooser;

         try{
            fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
         } // End of try statement
         catch(Exception e){
            fileChooser = new JFileChooser(System.getProperty("user.dir"));
         } // End of catch statement

         fileChooser.setDialogTitle("Open Image");
         fileChooser.setAcceptAllFileFilterUsed(false);
         //fileChooser.setFileFilter(new PNGFilter());
         fileChooser.setFileFilter(new ImageFilter());

         int result = fileChooser.showOpenDialog(this);

         if (result == JFileChooser.APPROVE_OPTION){
            interchangedImage1 = null;
            interchangedImage2 = null;
            version1RadioButton.setEnabled(false);
            version2RadioButton.setEnabled(false);
            widthSpinner.setEnabled(false);
            heightSpinner.setEnabled(false);
            startButton.setEnabled(false);
            saveButton.setEnabled(false);
            String path = fileChooser.getSelectedFile().getAbsolutePath();
            ImageInfo info = new ImageInfo(path);
            image2TextField.setText(info.getPath());
            thumb2.setImageInfo(info);

            image2 = info.getBufferedImage();
         } // End of if statement

         if (image1 != null && image2 != null){
            interchangedPanel.clear();
            SpinnerNumberModel widthModel = new SpinnerNumberModel();
            widthModel.setValue(8);
            widthModel.setMinimum(1);
            widthModel.setMaximum(image1.getWidth());
            widthModel.setStepSize(8);

            SpinnerNumberModel heightModel = new SpinnerNumberModel();
            heightModel.setValue(8);
            heightModel.setMinimum(1);
            heightModel.setMaximum(image1.getHeight());
            heightModel.setStepSize(8);

            widthSpinner.setModel(widthModel);
            heightSpinner.setModel(heightModel);
            widthSpinner.setEnabled(true);
      heightSpinner.setEnabled(true);
      startButton.setEnabled(true);
   } // End of if statement
} // End of else if statement
else if (evt.getSource() == startButton){
   if (image1.getWidth() == image2.getWidth() &&
       image2.getHeight() == image2.getHeight()){
      interchangedImage1 = new BufferedImage(image1.getWidth(),
                                       image1.getHeight(),
                                       BufferedImage.TYPE_INT_RGB);
      interchangedImage2 = new BufferedImage(image2.getWidth(),
                                       image2.getHeight(),
                                       BufferedImage.TYPE_INT_RGB);

      startButton.setEnabled(false);
      version1RadioButton.setSelected(true);
      version1RadioButton.setEnabled(false);
      version2RadioButton.setEnabled(false);
      image1Button.setEnabled(false);
      image2Button.setEnabled(false);
      interchangedPanel.setImage(interchangedImage1);

      interchangeThread = new Thread(){
         public void run(){
            int blockWidth, blockHeight;

            blockWidth = Integer.parseInt(widthSpinner.getValue().
                                            toString());
            blockHeight = Integer.parseInt(heightSpinner.getValue().
                                            toString());

            int RGB1 = 0, RGB2 = 0;
            int turn = 0;
            for (int y = 0; y < image1.getHeight(); y = y + blockHeight){
               for (int x = 0; x < image1.getWidth() &&
                    !this.isInterrupted(); x = x + blockWidth){

                  int widthLimit = Math.min(blockWidth,
                                      image1.getWidth() - x);
                  int heightLimit = Math.min(blockHeight,
                                       image1.getHeight() - y);

                  for (int j = 0; j < heightLimit; j++){
                     for (int i = 0; i < widthLimit; i++){
                        RGB1 = image1.getRGB(x + i, y + j);
                        RGB2 = image2.getRGB(x + i, y + j);

                        if (turn % 2 == 0){
                           interchangedImage1.setRGB(x + i, y + j, RGB2);
                           interchangedImage2.setRGB(x + i, y + j, RGB1);
                        } // End of if statemnet
                        else{
                           interchangedImage1.setRGB(x + i, y + j, RGB1);
                           interchangedImage2.setRGB(x + i, y + j, RGB2);
                        } // End of else statement
                     } // End of for loop
                  } // end of for loop

                  turn++;
               } // End of for loop
               interchangedPanel.setImage(interchangedImage1);
            } // End of for loop

            if (!this.isInterrupted()){
               version1RadioButton.setEnabled(true);
               version2RadioButton.setEnabled(true);
               startButton.setEnabled(true);
               saveButton.setEnabled(true);
               version1RadioButton.setEnabled(true);
               version2RadioButton.setEnabled(true);

               Toolkit.getDefaultToolkit().beep();
               JOptionPane.showMessageDialog(null, "Image block " +
                     "interchange was completed successfully", "Success",
                     JOptionPane.INFORMATION_MESSAGE);
            } // End of if statement

            image1 = null;
            image2 = null;
            image1Button.setEnabled(true);
            image2Button.setEnabled(true);
```

```
               } // End of public void run()
            };
            interchangeThread.start();
         } // End of if statement
         else{
            Toolkit.getDefaultToolkit().beep();
            JOptionPane.showMessageDialog(this, "Cannot interchange blocks " +
              "of two images of different dimensions", "Dimensions",
              JOptionPane.ERROR_MESSAGE);
         } // End of else statmenet
      } // end of if statement
      else if (evt.getSource() == version1RadioButton){
         interchangedPanel.setImage(interchangedImage1);
      } // End of else if statement
      else if (evt.getSource() == version2RadioButton){
         interchangedPanel.setImage(interchangedImage2);
      } // end of else if statement
      else if (evt.getSource() == saveButton){
         JFileChooser fileChooser = null;

         try{
            fileChooser = new JFileChooser(DEFAULT_SAVE_DIR);
         } // End of try statement
         catch(Exception e){
             fileChooser = new JFileChooser(System.getProperty("user.dir"));
         } // End of catch statement

         fileChooser.setAcceptAllFileFilterUsed(false);
         fileChooser.setFileFilter(new PNGFilter());

         int result = fileChooser.showSaveDialog(null);

         if (result == JFileChooser.APPROVE_OPTION){
            String filePath = fileChooser.getSelectedFile().getAbsolutePath();

            if (!filePath.toLowerCase().endsWith(".png")) filePath += ".png";

            result = JOptionPane.YES_OPTION;

            if (new File(filePath).exists()){
               Toolkit.getDefaultToolkit().beep();
               result = JOptionPane.showConfirmDialog(null, "Do you want to " +
                             "replace the existing file?", "File exists",
                             JOptionPane.YES_NO_OPTION);
            } // End of if statement

            if (result == JOptionPane.YES_OPTION){
               fileChooser.setCursor(new Cursor(Cursor.WAIT_CURSOR));

               try{
                  ImageIO.write(interchangedPanel.getImage(), "PNG",
                                new File(filePath));
               } // End of try statement
               catch(Exception ex){
                  Toolkit.getDefaultToolkit().beep();
                  JOptionPane.showMessageDialog(null, "An error occurred " +
                             "while saving the requested image.", "I/O Error",
                             JOptionPane.ERROR_MESSAGE);
               } // end of catch statement
               finally{
                  fileChooser.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
               } // End of finally
            } // End of if statement
         } // End of if statement
      } // End of else if statement
      else if (evt.getSource() == closeButton){
         this.setVisible(false);
         this.dispose();
      } // end of else if statement
   } // End of public void actionPerformed(ActionEvent evt)
} // End of public class InterchangeDialog extends JDialog
```

B.19 MD5Digest.java

```java
// MD5Digest.java
//
// MD5Digest is a static class that provides the hex and binary MD5 digest
// of a specific string

import java.security.*;
import javax.swing.*;

public class MD5Digest{
   // Return the hexadecimal representation of the MD5 digest of the
   // given string
   public static String getHexDigest(String msg){
      String hexString = new String();
      byte[] bytes = msg.getBytes();

      try{
         MessageDigest MD5 = MessageDigest.getInstance("MD5");
         MD5.reset();
         MD5.update(bytes);

         byte messageDigest[] = MD5.digest();


         for (int i = 0; i < messageDigest.length; i++){
            String hex = Integer.toHexString(0xFF & messageDigest[i]);

            if (hex.length() == 1) hexString += "0";
            hexString += hex;
         } // End of for loop
      }
      catch(Exception e){
         JOptionPane.showMessageDialog(null, "There was an error computing " +
             "the message digest.", "Digest Error", JOptionPane.ERROR_MESSAGE);
         System.exit(-1);
      } // End of catch statement

      return hexString;
   } // End of public static String getHexDigest(String msg)

   // Returns the binary representation of the MD5 digest of the given string
   public static String getBinDigest(String msg){
      String hexDigest = MD5Digest.getHexDigest(msg).substring(0, 32);
      String binary = new String();

      for (int i = 0; i < hexDigest.length(); i++){
         int dec = Integer.parseInt(hexDigest.substring(i, i + 1), 16);
         String tmpBin = Integer.toBinaryString(dec);

         for (int j = tmpBin.length(); j < 4; j++) tmpBin = "0" + tmpBin;
         binary += tmpBin;
      } // End of for loop

      return binary;
   } // End of public static String getBinDigest(String msg)
} // end of public class MD5Digest
```

B.20 PNGFilter.java

```java
// ImageFilter.java
//
// Image filter to be used to filter PNG files only

import java.io.File;
import javax.swing.filechooser.FileFilter;

public class PNGFilter extends FileFilter{
   // Return true if the file is a valid file determined by its file
   // extension, false otherwise
   public boolean accept(File file){
      if (file.isDirectory()) return true;

      String extension = file.getName().substring(
                 file.getName().lastIndexOf(".")).toLowerCase();

      if (extension.equals(".png")) return true;
      else return false;
    } // End of public boolean accept(File f)

   // Returns the description of this image filter
   public String getDescription(){
      return "PNG Files (*.png)";
   } // End of public String getDescription()
} // End of public class PNGFilter extends FileFilter
```

B.21 PreviewDialog.java

```
// PreviewDialog.java
//
// PreviewDialog is used for displaying the preview of the watermark image
// to be used in the insertion step. Will allow users to preview a sample
// tiled or scaled watermark image.

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.text.*;

public class PreviewDialog extends JDialog implements ActionListener{
   private static final Color CAPTION_BACKGROUND = new Color(184, 207, 229);
   private static final double DISPLAY_PERCENTAGE = 0.95;

   protected JLabel originalLabel, enlargedLabel, enlargedTextLabel;
   protected JLabel filePathLabel, dimensionsLabel, sizeLabel;
   protected JTextField filePathTextField, dimensionsTextField, sizeTextField;
   protected JRadioButton tiledRadioButton, scaledRadioButton;
   protected JButton closeButton;
   protected ImageInfo info;
   protected JPanel topPanel, bottomPanel;

   public PreviewDialog(JFrame parent, ImageInfo imageInfo){
      super(parent, "Watermark Preview", true);
      info = imageInfo;
      Rectangle desktopSize = GraphicsEnvironment.getLocalGraphicsEnvironment().
                              getMaximumWindowBounds();

      this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
      this.setSize((int)(desktopSize.getWidth() * DISPLAY_PERCENTAGE),
                   (int)(desktopSize.getHeight() * DISPLAY_PERCENTAGE));

      originalLabel = new JLabel(new ImageIcon(imageInfo.getImage()));
      originalLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 2));
      filePathLabel = new JLabel("File Path: ", JLabel.RIGHT);
      filePathLabel.setBackground(CAPTION_BACKGROUND);
      filePathLabel.setOpaque(true);
      dimensionsLabel = new JLabel("Dimensions: ", JLabel.RIGHT);
      dimensionsLabel.setBackground(CAPTION_BACKGROUND);
      dimensionsLabel.setOpaque(true);
      sizeLabel = new JLabel("File Size: ", JLabel.RIGHT);
      sizeLabel.setBackground(CAPTION_BACKGROUND);
      sizeLabel.setOpaque(true);
      filePathTextField = new JTextField(20);
      filePathTextField.setText(info.getPath());
      filePathTextField.setEditable(false);
      filePathTextField.setToolTipText(filePathTextField.getText());
      dimensionsTextField = new JTextField(20);
      dimensionsTextField.setText(info.getWidth() + " x " + info.getHeight());
      dimensionsTextField.setEditable(false);
      dimensionsTextField.setToolTipText(dimensionsTextField.getText());
      sizeTextField = new JTextField(20);
      sizeTextField.setText(new DecimalFormat("0.00 KB").format(
                            info.getFileSize() / 1024.0));
      sizeTextField.setEditable(false);
      sizeTextField.setToolTipText(sizeTextField.getText());

      enlargedLabel = new JLabel();
```

```
enlargedLabel.setHorizontalAlignment(JLabel.CENTER);
enlargedLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 2));
enlargedTextLabel = new JLabel();
enlargedTextLabel.setHorizontalAlignment(JLabel.CENTER);

tiledRadioButton = new JRadioButton("Tiled Watermark", true);
scaledRadioButton = new JRadioButton("Scaled Watermark");
ButtonGroup group = new ButtonGroup();
group.add(tiledRadioButton);
group.add(scaledRadioButton);
closeButton = new JButton("Close Window");
closeButton.setMnemonic(KeyEvent.VK_C);

bottomPanel = new JPanel();
bottomPanel.setLayout(new FlowLayout());
bottomPanel.add(tiledRadioButton);
bottomPanel.add(scaledRadioButton);
bottomPanel.add(closeButton);

this.setLayout(null); // Null layout
this.add(originalLabel);
this.add(filePathLabel);
this.add(dimensionsLabel);
this.add(sizeLabel);
this.add(filePathTextField);
this.add(dimensionsTextField);
this.add(sizeTextField);
this.add(enlargedLabel);
this.add(enlargedTextLabel);
this.add(bottomPanel);

int orgWidth = (int)originalLabel.getPreferredSize().getWidth();
int orgHeight = (int)originalLabel.getPreferredSize().getHeight();

originalLabel.setBounds(5, 5, orgWidth, orgHeight);
filePathLabel.setBounds(5 + orgWidth + 5, 5, 100, 20);
filePathTextField.setBounds(5 + orgWidth + 100 + 5 + 5, 5,
                          this.getWidth() - orgWidth - 100 - 40, 20);
dimensionsLabel.setBounds(5 + orgWidth + 5, 5 + 20 + 3, 100, 20);
dimensionsTextField.setBounds(5 + orgWidth + 100 + 5 + 5, 5 + 20 + 3,
                            this.getWidth() - orgWidth - 100 - 40, 20);
sizeLabel.setBounds(5 + orgWidth + 5, 5 + 20 + 3 + 20 + 3, 100, 20);
sizeTextField.setBounds(5 + orgWidth + 100 + 5 + 5, 5 + 20 + 3 + 20 + 3,
                      this.getWidth() - orgWidth - 100 - 40, 20);
int yStart = Math.max(5 + orgHeight + 5, 80);
enlargedLabel.setBounds(5, yStart, this.getWidth() - 20,
                      this.getHeight() - yStart -
                      (int)bottomPanel.getPreferredSize().getHeight() -
                      55);
enlargedTextLabel.setBounds(5, yStart + enlargedLabel.getHeight(),
                          enlargedLabel.getWidth(), 20);
bottomPanel.setBounds(5, yStart + enlargedLabel.getHeight() + 20,
                    this.getWidth() - 20,
                    (int)bottomPanel.getPreferredSize().getHeight());

// Tiling watermark image
BufferedImage tiled = new BufferedImage(enlargedLabel.getWidth(),
                                        enlargedLabel.getHeight(),
                                        BufferedImage.TYPE_INT_RGB);
Graphics2D graphics2D = tiled.createGraphics();
graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                            RenderingHints.VALUE_INTERPOLATION_BILINEAR);

for (int y = 0; y < tiled.getHeight(); y = y + info.getHeight()){
   for (int x = 0; x < tiled.getWidth(); x = x + info.getWidth()){
      graphics2D.drawImage(info.getImage(), x, y, info.getWidth(),
                           info.getHeight(), null);
   } // End of for loop
} // End of for loop
graphics2D.dispose();
enlargedLabel.setIcon(new ImageIcon(tiled));
enlargedTextLabel.setText("Sample Tiled Watermark (" +
                          tiled.getWidth() + " x " + tiled.getHeight() +
                          ")");
// Registering listeners
tiledRadioButton.addActionListener(this);
scaledRadioButton.addActionListener(this);
closeButton.addActionListener(this);

this.center();
this.setResizable(false);
```

```
      this.setVisible(true);
   } // End of public PreviewDialog(ImageInfo imageInfo)

   // Centers the dialog respective to the desktop size
   private void center(){
      Rectangle desktopSize = GraphicsEnvironment.getLocalGraphicsEnvironment().
                               getMaximumWindowBounds();
      this.setLocation((int)((desktopSize.getWidth() - this.getWidth()) / 2),
                       (int)((desktopSize.getHeight() - this.getHeight()) / 2));
   } // End of private void center()

   // Action Listeners event handler
   public void actionPerformed(ActionEvent evt){
      if (evt.getSource() == closeButton){
         this.setVisible(false);
         this.dispose();
      } // End of if statement
      else if (evt.getSource() == tiledRadioButton){
         int width = enlargedLabel.getWidth();
         int height = enlargedLabel.getHeight();

         BufferedImage tiled = new BufferedImage(width,
                                                 height,
                                                 BufferedImage.TYPE_INT_RGB);
         Graphics2D graphics2D = tiled.createGraphics();
         graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                     RenderingHints.VALUE_INTERPOLATION_BILINEAR);

         for (int y = 0; y < height; y = y + info.getHeight()){
            for (int x = 0; x < width; x = x + info.getWidth()){
               graphics2D.drawImage(info.getImage(), x, y, info.getWidth(),
                                    info.getHeight(), null);
            } // End of for loop
         } // End of for loop
         graphics2D.dispose();

         enlargedLabel.setIcon(new ImageIcon(tiled));
         enlargedTextLabel.setText("Sample Tiled Watermark (" +
                                   width + " x " +
                                   height + ")");
      } // End of else statement
      else if (evt.getSource() == scaledRadioButton){
         int width = enlargedLabel.getWidth();
         int height = enlargedLabel.getHeight();

         BufferedImage scaled = new BufferedImage(width, height,
                                                  BufferedImage.TYPE_INT_RGB);
         Graphics2D graphics2D = scaled.createGraphics();
         graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                                     RenderingHints.VALUE_INTERPOLATION_BILINEAR);

         graphics2D.drawImage(info.getImage(), 0, 0, width, height, null);
         graphics2D.dispose();

         enlargedLabel.setIcon(new ImageIcon(scaled));
         enlargedTextLabel.setText("Sample Scaled Watermark (" +
                                   width + " x " +
                                   height + ")");
      } // End of else if statement
   } // End of public void actionPerformed(ActionEvent evt)
} // End of public class PreviewDialog extends JDialog
```

B.22 PrivateFilter.java

```
// PrivateFilter.java
//
// Image filter to the used to filter only .private files

import java.io.File;
import javax.swing.filechooser.FileFilter;

public class PrivateFilter extends FileFilter{
   // Return true if the file is a valid file determined by its file
   // extension, false otherwise
   public boolean accept(File file){
      if (file.isDirectory()) return true;
      if (file.getName().toLowerCase().endsWith(".private")) return true;
      else return false;
    } // End of public boolean accept(File f)

   // Returns the description of this file filter
   public String getDescription(){
      return "Private Key File (*.private)";
   } // End of public String getDescription()
} // End of public class PrivateFilter extends FileFilter
```

B.23 ProposedAlgorithm.java

```
// ProposedAlgorithm.java
//
// Base class DisplayArea, provides the visual interface of the display area.
// In addition, insertion and extraction will be done using the proposed
// algorithm

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.math.*;

public class ProposedAlgorithm extends DisplayArea{
   public ProposedAlgorithm(ImageIntegrity pParent){
      super(pParent);    // call base class constructor
      ALGORITHM_TEXT = "PROPOSED ALGORITHM";
   } // End of public ProposedAlgorithm(ImageIntegrity pParent)

   // Insers the watermark
   public void insertWatermark(){
      BigInteger e = parent.getE();
      BigInteger n = parent.getN();

      if (e == null || n == null){
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(parent, "Insertion process cannot be " +
                "performed because private key is missing", "Missing Key",
                JOptionPane.ERROR_MESSAGE);
      } // End of if statement
      else{
         new InsertionDialog(parent, image, watermark, e, n, "Proposed " +
                             "Algorithm");
      } // End of else statmenet
   } // End of public void insertWatermark()

   // Extracts the watermark from the watermarked image
   public void extractWatermark(){
      new ExtractionDialog(parent, "Proposed Algorithm");
   } // end of public void extractWatermark()
} // End of public class ProposedAlgorithm extends DisplayArea
```

B.24 ProposedExtractionThread.java

```
// ProposedExtractionThread.java
//
// Thread to extract the watermark from the watermarked image using the
// proposed extraction algorithm

import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.math.BigInteger;
import java.util.Timer;
import java.io.*;

public class ProposedExtractionThread extends Thread{
   protected ExtractionDialog dialog;
   protected RSACryptoSystem RSA;
   protected Timer elapsedTimer = new Timer();
```

```
public ProposedExtractionThread(ExtractionDialog pDialog, BigInteger pD,
                            BigInteger pN){
   dialog = pDialog;
   RSA = new RSACryptoSystem();
   RSA.setPublicKey(pD, pN);
} // End of public ProposedExtractionThread(ExtractionDialog pDialog,
  //                                        BigInteger pD, BigInteger pN)

public void run(){
   int imgWidth = dialog.image.getWidth();
   int imgHeight = dialog.image.getHeight();
   int blockWidth = 0, blockHeight = 0, totalBlocks = 0;
   int imageRGB, red, green, blue, blockIndex = 0;
   String zeroedString, zeroedStringPrev, hashString, encodedString,
          digest, binaryDigest, prevDigest, decodedBinary, xorString,
          nonDetString, tmpDetString;
   BigInteger msg = null, dec = null, xor = null;

   elapsedTimer.schedule(new ElapsedTimerTask(dialog.elapsedTextField),
                         0, 1);

   // Obtaining the previous block for the first block
   int rowStart_prev = imgWidth - (imgWidth % dialog.BLOCK_WIDTH);
   int colStart_prev = imgHeight - (imgHeight % dialog.BLOCK_HEIGHT);

   if (rowStart_prev == imgWidth) rowStart_prev -= dialog.BLOCK_WIDTH;
   if (colStart_prev == imgHeight) colStart_prev -= dialog.BLOCK_HEIGHT;

   prevDigest = new String();
   zeroedStringPrev = new String();
   nonDetString = new String();
   for (int y_prev = colStart_prev; y_prev < imgHeight; y_prev++){
      for (int x_prev = rowStart_prev; x_prev < imgWidth; x_prev++){
         imageRGB = dialog.image.getRGB(x_prev, y_prev);

         red = (imageRGB >> 16) & 0xFF;
         green = (imageRGB >> 8) & 0xFF;
         blue = (imageRGB >> 0) & 0xFF;

         if (blue % 2 == 1) blue--;    // Zero blue LSB

         zeroedStringPrev += String.valueOf(red) +
                             String.valueOf(green) +
                             String.valueOf(blue);
      } // End of for loop
   } // End of for loop

   for (int y = 0; y < imgHeight; y = y + dialog.BLOCK_HEIGHT){
      for (int x = 0; x < imgWidth && !this.isInterrupted();
           x = x + dialog.BLOCK_WIDTH){
         blockIndex++;
         dialog.blockTextField.setText(blockIndex + " of " +
                              dialog.getTotalBlocksCount());

         blockWidth = Math.min(dialog.BLOCK_WIDTH, imgWidth - x);
         blockHeight = Math.min(dialog.BLOCK_HEIGHT, imgHeight - y);
         totalBlocks = blockWidth * blockHeight;

         zeroedString = new String();
         encodedString = new String();
         tmpDetString = new String();

         for (int j = 0; j < blockHeight; j++){
            for (int i = 0; i < blockWidth; i++){
               imageRGB = dialog.image.getRGB(x + i, y + j);

               red = (imageRGB >> 16) & 0xFF;
               green = (imageRGB >> 8) & 0xFF;
               blue = (imageRGB >> 0) & 0xFF;

               tmpDetString += String.valueOf(red) +
                               String.valueOf(green) +
                               String.valueOf(blue);

               if (blue % 2 == 1){
                  blue--;
                  encodedString += "1";
               } // End of if statement
               else{
                  encodedString += "0";
               } // End of else statement
               zeroedString += String.valueOf(red) +
                               String.valueOf(green) +
                               String.valueOf(blue);
            } // End of for loop
         } // End of for loop

         hashString = imgWidth + imgHeight + zeroedString +
                      zeroedStringPrev + blockIndex + nonDetString;
         zeroedStringPrev = zeroedString;    // Saving current zeroed string
                                             // for next iteration
         dialog.hashTextField.setText(MD5Digest.getHexDigest(hashString));

         digest = MD5Digest.getBinDigest(hashString);
         binaryDigest = digest.substring(0, totalBlocks);
         //nonDetString = binaryDigest + tmpDetString;
         nonDetString = encodedString;

         msg = new BigInteger(encodedString, 2);

         if (blockIndex != 1){
            msg = msg.xor(new BigInteger(prevDigest, 2));
         } // End of if statement

         prevDigest = digest.substring(totalBlocks, 2 * totalBlocks);

         if (totalBlocks < (dialog.BLOCK_WIDTH * dialog.BLOCK_HEIGHT) ||
             msg.compareTo(RSA.getN()) > 0){

           decodedBinary = msg.toString(2);

           for (int k = decodedBinary.length(); k < totalBlocks; k++)
              decodedBinary = "0" + decodedBinary;
           //decodedBinary = encodedString;
         } // End of if statement
         else{ // Decode message
            dec = RSA.decrypt(msg);
            decodedBinary = dec.toString(2);

            // Padding with leading zeroes
            for (int k = decodedBinary.length(); k < totalBlocks; k++)
               decodedBinary = "0" + decodedBinary;
         } // End of if statement

         xor = new BigInteger(binaryDigest, 2).xor(
                                           new BigInteger(decodedBinary, 2));
         xorString = xor.toString(2);

         // Pad with leading zeroes
         for (int k = xorString.length(); k < totalBlocks; k++)
            xorString = "0" + xorString;

         int index = 0;
         for (int jj = 0; jj < blockHeight; jj++){
            for (int ii = 0; ii < blockWidth; ii++){
               char c = xorString.charAt(index);

               if (c == '1')
                  dialog.image.setRGB(x + ii, y + jj, Color.WHITE.getRGB());
               else
                  dialog.image.setRGB(x + ii, y + jj, Color.BLACK.getRGB());
               index++;
            } // End of for loop
         } // End of for loop

         dialog.progress.setValue(dialog.progress.getValue() + 1);

         if (blockIndex % dialog.getBlockWidthCount() == 0){
            dialog.updateImage();
         } // End of if statement
      } // End of for loop
   } // End of for loop

   elapsedTimer.cancel();

   if (!this.isInterrupted()){
      dialog.activateSaveButton();
      Toolkit.getDefaultToolkit().beep();
      JOptionPane.showMessageDialog(dialog, "Proposed watermark " +
                                 "extraction process was completed " +
                                 "successfully.", "Success",
                                 JOptionPane.INFORMATION_MESSAGE);
         } // End of if statement
      } // end of public void run()
   } // End of public class ProposedExtractionThread extends Thread
```

B.25 ProposedInsertionThread.java

```
// ProposedInsertionThread.java
//
// Thread to insert the watermark into the cover image using the proposed
// insertion algorithm

import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.math.BigInteger;
import java.util.Timer;
import java.io.*;

public class ProposedInsertionThread extends Thread{
   protected InsertionDialog dialog;
   protected RSACryptoSystem RSA;
   protected Timer elapsedTimer = new Timer();

   public ProposedInsertionThread(InsertionDialog pDialog, BigInteger pE,
                              BigInteger pN){
      dialog = pDialog;
      RSA = new RSACryptoSystem();
      RSA.setPrivateKey(pE, pN);
   } // End of public ProposedInsertionThread(InsertionDialog pDialog,
     //                                  BigInteger pE, BigInteger pN)

   public void run(){
      int imgWidth = dialog.image.getWidth();
      int imgHeight = dialog.image.getHeight();
      int blockWidth = 0, blockHeight = 0, totalBlocks = 0;
      int red, green, blue, blockIndex = 0;
      int imageRGB, waterRGB;
      String zeroedString, zeroedStringPrev, waterString, digest, binaryDigest,
             prevDigest, encodedBinary, hashString, nonDetString;
      BigInteger msg = null, enc = null, prev = null;
      int count = 0;

      elapsedTimer.schedule(new ElapsedTimerTask(dialog.elapsedTextField),
                            0, 1);

      // Obtaining the previous block for the first block
      int rowStart_prev = imgWidth - (imgWidth % dialog.BLOCK_WIDTH);
      int colStart_prev = imgHeight - (imgHeight % dialog.BLOCK_HEIGHT);

      if (rowStart_prev == imgWidth) rowStart_prev -= dialog.BLOCK_WIDTH;
      if (colStart_prev == imgHeight) colStart_prev -= dialog.BLOCK_HEIGHT;

      prevDigest = new String();
      zeroedStringPrev = new String();
      nonDetString = new String();
      for (int y_prev = colStart_prev; y_prev < imgHeight; y_prev++){
         for (int x_prev = rowStart_prev; x_prev < imgWidth; x_prev++){
            imageRGB = dialog.image.getRGB(x_prev, y_prev);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            if (blue % 2 == 1) blue--;    // Zero blue LSB

            zeroedStringPrev += String.valueOf(red) +
                                String.valueOf(green) +
                                String.valueOf(blue);
         } // End of for loop
      } // End of for loop

      for (int y = 0; y < imgHeight; y = y + dialog.BLOCK_HEIGHT){
         for (int x = 0; x < imgWidth && !this.isInterrupted();
              x = x + dialog.BLOCK_WIDTH){
            blockIndex++;
```

```
dialog.blockTextField.setText(blockIndex + " of " +
                              dialog.getTotalBlocksCount());

blockWidth = Math.min(dialog.BLOCK_WIDTH, imgWidth - x);
blockHeight = Math.min(dialog.BLOCK_HEIGHT, imgHeight - y);
totalBlocks = blockWidth * blockHeight;

zeroedString = new String();
waterString = new String();

for (int j = 0; j < blockHeight; j++){
   for (int i = 0; i < blockWidth; i++){
      imageRGB = dialog.image.getRGB(x + i, y + j);
      waterRGB = dialog.watermark.getRGB(x + i, y + j);

      red = (imageRGB >> 16) & 0xFF;
      green = (imageRGB >> 8) & 0xFF;
      blue = (imageRGB >> 0) & 0xFF;

      if (blue % 2 == 1) blue--;    // Zero blue LSB

      zeroedString += String.valueOf(red) +
                      String.valueOf(green) +
                      String.valueOf(blue);

      if (waterRGB == Color.BLACK.getRGB())
         waterString += "0";
      else if (waterRGB == Color.WHITE.getRGB())
         waterString += "1";
      else
         JOptionPane.showMessageDialog(null, "ERROR WATER COLOR");
   } // End of for loop
} // End of for loop

hashString = imgWidth + imgHeight + zeroedString +
             zeroedStringPrev + blockIndex + nonDetString;
zeroedStringPrev = zeroedString;    // Saving current zeroed string
                                    // for next iteration
dialog.hashTextField.setText(MD5Digest.getHexDigest(hashString));

digest = MD5Digest.getBinDigest(hashString);
binaryDigest = digest.substring(0, totalBlocks);

//System.out.println(digest + "(" + digest.length() + ")");

msg = (new BigInteger(binaryDigest, 2)).xor(
                                    new BigInteger(waterString, 2));

if (totalBlocks < (dialog.BLOCK_WIDTH * dialog.BLOCK_HEIGHT) ||
    msg.compareTo(RSA.getN()) > 0){

   if (blockIndex != 1){
      msg = msg.xor(new BigInteger(prevDigest, 2));
   } // End of if statement

   prevDigest = digest.substring(totalBlocks, 2 * totalBlocks);

   encodedBinary = msg.toString(2);

   // Pad with leading zeroes
   for (int k = encodedBinary.length(); k < totalBlocks; k++)
      encodedBinary = "0" + encodedBinary;
} // End of if statement
else{ // Encode message
   enc = RSA.encrypt(msg);

   if (blockIndex != 1){
      enc = enc.xor(new BigInteger(prevDigest, 2));
   } // End of if statement

   prevDigest = digest.substring(totalBlocks, 2 * totalBlocks);

   encodedBinary = enc.toString(2);

   // Pad with leading zeroes
   for (int k = encodedBinary.length(); k < totalBlocks; k++)
      encodedBinary = "0" + encodedBinary;
} // End of if statement

nonDetString = encodedBinary;
```

```
            int index = 0;
            for (int jj = 0; jj < blockHeight; jj++){
               for (int ii = 0; ii < blockWidth; ii++){
                  imageRGB = dialog.image.getRGB(x + ii, y + jj);

                  red = (imageRGB >> 16) & 0xFF;
                  green = (imageRGB >> 8) & 0xFF;
                  blue = (imageRGB >> 0) & 0xFF;

                  char c = encodedBinary.charAt(index);
                  if (blue % 2 == 0 && c == '1') blue++;
                  else if (blue % 2 == 1 && c == '0') blue--;

                  //nonDetString += String.valueOf(red) +
                  //                String.valueOf(green) +
                  //                String.valueOf(blue);
                  dialog.displayImage.setRGB(x + ii, y + jj,
                                    new Color(red, green, blue).getRGB());
                  index++;
               } // End of for loop
            } // End of for loop

            dialog.progress.setValue(dialog.progress.getValue() + 1);

            if (blockIndex % dialog.getBlockWidthCount() == 0){
               dialog.updateImage();
            } // End of if statement
         } // End of for loop
      } // End of for loop

      elapsedTimer.cancel();

      if (!this.isInterrupted()){
         dialog.activateSaveButton();
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(dialog,
                                 "Proposed watermark insertion " +
                                 "process was completed successfully.",
                                 "Success", JOptionPane.INFORMATION_MESSAGE);
      } // End of if statement
   } // End of public void run()
} // End of public class ProposedInsertionThread extends Thread
```

B.26 PublicFilter.java

```
// PublicFilter.java
//
// Image filter to filter only .public files

import java.io.File;
import javax.swing.filechooser.FileFilter;

public class PublicFilter extends FileFilter{
   // Return true if the file is a valid file determined by its file
   // extension, false otherwise
   public boolean accept(File file){
      if (file.isDirectory()) return true;

      if (file.getName().toLowerCase().endsWith(".public")) return true;
      else return false;
    } // End of public boolean accept(File f)

   // Returns the description of this image filter
   public String getDescription(){
      return "Public Key File (*.public)";
   } // End of public String getDescription()
} // End of public class PublicFilter extends FileFilter
```

B.27 RSACryptoSystem.java

```java
// RSACryptoSystem.java
//
// Provides the implementation of an RSA cryptosytem

import java.math.BigInteger;
import java.security.SecureRandom;

public class RSACryptoSystem{
   protected int bitLength = 0;
   protected BigInteger d, e, n;

   // Default constructor
   public RSACryptoSystem(){
      d = BigInteger.ZERO;
      e = BigInteger.ZERO;
      n = BigInteger.ZERO;
   } // End of public RSACryptoSystem()

   // Generate the keys of the specified key length
   public RSACryptoSystem(int length){
      bitLength = length;
      BigInteger p, q, two = new BigInteger("2");
      SecureRandom rand = new SecureRandom();

      do{
         p = new BigInteger(bitLength / 2, 100, rand);
         q = new BigInteger(bitLength / 2, 100, rand);

         n = p.multiply(q);
      } while (n.bitLength() != bitLength);

      BigInteger phi = (p.subtract(BigInteger.ONE)).
                        multiply(q.subtract(BigInteger.ONE));

      e = new BigInteger("3");
      while(phi.gcd(e).intValue() > 1) e = e.add(new BigInteger("2"));

      d = e.modInverse(phi);
   } // End of public RSACryptoSystem

   // Returns the private key e
   public BigInteger getE(){
      return e;
   } // End of public BigInteger getE()

   // Returns the public key d
   public BigInteger getD(){
      return d;
   } // End of public BigInteger getD()

   // Returns the modulus key n
   public BigInteger getN(){
      return n;
   } // End of public BigInteger getN()

   // Sets the private key e and n to be used for encryption
   public void setPrivateKey(BigInteger pE, BigInteger pN){
      e = pE;
      n = pN;
   } // End of public void setPrivateKey(BigInteger pE, BigInteger pN)

   // Sets the private key d and n to be used for decryption
   public void setPublicKey(BigInteger pD, BigInteger pN){
      d = pD;
      n = pN;
   } // End of public void setPublicKey(BigInteger pD, BigInteger pN)

   // Encrypts and returns the result by encrypting the message with e and n
   public BigInteger encrypt(BigInteger message){
      return message.modPow(e, n);
   } // End of public BigInteger encrypt(BigInteger message)

   // Decrypts and returns the result by decrypting the message with d and n
   public BigInteger decrypt(BigInteger message){
      return message.modPow(d, n);
   } // End of public BigInteger decrypt(BigInteger message)
} // End of public class RSACryptoSystem
```

B.28 ThumbPanel.java

```
// Thumbpanel.java
//
// Thumbpanel provides the visual representation of the thumbnail displaying
// the image file in a thumbnail fashion and also displaying its file
// attributes such as dimensions and size.

import java.awt.*;
import javax.swing.*;
import java.text.*;
import java.io.*;

public class ThumbPanel extends JPanel{
   private static final Color CAPTION_BACKGROUND = new Color(184, 207, 229);
   private static final Color INFO_BACKGROUND = new Color(255, 175, 175);
   protected ImageInfo info;        // the current image info
   protected JPanel infoPanel;
   protected JLabel orgSizeCaption, orgSize;
   protected JLabel fileSizeCaption, fileSize;
   protected JLabel fileTypeCaption, fileType;

   // Default constructor
   public ThumbPanel(){
      info = null;
      Initialize();     // Initializing components
   } // End of public ThumbPanel()

   // Constructor: Initializes the panel with the image denoted by path
   public ThumbPanel(ImageInfo pInfo){
      info = pInfo;
      Initialize();     // Initializing components
   } // End of public ThumbPanel(String path)

   // Initialize components with their default values
   private void Initialize(){
      orgSizeCaption = new JLabel("Original Size: ", JLabel.RIGHT);
      orgSizeCaption.setBackground(CAPTION_BACKGROUND);
      orgSizeCaption.setOpaque(true);
      orgSize = new JLabel(" N/A");
      orgSize.setBackground(INFO_BACKGROUND);
      orgSize.setOpaque(true);
      fileSizeCaption = new JLabel("File Size: ", JLabel.RIGHT);
      fileSizeCaption.setBackground(CAPTION_BACKGROUND);
      fileSizeCaption.setOpaque(true);
      fileSize = new JLabel(" N/A");
      fileSize.setBackground(INFO_BACKGROUND);
      fileSize.setOpaque(true);
      fileTypeCaption = new JLabel("File Type: ", JLabel.RIGHT);
      fileTypeCaption.setBackground(CAPTION_BACKGROUND);
      fileTypeCaption.setOpaque(true);
      fileType = new JLabel(" N/A");
      fileType.setBackground(INFO_BACKGROUND);
      fileType.setOpaque(true);

      infoPanel = new JPanel();
      infoPanel.setLayout(new GridLayout(3, 2, 2, 2));
      infoPanel.add(orgSizeCaption);
      infoPanel.add(orgSize);
      infoPanel.add(fileSizeCaption);
      infoPanel.add(fileSize);
      infoPanel.add(fileTypeCaption);
      infoPanel.add(fileType);

      this.setLayout(new BorderLayout(5, 5));
      this.add(infoPanel, BorderLayout.SOUTH);
   } // End of private void Initialize()

   // Sets the current image info to be displayed and updates the
   // values on the screen
   public void setImageInfo(ImageInfo pInfo){
      info = pInfo;
      orgSize.setText(" " + info.getWidth() + " x " + info.getHeight());

      DecimalFormat sizeFormat = new DecimalFormat("0.00 KB");
      fileSize.setText(" " + sizeFormat.format(info.getFileSize() / 1024.0));

      fileType.setText(" " + info.getFileType());
      this.repaint();
   } // End of public void setImageInfo(ImageInfo pInfo)

   // Clears the contents and thumbnail display area
   public void clear(){
      info = null;
      orgSize.setText(" N/A");
      fileSize.setText(" N/A");
      fileType.setText(" N/A");
      this.repaint();
   } // End of public void clear()

   // Paint the components
   public void paint(Graphics g){
      super.paint(g);

      int width = this.getWidth() - 10;
      int height = this.getHeight() - infoPanel.getHeight() - 30;

      if (info == null){         // no image info to display
         g.drawRect(5, 5, width, height + 20);
         g.drawRect(6, 6, width - 2, height + 20 - 2);
         Font font = new Font("Arial", Font.BOLD, 20);
         FontMetrics fontMetrics = this.getFontMetrics(font);
         String imgText = "NO IMAGE";
         int fontWidth = fontMetrics.stringWidth(imgText);

         if (height - 30 > fontMetrics.getHeight()){
            g.setFont(font);
            g.drawString(imgText, (width / 2) - (fontWidth / 2) + 5,
                        (height / 2) - (fontMetrics.getHeight() / 2) + 10 + 20);
         } // End of if statement
      } // End of if statement
      else{
         ImageIcon icon = new ImageIcon(info.getImage());

         int startX = 10, startY = 10;
         float percentage = 1.0f;

         while ((int)(icon.getIconWidth() * percentage) > width)
            percentage -= 0.01;
         while ((int)(icon.getIconHeight() * percentage) > height)
            percentage -= 0.01;

         // width and height of the scaled image by percentage
         int imgWidth = (int)(icon.getIconWidth() * percentage);
         int imgHeight = (int)(icon.getIconHeight() * percentage);

         // Calculating starting point x and y to draw image
         startX += (width / 2) - (imgWidth / 2) - 5;
         startY += (height / 2) - (imgHeight / 2) - 5;

         // Draw image
         g.drawImage(icon.getImage(), startX, startY, imgWidth,
                     imgHeight, null);

         // Draw borders of the image
         g.drawRect(startX - 4, startY - 4, imgWidth + 8, imgHeight + 8);
         g.drawRect(startX - 3, startY - 3, imgWidth + 6, imgHeight + 6);

         Font font = new Font("Arial", Font.BOLD, 12);
         FontMetrics fontMetrics = this.getFontMetrics(font);
         String imgText = imgWidth + " x " + imgHeight + " (" +
                          (int)(percentage * 100) + "%)";
         int fontWidth = fontMetrics.stringWidth(imgText);

         // Displaying image size info
         g.setFont(font);
         g.drawString(imgText, (width / 2) - (fontWidth / 2) + 5,
                      startY + imgHeight + 20);
      } // End of else statement
   } // End of public void paint(Graphics g)
} // End of public class ThumbPanel extends JPanel
```

B.29 WongAlgorithm.java

```
// WongAlgorithm.java
//
// Base class DisplayArea, provides the visual interface of the display area.
// In addition, insertion and extraction will be done using the original
// watermarking algorithm proposed by Wong.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.math.BigInteger;

public class WongAlgorithm extends DisplayArea{
   public WongAlgorithm(ImageIntegrity pParent){
      super(pParent);    // call base class constructor
      ALGORITHM_TEXT = "WONG'S ALGORITHM";
   } // End of public WongAlgorithm(IamgeIntegrity pParent)

   // Insers the watermark
   public void insertWatermark(){
      BigInteger e = parent.getE();
      BigInteger n = parent.getN();

      if (e == null || n == null){
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(parent, "Insertion process cannot be " +
                "performed because private key is missing", "Missing Key",
                JOptionPane.ERROR_MESSAGE);
      } // End of if statement
      else{
         new InsertionDialog(parent, image, watermark, e, n,
                             "Wong's Algorithm");
      } // End of else statmenet
   } // End of public void insertWatermark()

   // Extracts the watermark from the watermarked image
   public void extractWatermark(){
      new ExtractionDialog(parent, "Wong's Algorithm");
   } // end of public void extractWatermark()
} // End of public class WongAlgorithm extends DisplayArea
```

B.30 WongExtractionThread.java

```
// WongExtractionThread.java
//
// Thread to extract the watermark from the watermarked image using Wong's
// extraction algorithm

import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.math.BigInteger;
import java.util.Timer;
import java.io.*;

public class WongExtractionThread extends Thread{
   protected ExtractionDialog dialog;
   protected RSACryptoSystem RSA;
   protected Timer elapsedTimer = new Timer();

   public WongExtractionThread(ExtractionDialog pDialog, BigInteger pD,
                               BigInteger pN){
      dialog = pDialog;
      RSA = new RSACryptoSystem();
      RSA.setPublicKey(pD, pN);
   } // End of public WongExtractionThread(ExtractionDialog pDialog,
    //                                   BigInteger pD, BigInteger pN)

   public void run(){
      int imgWidth = dialog.image.getWidth();
      int imgHeight = dialog.image.getHeight();
      int blockWidth = 0, blockHeight = 0, totalBlocks = 0;
      int imageRGB, red, green, blue, blockIndex = 0;
```

```
String zeroedString, hashString, encodedString, binaryDigest,
       decodedBinary, xorString;
BigInteger msg = null, dec = null, xor = null;

elapsedTimer.schedule(new ElapsedTimerTask(dialog.elapsedTextField),
                      0, 1);

for (int y = 0; y < imgHeight; y = y + dialog.BLOCK_HEIGHT){
   for (int x = 0; x < imgWidth && !this.isInterrupted();
        x = x + dialog.BLOCK_WIDTH){
      blockIndex++;
      dialog.blockTextField.setText(blockIndex + " of " +
                                    dialog.getTotalBlocksCount());

      blockWidth = Math.min(dialog.BLOCK_WIDTH, imgWidth - x);
      blockHeight = Math.min(dialog.BLOCK_HEIGHT, imgHeight - y);
      totalBlocks = blockWidth * blockHeight;

      zeroedString = new String();
      encodedString = new String();

      for (int j = 0; j < blockHeight; j++){
         for (int i = 0; i < blockWidth; i++){
            imageRGB = dialog.image.getRGB(x + i, y + j);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            if (blue % 2 == 1){
               blue--;
               encodedString += "1";
            } // End of if statement
            else{
               encodedString += "0";
            } // End of else statement

            zeroedString += String.valueOf(red) +
                            String.valueOf(green) +
                            String.valueOf(blue);
         } // End of for loop
      } // End of for loop

      hashString = imgWidth + imgHeight + zeroedString;
      dialog.hashTextField.setText(MD5Digest.getHexDigest(hashString));

      binaryDigest = MD5Digest.getBinDigest(hashString);
      binaryDigest = binaryDigest.substring(0, totalBlocks);
      msg = new BigInteger(encodedString, 2);

      if (totalBlocks < (dialog.BLOCK_WIDTH * dialog.BLOCK_HEIGHT) ||
          msg.compareTo(RSA.getN()) > 0){
         decodedBinary = encodedString;
      } // End of if statement
      else{ // Decode message
         dec = RSA.decrypt(msg);
         decodedBinary = dec.toString(2);

         // Padding with leading zeroes
         for (int k = decodedBinary.length(); k < totalBlocks; k++)
            decodedBinary = "0" + decodedBinary;
      } // End of if statement

      xor = new BigInteger(binaryDigest, 2).xor(
                                        new BigInteger(decodedBinary, 2));
      xorString = xor.toString(2);

      // Pad with leading zeroes
      for (int k = xorString.length(); k < totalBlocks; k++)
         xorString = "0" + xorString;

      int index = 0;
      for (int jj = 0; jj < blockHeight; jj++){
         for (int ii = 0; ii < blockWidth; ii++){
            char c = xorString.charAt(index);

            if (c == '1')
               dialog.image.setRGB(x + ii, y + jj, Color.WHITE.getRGB());
            else
               dialog.image.setRGB(x + ii, y + jj, Color.BLACK.getRGB());
            index++;
```

```
               } // End of for loop
            } // End of for loop

            dialog.progress.setValue(dialog.progress.getValue() + 1);

            if (blockIndex % dialog.getBlockWidthCount() == 0){
               dialog.updateImage();
            } // End of if statement
         } // End of for loop
      } // End of for loop

      elapsedTimer.cancel();

      if (!this.isInterrupted()){
         dialog.activateSaveButton();
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(dialog, "Wong's watermark extraction " +
                                       "process was completed successfully.",
                                       "Success", JOptionPane.INFORMATION_MESSAGE);
      } // End of if statement
   } // end of public void run()
} // End of public class WongExtractionThread extends Thread
```

B.31 WongInsertionThread.java

```
// WongInsertionThread.java
//
// Thread to insert the watermark into the cover image using Wong's
// insertion algorithm

import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.math.BigInteger;
import java.util.Timer;
import java.io.*;

public class WongInsertionThread extends Thread{
   protected InsertionDialog dialog;
   protected RSACryptoSystem RSA;
   protected Timer elapsedTimer = new Timer();

   public WongInsertionThread(InsertionDialog pDialog, BigInteger pE,
                              BigInteger pN){
      dialog = pDialog;
      RSA = new RSACryptoSystem();
      RSA.setPrivateKey(pE, pN);
   } // End of public WongInsertionThread(InsertionDialog pDialog,
     //                                  BigInteger pE, BigInteger pN)

   public void run(){
      int imgWidth = dialog.image.getWidth();
      int imgHeight = dialog.image.getHeight();
      int blockWidth = 0, blockHeight = 0, totalBlocks = 0;
      int red, green, blue, blockIndex = 0;
      int imageRGB, waterRGB;
      String zeroedString, waterString, binaryDigest, encodedBinary,
             hashString;
      BigInteger msg = null, enc = null;
      int count = 0;

      elapsedTimer.schedule(new ElapsedTimerTask(dialog.elapsedTextField),
                            0, 1);

      for (int y = 0; y < imgHeight; y = y + dialog.BLOCK_HEIGHT){
         for (int x = 0; x < imgWidth && !this.isInterrupted();
           x = x + dialog.BLOCK_WIDTH){
            blockIndex++;
            dialog.blockTextField.setText(blockIndex + " of " +
                                          dialog.getTotalBlocksCount());

            blockWidth = Math.min(dialog.BLOCK_WIDTH, imgWidth - x);
            blockHeight = Math.min(dialog.BLOCK_HEIGHT, imgHeight - y);
            totalBlocks = blockWidth * blockHeight;

            zeroedString = new String();
      waterString = new String();

      for (int j = 0; j < blockHeight; j++){
         for (int i = 0; i < blockWidth; i++){
            imageRGB = dialog.image.getRGB(x + i, y + j);
            waterRGB = dialog.watermark.getRGB(x + i, y + j);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            if (blue % 2 == 1) blue--;    // Zero blue LSB

            zeroedString += String.valueOf(red) +
                            String.valueOf(green) +
                            String.valueOf(blue);

            if (waterRGB == Color.BLACK.getRGB())
               waterString += "0";
            else if (waterRGB == Color.WHITE.getRGB())
               waterString += "1";
            else
               JOptionPane.showMessageDialog(null, "ERROR WATER COLOR");
         } // End of for loop
      } // End of for loop

      hashString = imgWidth + imgHeight + zeroedString;
      dialog.hashTextField.setText(MD5Digest.getHexDigest(hashString));

      binaryDigest = MD5Digest.getBinDigest(hashString);
      binaryDigest = binaryDigest.substring(0, totalBlocks);

      msg = (new BigInteger(binaryDigest, 2)).xor(
                                    new BigInteger(waterString, 2));

      if (totalBlocks < (dialog.BLOCK_WIDTH * dialog.BLOCK_HEIGHT) ||
          msg.compareTo(RSA.getN()) > 0){

         encodedBinary = msg.toString(2);

         // Pad with leading zeroes
         for (int k = encodedBinary.length(); k < totalBlocks; k++)
            encodedBinary = "0" + encodedBinary;
      } // End of if statement
      else{ // Encode message
         enc = RSA.encrypt(msg);
         encodedBinary = enc.toString(2);

         // Pad with leading zeroes
         for (int k = encodedBinary.length(); k < totalBlocks; k++)
            encodedBinary = "0" + encodedBinary;
      } // End of if statement

      int index = 0;
      for (int jj = 0; jj < blockHeight; jj++){
         for (int ii = 0; ii < blockWidth; ii++){
            imageRGB = dialog.image.getRGB(x + ii, y + jj);

            red = (imageRGB >> 16) & 0xFF;
            green = (imageRGB >> 8) & 0xFF;
            blue = (imageRGB >> 0) & 0xFF;

            char c = encodedBinary.charAt(index);
            if (blue % 2 == 0 && c == '1') blue++;
            else if (blue % 2 == 1 && c == '0') blue--;
            dialog.displayImage.setRGB(x + ii, y + jj,
                                new Color(red, green, blue).getRGB());
            index++;
         } // End of for loop
      } // End of for loop

      dialog.progress.setValue(dialog.progress.getValue() + 1);

      if (blockIndex % dialog.getBlockWidthCount() == 0){
         dialog.updateImage();
      } // End of if statement
   } // End of for loop
} // End of for loop

elapsedTimer.cancel();
```

```
      if (!this.isInterrupted()){
         dialog.activateSaveButton();
         Toolkit.getDefaultToolkit().beep();
         JOptionPane.showMessageDialog(dialog, "Wong's watermark insertion " +
                                    "process was completed successfully.",
                                    "Success", JOptionPane.INFORMATION_MESSAGE);
      } // End of if statement
   } // End of public void run()
} // End of public class WongInsertionThread extends Thread
```

VITA

Cheng-Ting Yeh

Candidate for the Degree of

Master of Science

Thesis: IMAGE INTEGRITY VERIFICATION USING CONTENT-BASED WATERMARKING

Major Field: Computer Science

Biographical:

Education: Graduated from Instituto Educativo Los Pinos High School, La Paz, Bolivia in November 1997; received Bachelor of Science degree in Computer Science from Iowa State University, Ames, Iowa in May 2002. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in May 2005.

Experience: Raised in La Paz, Bolivia; employed by Iowa State University, Department of Computer Science as an undergraduate teaching assistant from August 2001 to May 2002; employed by Iowa State University, Department of Agronomy as a Computer/Robotics support from May 2002 to December 2002.

Name: Cheng-Ting Yeh **Date of Degree:** May, 2005

Institution: Oklahoma State University **Location:** Stillwater, Oklahoma

Title of Study: IMAGE INTEGRITY VERIFICATION USING CONTENT-BASED WATERMARKING

Pages in Study: 156 **Candidate for the Degree of Master of Science**

Major Field: Computer Science

Digital watermarking is a technique to insert an information-carrying digital signature into a digital media so that the signature can be extracted for variety of purposes including ownership authentication and content verification. In recent years, the popularity of the Internet and increase of sophisticated image processing tools commercially available today have created opportunities for malicious parties to intercept images, modify its content, and then transmit the forgeries to their respective receivers, where alterations resemble closely to the work of professionals and are undetectable by the human eye. For this reason, in order to reduce the constant proliferation and threats for digital media in the World Wide Web, digital watermarking is one of the crucial techniques to protect the contents of products in digital form and enable copyright or ownership information to be embedded into the multimedia data.

The purpose of this study was to examine the weaknesses against common watermarking attacks of blockwise independent and content-based watermarking algorithms for image integrity verification and implement a new and more secure invisible fragile watermarking technique for color or grayscale images that increases the message digest size from the proposed 64 to 128 bits using the same small-size blocks and maintaining accurate localization of image changes. Our watermarking technique is capable to detect any changes made to the image since the time it was stamped, any changes to the pixel values and also to the dimensions of the image will be automatically detected and localized. Our scheme consists of a watermark insertion process that uses a private key to embed a watermark image into a cover image, and a watermark extraction process that uses a public key to extract the watermark from the watermarked image. The embedded watermark can only be extracted by someone who has possession of a proper verification key. This implies that someone who does not have a valid key will not be able to forge a watermark that will pass undetected. Finally, our technique provides means of ensuring data integrity; adds more security to the contents of digital media and allows recipients of an image to verify its authenticity with ease as well as display the ownership information embedded within an image.

ADVISER'S APPROVAL: Dr. H. K. Dai