# CRYPTOGRAPHIC METHODS
# IN MULTIMEDIA IDENTIFICATION AND AUTHENTICATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Michael Todd Malkin
September 2006

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Dan Boneh) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(John Mitchell)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Ton Kalker)

Approved for the University Committee on Graduate Studies.

# Abstract

The work in this dissertation addresses several multimedia identification and authentication problems with concepts inspired by the field of cryptography.

Digital watermarking systems are used to authenticate ownership of and embed data in multimedia signals (audio, images, and video). Such systems generally assume that the watermark embedding system and the watermark detection system share the same secret watermarking key and are both operating in a trusted environment. In real-world applications, however, this is often not the case. Watermark embedders are generally run in a trusted environment, but watermark embedders, such as DVD players, are usually run in a hostile environment. It is necessary, therefore, to ensure that the secret watermarking key does not become compromised in the hostile environment of the watermark detector. Our Secure Watermarking system solves this problem with the help of a secure module, for example a smartcard.

The identification of images is usually considered to be an electrical engineering problem. Generally, images are assumed to be corrupted by natural processes such as noise and must be identified or authenticated in spite of this. Some more sophisticated methods consider that an attacker may be behind these processes, usually assuming that the same attacks are involved but are calibrated more intelligently. We inject cryptographic concepts into this electrical engineering problem with the randlet transform. By randomly choosing basis functions, an attacker cannot take advantage of knowledge of the basis functions in mounting attacks, as it could, for example, with the wavelet transform. The randlet transform proves to be very successful at identifying images which have been attacked, and is especially successful against rotation and cropping attacks, two attacks which have generally been very

difficult to overcome.

Image watermarking is also a problem that is considered to belong to the field of electrical engineering. We apply the randlet transform to this problem, using a variation of the QIM watermarking system with non-orthogonal basis functions (randlets) that is robust to attacks. Because the randlet transform is used, it is more difficult for an adversary to attack the randlet watermarking system.

Finally, we consider plagiarism detection. If a large number of documents are being compared, it is infeasible to run a complete search of all pairs of documents. The problem of finding duplicate documents in a large database of documents is well-studied, but very little attention has been paid to intelligent plagiarists. We show that previous systems can easily be circumvented by an adversary who knows that the systems are being used, and present a system that is robust to such an attacker. Again, our solution relies on cryptographic concepts to achieve its success.

# Acknowledgments

The road to this thesis was long and winding, filled with excitement, suspense, and adventure, but it has finally come to an end. I was aided in this journey by many people, all of whom have my eternal gratitude. I would like to thank my advisor, Dan Boneh, for his support, guidance, and patience. I am deeply grateful for the opportunity he gave me to earn a Ph.D. I can never thank Ramarathnam Venkatesan (Venkie) enough for his advice, support, collaboration, and practice playing foosball. Venkie has shown me enormous hospitality and friendship throughout the years I have known him. Ton Kalker was also instrumental in this thesis, constantly challenging me to improve my research. Our many long discussions and email exchanges were extremely helpful. John Mitchell's comments at my defense helped me to see the work in this thesis in a larger context. Finally, John Gill not only taught me information theory and error-correcting codes and served as my defense chair, he also helped me to surmount some last-minute potholes that appeared on the road to graduation.

Nothing that I have done in the past decade would have been possible if not for the constant support and patience of my wife, Catherine Nghiem. She is always my light at the end of the tunnel. This thesis is also the product of the wonderful start given to me by my parents, Barbara and Hal Malkin. From a very young age, they instilled in me a love of math and science that has stayed with me to this day. Thanks also to my in-laws, Anh and Nhin Nghiem, for raising such a wonderful daughter and for accepting me as one of their own. And for my little brother, Jonathan Malkin, I hope I've been a good role-model.

I would also like to acknowledge the help and friendship of Emin Martinian, Alex Aravanis, and Jarrod Chapman. Since our time together in Berkeley, our discussions

about life, the universe and everything have kept alive my interest in science and engineering. Thanks also to my friends Raj Batra and Sherman Lo for their invaluable assistance in preparing for my oral defense.

This thesis is dedicated to the memory of my grandmothers, Lila Berkson and Belle Malkin, and my friend, Matt Rynd.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Cryptography is a relatively young field, and many of the concepts of cryptography have not yet found widespread use in other fields. This dissertation presents several applications of the ideas of cryptography to multimedia identification and authentication.

The first idea from cryptography is that of an adversarial model. Often in fields such as watermarking and image identification, attacks on a signal are assumed to be made by nature. For example noise could be added to an image by a communications channel. An important concept from cryptography is the assumption that attacks are made by an intelligent adversary. Even when security against such adversaries can not be guaranteed, it is important to keep this framework in mind.

The second idea we take from from cryptography is to use randomization to make life more difficult for an adversary. Many signal processing algorithms involve randomness that is inherent in the signals, not in the algorithms themselves. For example, the entropy of an output can be measured when taken over a distribution of all possible inputs. This framework assumes a given distribution on inputs which may not be valid in the case of an intelligent adversary. By explicitly randomizing the algorithms, it may be possible to block all attacks by an adversary, or at least make them less effective.

In Chapters 2 and 3 we present a secure watermarking scheme. Watermarking is used to imperceptibly embed data into a signal (audio, image, or video) so that it

can be extracted even after the signal has been perturbed by an attacker. Instead of considering how a watermark is embedded and detected, we consider who is doing the embedding and detecting. The standard assumption is that both the watermark embedder and the watermark detector are trustworthy. However, in many actual applications the watermark detector is not trustworthy. For example, a DVD player may be manipulated to discover its watermarking secret.

In Chapter 2 we present a semi-public key implementation of quantization index modulation (QIM) watermarking called Secure QIM (SQIM). Given a signal, an honest-but curious (i.e. passive) watermark detector can learn if a signal was watermarked with SQIM watermark without learning anything anything else from the detection process. The watermark detector first transforms the signal with a secret transform, unknown to the detector, and then quantizes the transform coefficients with secret quantizers, also unknown to the detector. This is done with the use of homomorphic cryptosystems, where calculations are performed in an encrypted domain. A low-power, trusted, secure module is used at the end of the process and reveals only if the signal was watermarked or not. Even after repeated watermark detections, no more information is revealed than the watermarked status of the signals.

The SQIM system is limited in two ways. First, the quantizers used in QIM can only have a step size of two; and second, it is only secure against a passive adversary. In Chapter 3 we show how to use quantizer step sizes of any even value, and we address the security issue, presenting a version of the SQIM system that is provably secure against active adversaries. In Chapter 3 we also present a version of of Kalker's [22] Secure Spread Spectrum system which has been modified to make it secure against active adversaries.

In Chapter 4 we introduce a new transform called the *randlet transform* and explore applications to universal perceptual image hashing and image identification. Our transform yields signal representations robust to several common attacks. Our signal representation is hard to guess without the secret key used in its derivation and is motivated by applications such as image identification and watermarking where attack resistance is important.

We consider the performance of the randlet transform and the wavelet transform against attacks based on rotation, cropping, scaling, additive noise, and JPEG compression. The randlet transform achieves superior performance to the wavelet transform in the task of image identification. For example, a randlet transform with a false positive rate of 2% can detect a 10-degree rotation with a false negative rate of 3.2%, while the best wavelet transform with the same false positive rate has a false negative rate of 38.2%.

Next, in Chapter 5 we consider the use of the randlet transform in image watermarking. The randomized nature of the transform lends itself well to watermarking applications by making it difficult for an attacker to discover the watermark. The randlet transform is also robust to many perceptually insignificant image modifications, including malicious attacks. Watermark embedding is done by quantizing the values of randlet transform coefficients. Since the basis functions are nonorthogonal, an optimization routine is used to minimize the distortion induced by the quantization. The randomization inherent in the randlet transform means that image modifications appear as noise in the transform coefficients, lending to good robustness and security characteristics, especially when used with soft decoding.

Finally, in Chapter 6, we discuss text sifting. This is a method for quickly and securely identifying documents for database searching, copy detection, duplicate email detection and plagiarism detection. A small amount of text is extracted from a document using hash functions and is used as the document's fingerprint. We build upon previous work by Broder et al. [6,7] and Heintze [20], specifically addressing a certain set of attacks that we discovered to be very powerful against previous systems. For example, substituting every fourth word in a document with a synonym would enable an attacker to circumvent other systems, but will not defeat our system. We achieve robustness against these attacks with a new selection process. We also give theoretical and experimental results for these and other attacks on text sifting functions. Novel methods are presented to ensure that the identification process is secure against a malicious adversary under a given attack model.

Whereas in the randlet transform we used explicit randomization, this is not possible in text sifting because of the discrete nature of text. For example, in an

image it is usually possible to slightly perturb a given pixel's value without visually changing an image. It is generally *not* possible to change a single letter of a word and have the change be unnoticed by a person reading the word. Instead of randomization, we use secret hash functions to select letters and words from a document. Our choices are made deterministically, but an adversary without the secret key has no way of knowing which will be selected.

## 1.1 Previous Publication

The Secure QIM scheme of Chapter 2, as well as some of the modifications presented in Chapter 3 to make the Secure QIM scheme secure versus active adversaries, were joint work with Ton Kalker and were presented at the Information Hiding Workshop 2006 [27].

The randlet transform and the randlet-based image identification scheme of Chapter 4 was joint work with Ramarathnam Venkatesan and was presented at the 42nd Annual Allerton Conference on Communications, Control, and Computing 2004 [30].

The randlet watermarking scheme of Chapter 5, joint work with Ramarathnam Venkatesan, was presented at the 43rd Annual Allerton Conference on Communications, Control, and Computing 2005 [28].

The text sifting scheme of Chapter 6, joint work with Ramarathnam Venkatesan, was presented at the Australian Information Security Workshop 2005 [29].

# Chapter 2

# Secure Watermarking

When watermarking occurs for the purposes of digital rights management (DRM), watermark embedding is performed in a trusted environment, while watermark detection is performed "in the wild". That is, the watermark detector is assumed to be a trusted party, but it is generally operating in a hostile environment where the end-user would like to circumvent the DRM. This reality is in contrast to the standard watermark model, shown in Figure 2.1, where both the embedder and detector are assumed to operate in trusted environments and therefore share the same secret key. We adopt instead a watermarking model in which only the watermark embedder is in a trusted environment, shows in Figure 2.2.

It is usually not possible to make an entire system, such as a computer or DVD

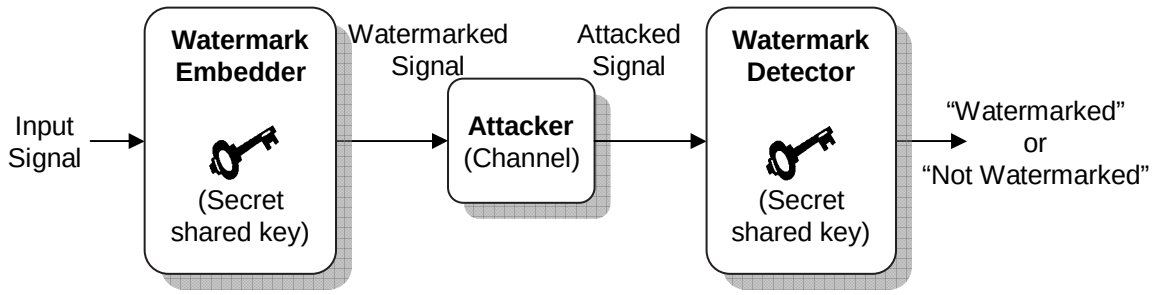


Figure 2.1: Standard watermarking model.

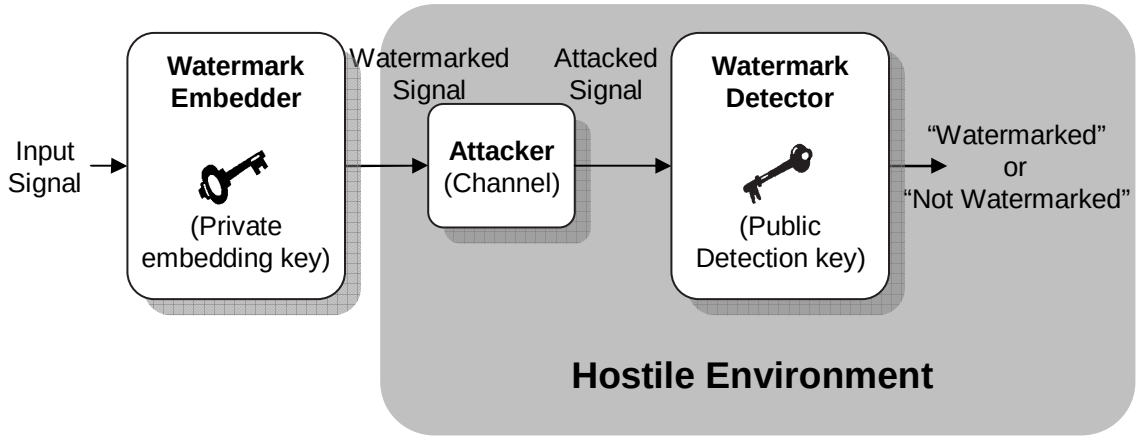


Figure 2.2: Watermarking model where watermark embedder is not assumed to be in a trusted environment.

player, into a trusted environment. Generally, the scale and complication of such systems makes this task difficult, and the cost is often prohibitive as well. Another way to keep the watermarking secret and functionality out of the hands of hostile parties is to embed it in a very small device which *can* be made physically secure. This device, for example a smartcard, would then be operated in a black-box manner. The problem is that such devices generally have very low computing capacity, and will be unable to perform watermark detection very quickly on their own. The strategy we use is to have the watermark detector work in an encrypted domain and use a trusted secure device, called the *secure module*, to finish the detection process. Figure 2.3 illustrates this model.

The focus of this chapter and Chapter 3 is on the design of the watermark detection algorithm and the interaction of the watermark detector with the secure module.

We note that the use of a secure module opens the door to new attacks. For example, an attacker could impersonate the secure module and always reply that signals are watermarked. Many attacks of this nature can be avoided by having the secure module sign its output. It would sign a hash of the query it receives concatenated with its response, allowing the verification of the response.

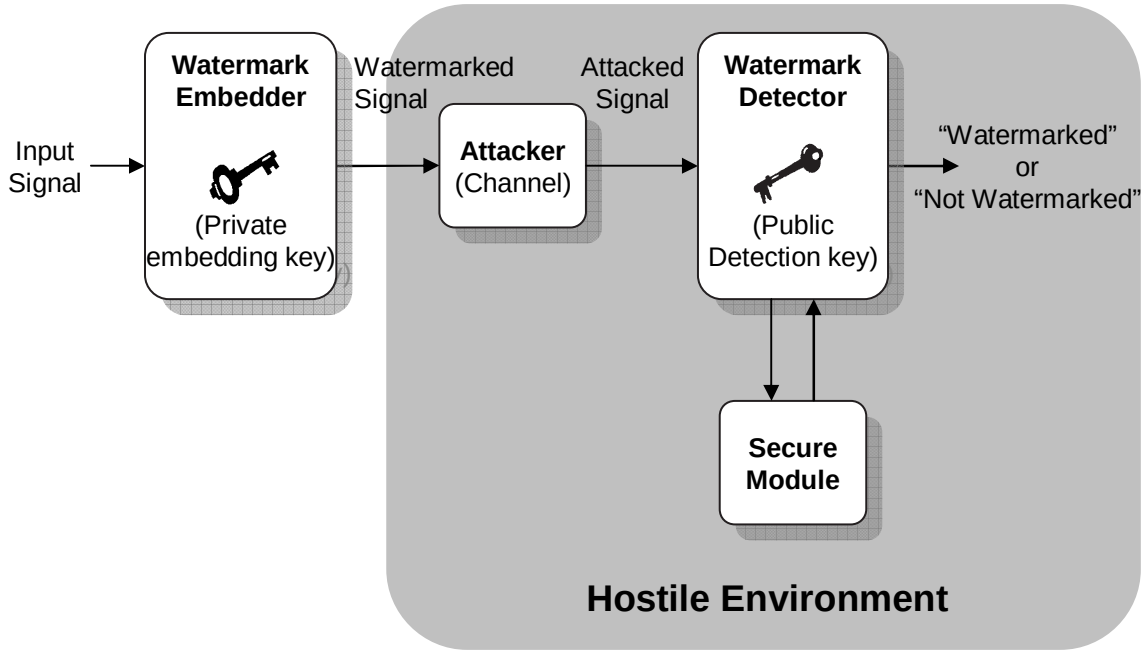


Figure 2.3: Model of a secure watermarking system utilizing a secure module.

Secure QIM (SQIM) uses public and private keys much like public key cryptosystems such as RSA. The watermarking private key is used by the watermark embedder to generate watermarks while the watermarking public key is used by the watermark detector to perform watermark detection in an encrypted domain. Finally, the secure module uses the watermarking private key to decrypt the results produced by the watermark detector. The secure module must be initialized by communication with the watermark embedder to receive the private key information. In a sense, this system is not truly asymmetric but rather semi-asymmetric, since the aid of a trusted third party (the secure module) is required.

As is usual for cryptographic systems, there is an adversary that must be considered. In an SQIM system the adversary is generally considered, without loss of generality, to be the watermark detector, which could be compromised. This chapter presents an SQIM system that is secure against an honest-but-curious watermark detector. This is a passive detector which will try to learn the watermark detection

secrets, but will do so while following along with the SQIM algorithm. Chapter 3 will present a modified version of SQIM which is secure against an active watermark detector, that is, a watermark detector which is not constrained to follow any set algorithm.

Our first goal is to ensure that the act of detecting a watermark reveals as little information as possible to the watermark detector. Because the secure module is low-power and low-bandwidth, our second goal is to ensure that the watermark detector takes on as much of the computational burden as is possible, and transmits as few bits to the secure module as possible. Our third goal is for the secure module carry as little information as possible. Ideally, it should not have to know any information other than cryptographic decryption keys, and should be able to respond correctly to any watermark detection queries where the watermark embedder used those keys. Not needing to know other information in advance, for example quantization information, will allow a secure module to be used with many different secure watermarking schemes instead of being hard-coded to only a few schemes.

Two cryptosystems are used to allow the watermark detector to perform the necessary calculations without learning any information about the watermarking secrets. These systems are *homomorphic*, meaning that an operation performed on ciphertexts corresponds to another operation performed on plaintexts. For example, in the Paillier cryptosystem (see Section 2.2.1), if $E(\cdot)$ is the encryption function, then $E(x)E(y) = E(x+y)$. The homomorphic properties of these cryptosystems are what make it possible for the watermark detector to run the algorithm without learning anything. While we use public key cryptosystems, it is interesting to note that there is no public access to either encryption or decryption. The only operations available to the watermark detector are homomorphic operations.

However, even though the watermark detector gains no extra knowledge through the detection process, knowledge of the presence or absence of watermarks is sufficient to mount *oracle attacks* (see Cox and Linnartz [13], Venturini [46], and Li and Chang [26], for example). The purpose of these attacks is to find the boundary separating watermarked signals from non-watermarked signals, and use this boundary to learn the watermarking secret. Such attacks are much more powerful than attacks on

the cryptosystems presented in this paper, and are possible whenever a watermark detector can test signals for watermarks. Such attacks are not yet possible against QIM watermarking systems, but we consider the possibility that such attacks may be discovered in the future.

One defense against oracle attacks is to increase the time required for watermark detection, effectively limiting the speed of the "oracle" (See Venturini [46]). This would not be possible with a fully assymetric watermarking scheme, since the speed of such a watermark detector would be limited only by the speed of the machine that is running it. A trusted secure module could have a built-in delay, or a limit on the number of watermark detections per minute, and could thereby help to slow the rate of convergence of oracle attacks. However, since many watermark detectors could be run in parallel, these measures would do more to deter casual attackers than those who are extremely determined. For example, a large-scale movie pirate could respond to a built-in delay by buying more DVD players.

Because the number of oracle queries necessary to mount an oracle attack is linear in the size of the watermarked signals, another defense against oracle attacks is to increase the size of the signals that are watermarked. For example, instead of splitting a large image into blocks and watermarking these blocks separately, the whole image could be watermarked at once. In general, the larger the signal, the more difficult it is to mount an oracle attack.

On the other hand, the use of a secure module introduces side channel attacks, for example timing attacks (see Kocher [25], and Brumley and Boneh [8]), and power attacks (see Kocher et al. [24]). In these attacks, the secure module is monitored externally to guess at the operations occurring internally. An implementation of SQIM would have to take side channel attacks into account, but a detailed discussion of these attacks is beyond the scope of this thesis.

Quantization Index Modulation (QIM), developed by Chen and Wornell [11], embeds a watermark into an signal by manipulating the signal so that transform coefficients are quantized in a specific manner. A watermark detector transforms a signal and checks to see if the transform coefficients are appropriately quantized. There are

two phases to securely detecting a QIM watermark. First a *hidden transform* is performed on the signal, and second the transform coefficients are quantized via *hidden quantization*. After hidden quantization the secure module counts the number of watermarked transform coefficients and reveals whether a threshold of the coefficients were watermarked. The SQIM algorithm presented in this chapter only works for quantizers with step size 2. In Chapter 3 we present a version of the SQIM algorithm that will work with arbitrary step sizes.

Attempts have been made at completely asymmetric watermarking schemes (see Eggers et al. [15] and Hachez and Quisquater [19]), but these have generally not been completely successful. Another specific method of performing asymmetric watermarking involves multi-round zero knowledge proofs (see Adelsbach and Sadeghi [4], for example). Kalker [22] introduced the idea of using a secure module to enable semi-public key watermarking, using a variant of the Paillier cryptosystem to perform secure spread spectrum watermarking. In comparison with the spread spectrum scheme, a SQIM scheme must implement a nonlinear operation in an encrypted domain, namely quantization.

Section 2.1 outlines the QIM watermarking scheme. Section 2.2 reviews homomorphic cryptography and introduces the two cryptosystems used in this paper. Section 2.3 discusses how to perform a hidden transform, while Section 2.4 discusses how to perform hidden quantization. Section 2.5 presents the full Secure QIM system. Finally, in Section 2.6 we discuss the efficiency of SQIM.

## 2.1 QIM

We consider a simple variant of QIM with a single scalar quantizer. Our purpose is not to improve the watermarking aspects of QIM, but to ensure that watermark detection is secure. Therefore, watermark embedding is not changed at all, and watermark detection is changed only in that all calculations are performed in a secure manner. We are only concerned with whether or not a signal was watermarked, so we do not use the watermark to embed data into a signal.

Embedding a watermark into a signal involves changing the signal so that the

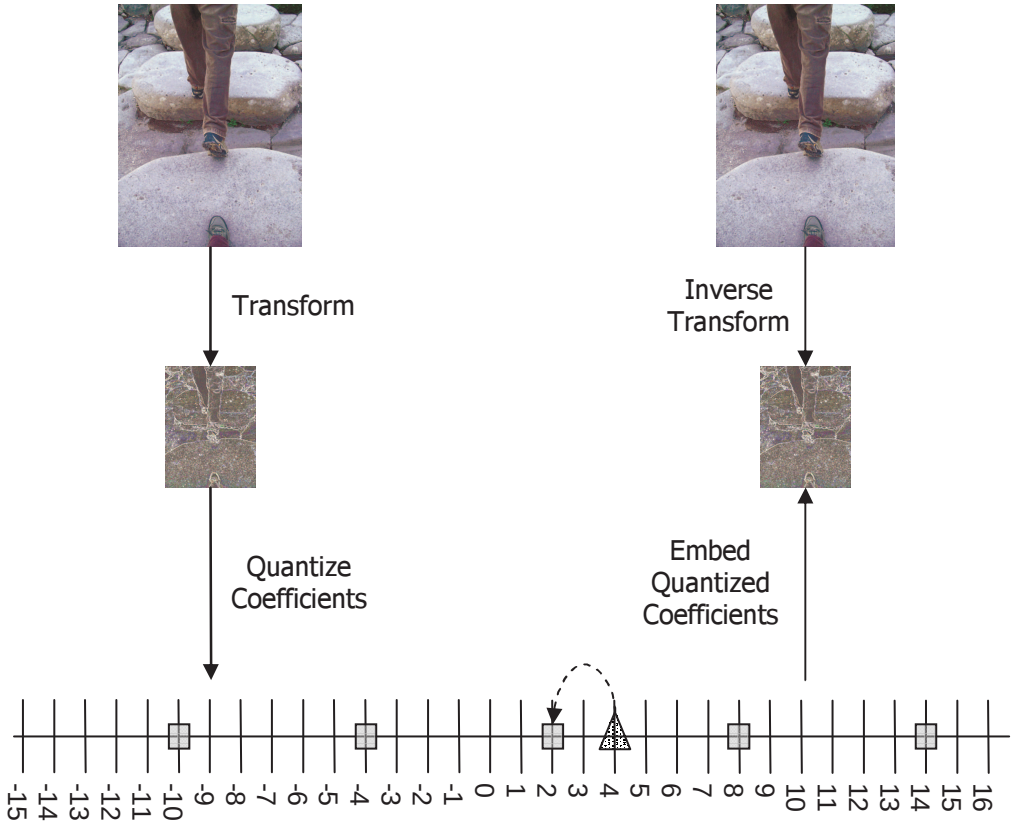


Figure 2.4: QIM watermark embedding.

transform values are quantized. The most straightforward approach is to transform the signal, quantize the transform coefficients, and perform the inverse transform. Figure 2.4 illustrates this process. Other embedding schemes, such as distortion-compensated QIM [11], may also be used. Secure QIM can be used with any QIM watermark embedding scheme.

To detect a watermark in a signal, the signal is first transformed with a secret, random linear transform, for example a randlet transform, a DCT, or a wavelet transform. For every transform coefficient $c_i$, there is a secret quantizer, $Q_i$. If the distance from $c_i$ to the nearest quantization point on $Q_i$ is less than a threshold $\delta$, then the coefficient is considered watermarked. If a threshold $\tau$ of transform coefficients are watermarked, then the entire signal is considered to be watermarked. If not, the

signal is considered not watermarked. Let

$$F_\delta(x) = \begin{cases} 1 & \text{if } |x| \leq \delta \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Then a signal is considered watermarked if

$$\frac{1}{k}\sum_{j=1}^{k} F_{\delta_j}(Q_j(c_j) - c_j) \geq \tau. \tag{2.2}$$

Figure 2.5 illustrates an example of the watermark detection process.

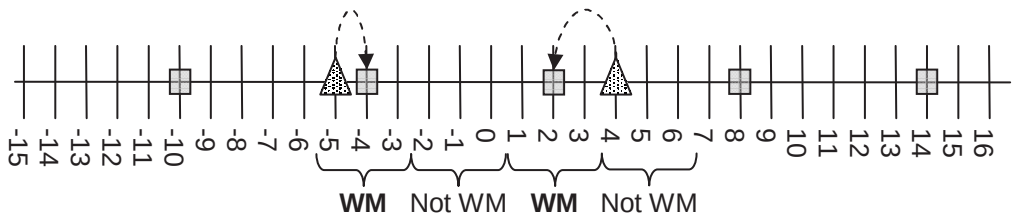


Figure 2.5: QIM watermark detection.

Note that it is not necessary to quantize all transform coefficients. See, for example, the spread-transform dither modulation (STDM) of Chen and Wornell [11]. A subset of coefficients can be chosen, and only those coefficients need be quantized during embedding or checked during detection. In the following discussion, when we refer to transform coefficients, we a referring to those which will be quantized and checked in QIM.

## 2.2 Homomorphic cryptosystems

We use two cryptosystems, the Paillier cryptosystem and the Goldwasser-Micali cryptosystem, both of which are *probabilistic public-key* cryptosystems. They are public key in that a public key is used to encrypt plaintext, while a private key is needed to decrypt a ciphertext, and the two keys are computationally not easily derived from each other. They are probabilistic in the sense that the same plaintext is represented

by a large number of ciphertexts. This is important when the range of possible plaintexts is small. For example, when encrypting 0 or 1, a non-probabilistic cryptosystem can produce only 2 possible ciphertexts, whereas a probabilistic cryptosystem can produce many different ciphertexts.

This last property is especially important in the current application. For example, if samples were in the range $[0, ..., 255]$, then there would be only 256 possible encryptions of the samples. This would make it much easier to break the system by looking at the transcripts of many watermark detections. Even relabelling the sample values would not solve the problem; there would be 256! possible relabellings, but statistical analysis could be used to easily find the correct one. With probabilistic cryptosystems, the values would be effectively *blinded*, so that this essentially brute-force searching attack would not be possible.

Both of these cryptosystems share another important property: they are homomorphic. This means that a mathematical operation performed on ciphertexts corresponds to a mathematical operation performed on plaintexts. For example, if $E(\cdot)$ corresponds to encryption in the Paillier cryptosystem, then we can write the homomorphism of the Paillier cryptosystem as

$$E(a_1)E(a_2) \equiv E(a_1 + a_2).$$

Homomorphic cryptosystems enable the watermark detector to perform calculations without explicitly knowing what is being calculated or finding out the results of the calculation. For example, given $\alpha = E(a)$, but not knowing the value of $a$, we could compute the encryption of $7a + 3$ as

$$\alpha^7 E(3) \equiv E(7a + 3).$$

Furthermore, with the right public values, it would be possible for us to compute, in the encrypted domain, any polynomial function of a given public input. For example, say $\alpha_1 = E(a_1)$, $\alpha_2 = E(a_2)$, and $\alpha_3 = E(a_3)$ were public, and we were asked

to compute $a_1x^2 + a_2x + a_3$ in the encrypted domain. We could do this as

$$\alpha_1^{(x^2)}\alpha_2^x\alpha_3 \equiv E(a_1x^2 + a_2x + a_3).$$

We would know the encryption of the polynomial, but have no knowledge of the actual value.

### 2.2.1 Paillier Cryptosystem

The Paillier cryptosystem is homomorphic, with multiplication of ciphertexts corresponding to the addition of the plaintexts. Furthermore, exponentiation of a ciphertext corresponds to multiplication of the plaintext. We present a very brief summary of the system. See Paillier [36] for more details.

Optimizations to the Paillier cryptosystem are discussed by Catalano et al. [10], Damgård and Jurik [14], and Kalker [22].

**Key Generation:** Let $N = pq$, where $p$ and $q$ are primes. Choose $g \in \mathbb{Z}^*_{N^2}$ such that the order of $g$ is divisible by $N$. Any such $g$ is of the form $g \equiv (1+N)^a b^N \bmod N^2$ for a pair $(a, b)$, where $a \in \mathbb{Z}_N$ and $b \in Z^*_N$. Note that $(1 + N)^a \equiv 1 + aN \bmod N^2$, so $g \equiv (1 + aN)b^N \bmod N^2$. Let $\lambda = \mathrm{lcm}(p - 1, q - 1)$. The public key is $(g, N)$, the private key is $\lambda$.

**Encryption:** For message $m$ and blinding factor $r \in \mathbb{Z}^*_N$, Paillier encryption is defined as

$$E_P(m, r; g, N) = g^m r^N \bmod N^2.$$

**Decryption:** In the Paillier cryptosystem, decryption is more complicated than encryption. First note that for any $x \in \mathbb{Z}^*_{N^2}$,

$$\begin{aligned} x^\lambda &\equiv 1 (\mathrm{mod}\, N), \\ x^{N\lambda} &\equiv 1 (\mathrm{mod}\, N^2). \end{aligned}$$

Given $c = E_P(m, r; g, N) = g^m r^N \bmod N^2$, we can see that

$$\begin{aligned} c^\lambda &\equiv g^{m\lambda} r^{N\lambda} \\ &\equiv (1+N)^{am\lambda} b^{\lambda N m} \\ &\equiv 1 + am\lambda N \pmod{N^2}. \end{aligned}$$

Note also that $g^\lambda \equiv [(1+N)^a b^N]^\lambda \equiv 1 + a\lambda N \pmod{N^2}$. Therefore,

$$\frac{(c^\lambda \mod N^2) - 1}{N} = a\lambda m \quad \text{and} \quad \frac{(g^\lambda \mod N^2) - 1}{N} = a\lambda$$

To simplify, let $f_N(x) = \frac{(x \mod N^2)-1}{N}$. Then we decrypt by computing

$$m = D_P(c; g, \lambda, N) = \frac{f_N(c^\lambda)}{f_N(g^\lambda)} \bmod N.$$

**Additive Homomorphism:** This homomorphism allows two unknown plaintexts to be added by multiplying their corresponding ciphertexts. For unknown $m_1$, $m_2$, compute

$$E_P(m_1, r_1; g, N) \cdot E_P(m_1, r_1; g, N) = E_P(m_1 + m_2, r_1 r_2; g, N).$$

**Multiplicative Homomorphism (multiplication by a constant):** This homomorphism allows an unknown plaintext to be multiplied by a known constant by exponentiating its ciphertext. For unknown $m$ and known $k$ compute

$$E_P(m, r; g, N)^k = E_P(mk, r^k; g, N).$$

### 2.2.2 Quadratic Residues

The Goldwasser-Micali cryptosystem is based on *quadratic residues*. A number is a quadratic residue modulo an odd prime $p$ if it is the square of some number modulo $p$. Before we present the Goldwasser-Micali cryptosystem, we discuss quadratic residues, which form the basis of that system.

**Definition 2.2.1 (Legendre symbol)** *The Legendre symbol is defined as*

$$\left(\frac{x}{p}\right) = \begin{cases} 0 & \text{if } x \equiv 0 \pmod{p} \\ 1 & \text{if } x \text{ is a quadratic residue modulo } p \\ -1 & \text{if } x \text{ is a quadratic non-residue modulo } p \end{cases}$$

By Euler's criterion [42], we compute $\left(\frac{x}{p}\right) = x^{\frac{p-1}{2}} \pmod{p}$.

In the case of a composite modulus, the *Jacobi symbol* is used instead of the Legendre symbol.

**Definition 2.2.2 (Jacobi symbol)** *For $N = pq$, where $p$ and $q$ are odd primes, the Jacobi symbol is*

$$\left(\frac{x}{N}\right) = \begin{cases} 0 & \text{if } \gcd(x, N) > 1 \\ 1 & \text{if } \left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) \\ -1 & \text{if } \left(\frac{x}{p}\right) = -\left(\frac{x}{q}\right) \end{cases}$$

**Definition 2.2.3 (QR)** *Let $QR(N)$ be the set of all quadratic residues modulo $N$.*

**Lemma 2.2.1** *$x$ is a quadratic residue modulo $N$ iff $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = 1$. If $x$ is a quadratic residue modulo $N^2$ then it is a quadratic residue modulo $N$.*

**Proof** If $x \in \mathrm{QR}(N)$ then $x = y^2 + kN = y^2 + kpq$ for some $y, k$, so $x \equiv y^2 \pmod{p}$ and $x \bmod p \in \mathrm{QR}(p)$. The same holds for $q$, so $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = 1$. Given $x$ such that $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = 1$, we know that there exist $a$ and $b$ such that $a^2 \equiv x(\bmod\ p)$ and $b^2 \equiv x(\bmod\ q)$. By the Chinese Remainder Theorem [23], there exists a $y$ such that $y \equiv a(\bmod\ p)$ and $y \equiv b(\bmod\ q)$. Since, $y^2 \equiv x(\bmod\ p)$ and $y^2 \equiv x(\bmod\ q)$, we know that $y^2 \equiv x(\bmod\ N)$, and therefore $x \in \mathrm{QR}(N)$. If $x \in \mathrm{QR}(N^2)$ then $x = y^2 + kN^2$ for some $y, k$, so $x \bmod N \in \mathrm{QR}(N)$. □

**Definition 2.2.4 ($\tilde{\mathrm{QR}}$)** *$x$ is a pseudosquare modulo $N$ if $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1$. Define $\tilde{QR}(N)$ to be the set of pseudosquares modulo $N$.*

It is easy to calculate Jacobi symbols, even if the factors of $N$ are unknown (see Koblitz [23]). However, if the factorization of $N$ is unknown, it is not always easy to determine quadratic residuosity. For any $x \in \mathrm{QR}(N) \cup \tilde{\mathrm{QR}}(N)$, $\left(\frac{x}{N}\right) = 1$, but determining if $x \in \mathrm{QR}(N)$ is a classical hard problem in cryptography and is assumed to be impossible without factoring $N$. If $p$ and $q$ are known, it is easy to determine if such an $x$ is a quadratic residue by computing $\left(\frac{x}{p}\right) = x^{\frac{p-1}{2}} (\mathrm{mod} p)$ as above.

### 2.2.3 The Goldwasser-Micali Cryptosystem

The Goldwasser-Micali cryptosystem was developed in 1984 by Goldwasser and Micali [18]. It encrypts a single bit of information and is homomorphic in that multiplying ciphertexts corresponds to finding the XOR of the plaintexts.

**Key Generation:** Let $N = pq$, where $p$ and $q$ are safe primes. Choose $g \in \tilde{\mathrm{QR}}(N)$. $N$ and $g$ are public while the factorization of $N$ is private.

**Encryption:** Encryption takes as input a single bit $b$ and a random blinding factor $r \in \mathbb{Z}_N^*$. Encryption in the Goldwasser-Micali cryptosystem is defined as

$$E_{\mathrm{GM}}(b, r; g, N) = g^b r^2 \bmod N.$$

**Decryption:** Decryption is defined as

$$D_{\mathrm{GM}}(x; p, q) = \begin{cases} 0 & \text{if } x \in \mathrm{QR}(N) \\ 1 & \text{if } x \in \tilde{\mathrm{QR}}(N) \end{cases}$$

If the factorization of $N$ is known, decryption can easily be done by computing $\left(\frac{x}{p}\right) = x^{(p-1)/2} \bmod p$. Otherwise decryption is not possible, since it requires distinguishing members of $\mathrm{QR}(N)$ from members of $\tilde{\mathrm{QR}}(N)$ (see Section 2.2.2).

**XOR Homomorphism:** This system is homomorphic in that multiplying ciphertexts is equivalent to XORing plaintexts. Note the following congruences:

$$E_{\mathrm{GM}}(b_1, r_1; g, N) \cdot E_{\mathrm{GM}}(b_2, r_2; g, N) \equiv g^{b_1+b_2}(r_1 r_2)^2 \pmod{N}.$$

$$\equiv E_{\mathrm{GM}}(b_1 \oplus b_2, r_1 r_2; g, N) \pmod{N}.$$

The last equality holds because only the least bit of $b_1 + b_2$ matters in determining quadratic residuosity, and $\oplus$ is equivalent to modulo 2 addition.

## 2.3 Phase I: Hidden Transform

The first phase of Secure QIM is a hidden linear transform. This means that the watermark detector takes the sample values from the signal and performs a transform on the sample values without learning the transform or the resulting transform coefficients.

Using the Paillier cryptosystem, we know how to perform addition and multiplication in the plaintext domain by performing the corresponding operations of multiplication and exponentiation in the ciphertext domain. Let a signal consist of $m$ samples, $\mathbf{y} = (y_1, ..., y_m)^T$. The random transform takes $\mathbf{y}$ as input and produces $n$ transform coefficients, $\mathbf{t} = (t_1, ..., t_n)^T$. Let the watermark embedder choose an orthogonal transform $\mathbf{S} = \{s_{ij}\}$, for $i = 1...n$ and $j = 1...m$, and let $\mathbf{s}_i$ be row $i$ of the transform. Note that $\mathbf{t} = \mathbf{S}\mathbf{y}$ and $t_i = \mathbf{s}_i \cdot \mathbf{y}$.

The watermark detector is not allowed to know any of the values of $\mathbf{S}$, nor any of the values of $\mathbf{t}$. This is achieved by performing all the calculations in the Paillier encrypted domain. First, the watermark embedder chooses $N = pq$, where $p$ and $q$ are primes, and chooses a random $g \in \mathbb{Z}^*_{N^2}$ such that the order of $g$ is divisible by $N$. Next, for $i \in [1, n], j \in [1, m]$, it generates random $\beta_{ij} \in \mathbb{Z}^*_N$. The public key consists of encryptions of the transform matrix $\mathbf{V} = \{v_{ij}\}$ where $v_{ij} = E_P(s_{ij}, \beta_{ij}; g, N)$.

The watermark detector wants to find $\mathbf{c} = (c_1, ..., c_m)$, the hidden transform coefficients. It does so by computing

$$c_i = \prod_j (v_{ij})^{y_j} \bmod N^2.$$

For later convenience in notation, define $w_i = \prod_{j=1}^m \beta_{ij}^{y_j}$. Then by the homomorphic

properties of the Paillier cryptosystem, we have

$$c_i = \prod_j v_{ij}^{y_j} \bmod N^2 = \prod_j E_P(s_{ij}, \beta_{ij}; g, N)^{y_j} \bmod N^2$$

$$= \prod_j E_P(s_{ij} y_j, \beta_{ij}^{y_j}; g, N) \bmod N^2 = E_P(\sum_j s_{ij} y_j, \prod_j \beta_{ij}^{y_j}; g, N)$$

$$= E_P(\mathbf{s}_i \cdot \mathbf{y}, w_i; g, N) = E_P(t_i, w_i; g, N).$$

## 2.4 Phase II: Hidden Quantization

This section will present a simplified version of the hidden quantization scheme, uncoupled from the hidden transform, for a clearer presentation. The full version will be presented in Section 2.5.

The watermark embedder chooses $N = pq$, where $p$ and $q$ are safe primes, and $g \in \tilde{\mathrm{QR}}(N)$. It also chooses private quantization values $\mathbf{q} = (q_1, ..., q_n)$ where each $q_i \in \{0, 1\}$, and blinding values $\gamma = (\gamma_1, ..., \gamma_n)$ where each $\gamma_i \in \mathbb{Z}_N^*$, and calculates $\mathbf{k} = (k_1, ..., k_n)$, $k_i = E_{\mathrm{GM}}(q_i, \gamma_i; g, N)$. It publishes $g$, $N$, and $\mathbf{k}$, and reveals the value of $p$ to the secure module.

The watermark detector knows the public values $g$, $N$, and $\mathbf{k}$. Assume in this section that it has $n$ unencrypted transform coefficients, $\mathbf{t} = (t_1, ..., t_n)$. If the signal is watermarked, these coefficients will each quantized so that $t_i \equiv q_i (\mathrm{mod} 2)$, but for any given coefficient, the watermark detector does not know the correct quantization value. The key point to notice is that if the signal is watermarked, $t_i \oplus q_i \equiv 0 (\bmod 2)$.

First, the watermark detector chooses $\alpha = (\alpha_1, ..., \alpha_n)$, with $\alpha_i \in \mathbb{Z}_N^*$, and encrypts the transform coefficients,

$$c_i = E_{\mathrm{GM}}(t_i \bmod 2, \alpha_i; g, N).$$

Then it computes

$$\begin{aligned} f_i &= c_i k_i \\ &= E_{\mathrm{GM}}(t_i \bmod 2, \alpha_i; g, N) E_{\mathrm{GM}}(q_i, \gamma_i; g, N) \\ &= E((t_i \bmod 2) \oplus q_i, r_i; g, N). \end{aligned}$$

Note that $f_i$ is a Goldwasser-Micali encryption of 0 if $t_i$ is watermarked, 1 otherwise.

The secure module has a threshold function, $T(n)$. It is given as input $f_1, ..., f_n$, decrypts each $f_i$, sums the values, and announces that the data is watermarked if

$$\sum_{i=1}^{n} D_{\mathrm{GM}}(f_i; p, q) \leq T(n).$$

## 2.5 Secure QIM

This section presents the full system, in which the watermark detector performs a hidden transform on the input data and then quantizes the transform coefficients while still in the encrypted domain. It is a combination of the systems from Sections 2.3 and 2.4 with a careful choice of $g$ and the blinding factors so that the ciphertexts of the hidden transform can be used for hidden quantization. In Sections 2.5.1, 2.5.2, and 2.5.3, we present the full SQIM scheme. Figure 2.6 illustrates the Secure QIM protocol.

### 2.5.1 Initialization

The watermark embedder chooses $N = pq$, where $p$ and $q$ are safe primes. Recall that for such $N$, $\lambda = \textsc{lcm}(p-1, q-1)$. $g \in \mathbb{Z}^*_{N^2}$ is chosen so that $g \bmod N \in \tilde{\mathrm{QR}}(N)$ and the order of $g$, denoted $\textsc{ord}(g)$, is $kN$, where $k|\lambda$. All such $g$ can be generated as follows. Choose $a \in Z^*_N$, so $\textsc{gcd}(a, N) = 1$, and $b \in \tilde{\mathrm{QR}}(N)$. Let $g = (1+N)^a b^N \bmod N^2$. Because $(1+N)^a \equiv 1 + aN \bmod N$, we can write

$$g = (1 + aN) b^N \bmod N^2.$$

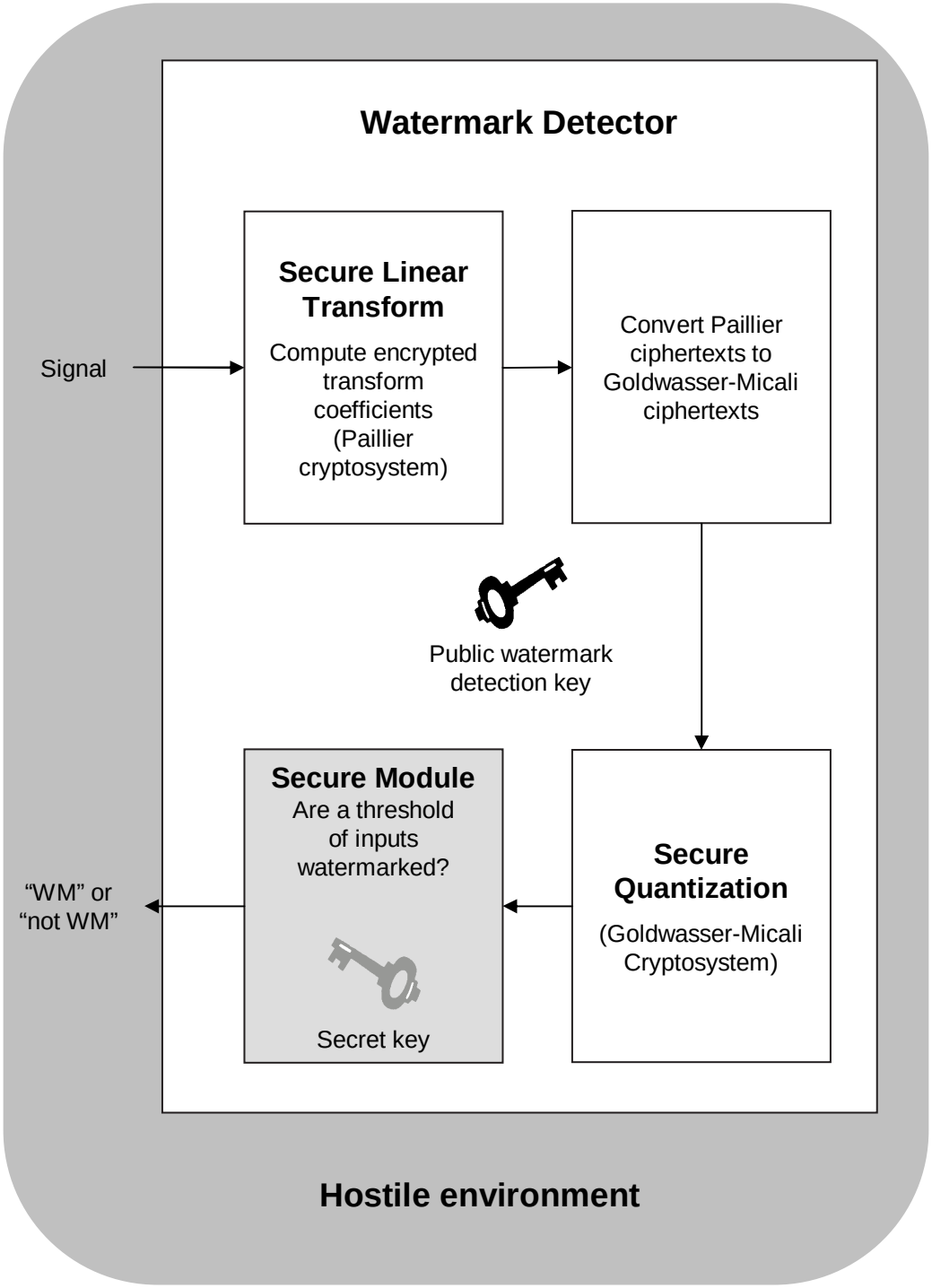


Figure 2.6: The Secure QIM protocol.

*Claim.* $g \bmod N \in \tilde{\mathrm{QR}}(N)$ and $\textsc{ord}(g) = kN$.

**Proof** Notice that $g \equiv b^N \pmod{N}$. Since $N$ is odd, $b^N \bmod N \in \tilde{\mathrm{QR}}(N)$, so $g \bmod N \in \tilde{\mathrm{QR}}(N)$. Because $\textsc{ord}(1+N) = N$ and $\textsc{gcd}(a, N) = 1$, $\textsc{ord}(1+N)^a = N$. Let $k = \textsc{ord}(b^N)$. Since $b^N \in Z_N^*$, $k|\phi(N)$, and since $N$ and $\phi(N)$ share no factors, $\textsc{gcd}(k, N) = 1$. Therefore, $\textsc{ord}(g) = \textsc{ord}(1 + N)\textsc{ord}(b^N) = kN$. □

### 2.5.2 Watermark Embedding

The watermark embedder chooses an orthogonal transform $\mathbf{S} = \{s_{ij}\}$, for $i = 1...n$ and $j = 1...m$. Let $\mathbf{s}_i$ be row $i$ of the transform. The embedder also chooses $\mathbf{q} = (q_1, ..., q_n)$, with each $q_i \in \{0, 1\}$. It takes as input the signal $\mathbf{x} = (x_1, ..., x_m)$ and produces a watermarked signal $\mathbf{y} = (y_1, ..., y_n)$ such that for all $i$,

$$\mathbf{s}_i \cdot \mathbf{y} \equiv q_i \pmod{2}.$$

For $i \in [1, n], j \in [1, m]$, $\beta_{ij}$ is chosen such that $\beta_{ij} \in \mathrm{QR}(N)$. For $i \in [1, n]$, $\gamma_i$ is chosen so that $\gamma_i \in \mathrm{QR}(N)$. Let $\mathbf{V} = \{v_{ij}\}$ where $v_{ij} = E_P(s_{ij}, \beta_{ij}; g, N)$, and $\mathbf{k} = (k_1, ..., k_n)$ where $k_i = E_{\mathrm{GM}}(q_i, \gamma_i; g, N)$.

The SQIM public watermarking key is $(N, \mathbf{V}, \mathbf{k})$.

### 2.5.3 Watermark Detection

First, the watermark detector finds the vector of encrypted transform coefficients $\mathbf{c} = (c_1, ..., c_m)$. Recall that $t_i = \mathbf{s}_i \cdot \mathbf{y}$. Let $w_i = \prod_{j=1}^{m} \beta_{ij}^{y_j}$. Then

$$c_i = \prod_j v_{ij}^{y_j} \bmod N^2 = E_P(t_i, w_i; g, N) = g^{t_i} w_i^N \bmod N^2.$$

At this point we change from looking at ciphertexts modulo $N^2$ and begin looking at them modulo $N$. This is done so that the ciphertexts will be compatible with the Goldwasser-Micali cryptosystem.

We will now show that if $t_i \bmod 2 = 0$, then $c_i \bmod N \in \mathrm{QR}(N)$, otherwise

$c_i \bmod N \in \tilde{\mathrm{QR}}(N)$. Since each $\beta_{ij} \in \mathrm{QR}(N)$, then $w_i \in \mathrm{QR}(N)$ and therefore $(w_i)^N \bmod N \in \mathrm{QR}(N)$. If $t_i \bmod 2 = 0$, then $g^{t_i} \bmod N \in \mathrm{QR}(N)$. Otherwise, since $g \bmod N \in \tilde{\mathrm{QR}}(N)$, $g^{t_i} \bmod N \in \tilde{\mathrm{QR}}(N)$. Therefore, if $t_i \bmod 2 = 0$ then $c_i \in \mathrm{QR}(N)$ otherwise, $c_i \in \tilde{\mathrm{QR}}(N)$. In both cases $\left(\frac{c_i}{N}\right) = 1$. Therefore,

$$c_i \bmod N = E_{\mathrm{GM}}(t_i \bmod 2, w_i^N g^{2\lfloor \frac{t_i}{2} \rfloor}; g, N).$$

Now we begin hidden quantization. Let $f_i = c_i k_i \bmod N$ and $z_i = w_i^N g^{2\lfloor \frac{t_i}{2} \rfloor}$.

$$f_i = c_i k_i \bmod N = E_{\mathrm{GM}}(t_i \bmod 2, z_i; g, N) E_{\mathrm{GM}}(q_i, \gamma_i; g, N) \bmod N$$

$$= E_{\mathrm{GM}}((t_i \bmod 2) \oplus q_i, \gamma_i z_i; g, N).$$

So, $D_{\mathrm{GM}}(f_i; p, q) = (t_i \bmod 2) \oplus q_i$, which is 0 if $t_i$ was correctly quantized, 1 otherwise.

The secure module is given as input $f_1, ..., f_n$ and knows a threshold function $T(n)$. It decrypts each $f_i$, sums the values, and announces that the data is watermarked if

$$\sum_{i=1}^{n} D_{\mathrm{GM}}(f_i; p, q) \leq T(n).$$

## 2.6 Efficiency

In this section we will compare the efficiency of standard QIM with that of Secure QIM. Note that in both cases, the secure module starts off knowing the Paillier and Goldwasser-Micali private keys, but does not know any transform matrices or quantization values. In standard QIM, performed with a random transform on a secure module, the secure module is given a signal, an encrypted transform matrix, and encrypted quantization values, and performs QIM watermark detection by itself.

In our analysis we are concerned with the communication and computation required of the secure module. Let signals be of length $m$ and let there be $n$ transform coefficients. The samples of the signal and elements of the transform matrix are $k$-bit numbers and encryptions modulo $N$ have $\ell = \log_2 N$ bits.

### Communication

In standard QIM, the secure module can receive all the information in an efficiently encrypted form. As such, we will calculate the number of bits that would be required unencrypted, and will assume that encryption adds negligible overhead, an assumption which is favorable to standard QIM. The secure module receives the signal ($mk$ bits), the encrypted transform matrix ($mnk$ bits), and the encrypted quantization values ($n$ bits), for a total of $mnk + mk + n$ bits. The $mnk$ term will dominate.

In SQIM, each number sent to the secure module is encrypted modulo $N$. The secure module receives $n$ $\ell$-it numbers, $n\ell$ bits total.

In comparison, standard QIM requires approximately $\frac{mk}{\ell}$ times more bits than SQIM. Since $m$ is the number of samples in the signal, this is a very large difference. For example, consider a signal of $m = 10^5$ samples, with $k = 24$, $n = 25$, and $\ell = 1024$. Then SQIM requires 3.1 kilobytes while standard QIM requires 7.4 megabytes, 2,400 times larger.

### Computation

The computational costs are harder to compute and more dependent on specific implementation. We estimate the cost of standard QIM as the cost of performing the transform. Assuming straightforward matrix multiplication, this will have a running time of $O(nmk^2)$. We estimate the cost of SQIM based on the total number of decryptions. There are $n$ decryptions, which results in a running time of $O(n\ell^3)$. $\ell^3$ is very large, but $\ell$ is fixed based on security needs. $k$ is generally in a small range, say 8 to 24 bits. So, the relative performance is highly dependent on the number of samples in the signal. With a relatively small number of samples, standard QIM will be faster, while with a relatively large number of samples SQIM will be faster.

## 2.7 Security

Given the security of the Goldwasser-Micali and Paillier cryptosystems, our system is secure and reveals the minimum possible information.

### 2.7.1 Minimum Possible Information

Any watermarking system that is susceptible to oracle attacks can not justly be called zero knowledge, because the watermark detector gains useful knowledge with every response from the secure module, namely the watermarked status of a signal. Known oracle attacks work against linear watermarks, and so currently none have been found which work against QIM systems because of the non-linear quantization operations. However, it is not unreasonable to assume that oracle attacks will eventually be found which work against QIM watermarking systems.

Even assuming the existence of an oracle attack against SQIM, we point out that an SQIM system reveals the *minimum possible information*. If the watermark detector is honest (passive), the only information it can learn is the watermarked status of each signal it performs detection upon. As a watermark detector, this is the minimum information it is able to learn, and assuming the semantic security of the Paillier and Goldwasser-Micali cryptosystems, this is all it learns. If this information is enough to mount an oracle attack, then the minimum possible information was leaked. On the other hand, if no oracle attack is possible against QIM systems, then SQIM is a zero knowledge watermarking system.

### 2.7.2 Proof of Security

Again, the possible future discovery of oracle attacks on QIM systems makes security statements difficult. If there turns out not to be any oracle attacks against QIM systems, then the zero knowledge property of SQIM and the semantic security of the Paillier and Goldwasser-Micali cryptosystems is enough to prove security. However, assuming that an oracle attack will one day be found, we claim that SQIM is secure against passive watermark detectors modulo oracle attacks. That is, the cryptographic components of SQIM are secure; it is only the continuous nature of the inputs which make possible oracle attacks. Below we prove the cryptographic security of the various elements of an SQIM system.

In both the Paillier and Goldwasser-Micali cryptosystems, any properly-formed $N$ with large-enough factors is secure. The security of the Paillier cryptosystem is

independent of the choice of $g$, and any $g$ such that $g \bmod N \in \tilde{\mathrm{QR}}(N)$ is secure for the Goldwasser-Micali cryptosystem. Therefore, even though we choose $g$ in a unique manner, it is still secure.

The security of Paillier encryption depends on the blinding factors, which we also choose in a unique manner. Since the blinding factors in standard Paillier encryption are chosen at random from $Z_N^*$, choosing them from a subset of non-negligible size does not introduce security problems; if it did, then choosing them at random from $Z_N^*$ would have a non-negligible chance of encountering the same problems. A problem would exist if the subset were small enough to make a brute-force search possible, but $|\mathrm{QR}(N)| = \frac{1}{4}|\mathbb{Z}_N^*|$, so this is not a concern. Therefore, our choice of blinding factors is secure.

The only public watermarking values are either Paillier ciphertexts or Goldwasser-Micali ciphertexts which are acted upon homomorphically. Since $N$, $g$, and the blinding factors are chosen securely, these ciphertexts are as secure as the Paillier and Goldwasser-Micali cryptosystems, respectively.

## 2.8 Conclusion

We have presented a Secure QIM system, one in which most of the work of watermark detection can be performed in the open without any information about the private key being leaked. A secure module, such as a smartcard, is used to perform the final portion of watermark detection, revealing only if the signal is watermarked and no other information about the watermark. Our hidden transform utilizes the Paillier cryptosystem, while our hidden quantization uses the Goldwasser-Micali cryptosystem, and the entire system has provably maximal security against passive attackers.

# Chapter 3

# Secure Watermarking Extensions

We now present several improvements to the secure watermarking system of Chapter 2, an improvement to a related system, the Secure Spread Spectrum (SSS) system of Kalker [22].

One of the biggest limitation of SQIM is that all quantizers must have step size 2. This is to ensure that the watermark detector can homomorphically quantize coefficients to the appropriate quantization point. In Section 3.1 we change paradigms, and instead of homomorphically moving each coefficient to the appropriate quantization *point*, the watermark detector homomorphically moves each transform coefficient to the appropriate quantization *region*. By allowing loosening the constraints on the watermark detector in this manner, we enable the use of quantizers with arbitrary step size. This new system is known as generalized SQIM (G-SQIM).

In Section 3.2 we address the issue of security against active watermark detectors, the adversaries in an SQIM system. The security of an SQIM system can not be proven against active adversaries because the proof of security fails if the attacker strays from the SQIM algorithm. The point of weakness in these proofs is the secure module, because the only new information that is revealed to the watermark detector is the single bit returned by the secure module. All other information computed during watermark detection is simply homomorphic manipulations of the signal and the public watermarking key, and thus tells the watermark detector nothing.

When a watermark detector submits a legitimate query to the secure module,

then, it is operating in an honest-but-curious manner, and security is proven. The proofs fail only when the watermark detector may submit illegitimate queries if it desires. To make a system which is secure against active adversaries, we therefore modify the SQIM system so that it will only respond to legitimate queries, and will reject all others. We call this modified system Verified G-SQIM (VG-SQIM).

Finally, in Section 3.3 we introduce Kalker's [22] Secure Spread Spectrum system, and show how to modify it to make it secure against active adversaries.

## 3.1 Generalized SQIM (G-SQIM)

We now present a variant of SQIM known as generalized SQIM (G-SQIM), related to SQIM but able to use quantizers with arbitrary step size. The key difference is that the watermark detector no longer performs hidden quantization by moving each transform coefficient to the appropriate quantization *point*, but instead moves each coefficient to the appropriate quantization *region*, and the secure module determines if each coefficient is in the correct region. This is a trivial amount of extra work for the secure module compared to the work it performs in decryption. Furthermore, a modified version of the Paillier cryptosystem due to Catalano et al. [10] is used. This cryptosystem has more efficient decryption than the unmodified Paillier cryptosystem, so G-SQIM is only a factor of 2 slower than SQIM.

The quantizers in G-SQIM are uniform scalar quantizers with step size $\Delta$ which have been randomly shifted. For each transform coefficient $c_j$, the quantizer $Q_j$ is offset by a random shift $\alpha_j \in [0, 1, ..., \Delta_j]$. In SQIM, the possible values of $\alpha_j$ were 0 and 1, as compared to the much larger range in G-SQIM. The quantization points of $Q_j$ are

$$Q_j = \{..., -2\Delta + \alpha_j, -\Delta + \alpha_j, \alpha_j, \Delta + \alpha_j, 2\Delta + \alpha_j, ...\}.$$

Quantization regions are formed around each quantization point. Any coefficient within a distance of $\delta$ of a quantization point is considered watermarked, so each quantization region around a quantization point is labelled as `WM`. In between each `WM`

region is a region labelled `UW` (unwatermarked), and all coefficient in an `UW` region are considered not to be watermarked. All points are in either in a `WM` or `UW` region. The distance from the beginning of one `WM` the next is $\Delta$, likewise for `UW` regions. All `WM` regions are of the same size, as are all `UW` regions, however `WM` and `UW` regions need not be the same size. Figure 3.1 shows an example quantizer with $\alpha_j = 2$, $\Delta = 6$, and $\delta = 3$.

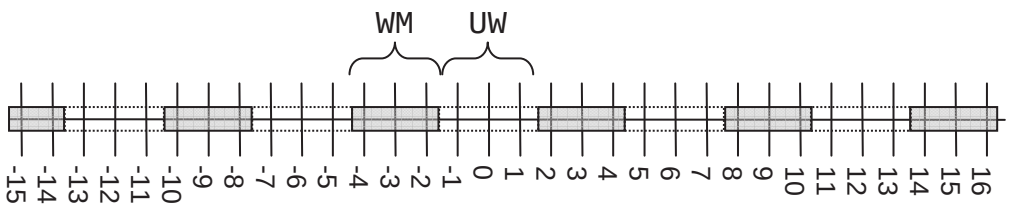


Figure 3.1: An example quantizer. In this case, $\alpha_j = 2$, $\Delta = 6$, and $\delta = 3$.

The secure module is given $\Delta$ and $\delta$ in encrypted format, but has no knowledge of the random shifts $\alpha_j$, instead using a canonical quantizer with no shift ($\alpha = 0$). During Hidden Quantization (Section 2.4), the watermark detector shifts the transform coefficients homomorphically to compensate for the difference between the actual quantizers (with $\alpha_j$) and the canonical quantizer (with $\alpha = 0$). After this shift, transform coefficients which are correctly watermarked will lie in a `WM` region of the secure module's canonical quantizer. Likewise, transform coefficients which are not watermarked will lie in a `UW` region.

Finally, the secure module will receive the encrypted, shifted transform coefficients, along with the encrypted values of $\Delta$ and $\delta$. It will decrypt all the values, and for each coefficient $c_i$ will calculate that the coefficient is quantized if

$$c_i mod \Delta \leq \delta. \tag{3.1}$$

SQIM uses an equivalent system with $\Delta = 2$ and $\delta = 1$, and with $\alpha_j \in \{0, 1\}$. In SQIM, shifting each coefficient $c_j$ by $\alpha_j$ is much easier, because it amounts to leaving $c_j$ unchanged if $\alpha_j = 0$, and computing $c_j \oplus 1$ if $\alpha_j = 1$. In this special case, the computation can be performed entirely by the watermark detector by using

the homomorphic properties of the Goldwasser-Micali cryptosystem. In the case of G-SQIM, the secure module calculates Equation 3.1 for each coefficient and in return an arbitrary choice of $\Delta$ and $\delta$ is allowed.

### 3.1.1 Initialization

The watermark embedder chooses $N = pq$, where $p$ and $q$ are safe primes. $e$ and $d$ are chosen for the modified Paillier cryptosystem as described in Section 3.1.4.

### 3.1.2 Watermark Embedding

The watermark embedder chooses an orthogonal transform $\mathbf{S} = \{s_{ij}\}$, for $i = 1...n$ and $j = 1...m$. Let $\mathbf{s}_i$ be row $i$ of the transform. Next it chooses $\Delta$ and $\delta$, the quantization parameters. The embedder also chooses $\alpha = (\alpha_1, ..., \alpha_n)$, with each $\alpha_i \in [0,\ Delta]$. It takes as input the signal $\mathbf{x} = (x_1, ..., x_m)$ and produces a watermarked signal $\mathbf{y} = (y_1, ..., y_n)$ such that for all $i$,

$$(\mathbf{s}_i \cdot \mathbf{y} - \alpha_i) \bmod \Delta \leq \delta.$$

Generally, the watermark embedder will produce a signal such that for all $i$,

$$(\mathbf{s}_i \cdot \mathbf{y} - \alpha_i) \bmod \Delta = \begin{cases} \frac{\delta}{2} - 1 \text{ or } \frac{\delta}{2} & \text{if } \delta \text{ even} \\ \frac{\delta - 1}{2} & \text{if } \delta \text{ odd,} \end{cases}$$

as this means putting the coefficients as close to the center of a watermarked region as possible.

For $i \in [1, n], j \in [1, m]$, $r_{ij} \in \mathbb{Z}_N^*$ and $r_i \in \mathbb{Z}_N^*$ are chosen at random, as are $r_\Delta$ and $r_\delta$. Let $\mathbf{V} = \{v_{ij}\}$ where $v_{ij} = E_P(s_{ij}, r_{ij}; g, N)$, and $\mathbf{k} = (k_1, ..., k_n)$ where $k_i = E_P(\Delta - \alpha_i, r_i; g, N)$. Let $\Psi = E_P(\Delta, r_\Delta; g, N)$ and let $\psi = E_P(\delta, r_\delta; g, N)$.

The G-SQIM public watermarking key is $(N, \Psi, \psi, \mathbf{V}, \mathbf{k})$.

### 3.1.3 Watermark Detection

First, the watermark detector finds the vector of encrypted transform coefficients $\mathbf{c} = (c_1, ..., c_m)$ as in SQIM. Recall that $t_i = \mathbf{s}_i \cdot \mathbf{y}$. Let $w_i = \prod_{j=1}^{m} r_{ij}^{y_j}$. Then

$$c_i = \prod_j v_{ij}^{y_j} \bmod N^2 = E_P(t_i, w_i; g, N).$$

Now that the transform coefficients have been calculated, they are homomorphically shifted to the canonical quantizer. Let $f_i = c_i k_i \bmod N^2$, then

$$\begin{aligned} f_i &\equiv c_i k_i \equiv E_P(t_i, w_i; g, N) E_P(\Delta - \alpha_i, r_i; g, N) \pmod{N^2} \\ &\equiv E_P(t_i + \Delta - \alpha_i, w_i r_i; g, N) \pmod{N^2}. \end{aligned}$$

The secure module is given the query $(\mathbf{f}, \Psi, \psi)$, where $\mathbf{f} = \{f_1, ..., f_n\}$. It knows a threshold function $T(n)$ and has an indicator function $\Upsilon(\cdot)$, where

$$\Upsilon_\delta(x) = \begin{cases} 0 & \text{if } x \leq \delta \text{ (watermarked)} \\ 1 & \text{otherwise (not watermarked).} \end{cases}$$

The secure module decrypts $\Psi$ and $\psi$ to find $\Delta$ and $\delta$, and announces that the data is watermarked if

$$\sum_{i=1}^{n} \Upsilon_\delta(D_P(f_i; g, \lambda, N) \bmod \Delta) \leq T(n)$$

For some threshold function $T(n)$.

### 3.1.4 Modified Paillier Cryptosystem

We now present the modifications to the Paillier cryptosystem as presented by Catalano et al. [10].

**Key Generation:** Choose two primes $p$, $q$ such that $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are also prime. Let $N = pq$ and define $\lambda = \text{lcm}(p-1, q-1)$ . Choose $e \in \mathbb{Z}_N$ such that $\gcd(e, \lambda) = 1$, and $d$ such that $ed \equiv 1 \pmod{\varphi(N)}$, where $\varphi(N)$ is Euler's totient function.

**Encryption:** Message $m \in \mathbb{Z}_N^*$ and public key $(N, g, e)$ are given. Choose a random $r \in \mathbb{Z}_N^*$. Encryption is defined as

$$E_M(m, r; N, g, e) = (1 + N)^m r^e \bmod N^2.$$

**Decryption:** Ciphertext $c = E_M(m, r; N, g, e)$ and private key $d$ are given. First the ciphertext is examined modulo $N$ to find $r^e$, and an RSA decryption is performed to find $r$:

$$c \equiv r^e \bmod N,$$

$$c^d \equiv r^{ed} \equiv r \bmod N.$$

We note that

$$c \equiv (1 + N)^m r^e \equiv (1 + mN) r^e \bmod N^2,$$

and complete decryption by computing

$$m = \frac{(cr^{-e} \bmod N^2) - 1}{N}.$$

The only modular exponentiation in the modified Paillier cryptosystem is the RSA decryption. This decryption can be performed as one modular exponentiation over $p$ and another over $q$. Since exponentiation modulo $N$ has running time $O(\log^3 N)$, the original Paillier cryptosystem does approximately 32 times more work during decryption than does the modified Paillier cryptosystem.

**Homomorphisms:** We choose $e = N$, so the homomorphisms are unchanged from the unmodified Paillier cryptosystem (Section 2.2.1). This choice of $e$ does not affect the work done by the secure module at all.

### 3.1.5 Efficiency

G-SQIM sends the secure module $n + 2$ Paillier ciphertexts, for a total of $2\ell(n + 2)$ bits, while SQIM sends only $n$ Goldwasser-Micali ciphertexts, for a total of $\ell n$ bits. G-SQIM sends slightly more than twice as many bits to the secure module than SQIM.

The secure module must also perform more work in G-SQIM than in SQIM. In both G-SQIM and SQIM, the computational load of the secure module is dominated by the decryptions that it performs. G-SQIM requires $n + 2$ Paillier decryptions, whereas SQIM requires $n$ Goldwasser-Micali decryptions. By using the modified Paillier cryptosystem (see Section 3.1.4) in G-SQIM, the exponentiations in the G-SQIM decryption can be performed as two modulo exponentiations modulo $p$ and $q$, whereas SQIM requires only a single modulo exponentiation modulo $p$. Therefore, G-SQIM is approximately 2 times slower than SQIM.

### 3.1.6 Security

G-SQIM is secure against passive adversaries assuming the semantic security of the Paillier cryptosystem. The claims of zero knowledge (or minimal information) follow directly that of SQIM in Section 2.7.1. The claim of security follows that of SQIM in Section 2.7.2, but is even more straightforward since only the Paillier cryptosystem is used.

## 3.2 Verified G-SQIM (VG-SQIM)

It is possible for an active malicious watermark detector to abuse the secure module. For example in SQIM, if a watermark detector has a Goldwasser-Micali ciphertext $y = E_{\mathrm{GM}}(x, r; g, N)$ but does not know $x$, it can set the input to the secure module to be $n$ copies of $y$. If the secure module says "watermarked", then the detector knows $x = 0$, otherwise it knows that $x = 1$. In this way, the watermark detector can trick the secure module into functioning as a Goldwasser-Micali decryption oracle.

This example illustrates the dangers of active adversaries. The above attack would not be possible with an honest-but-curious watermark detector, because the query

to the secure module could not have been generated by such a detector. To prevent active attacks, we force the watermark detector to prove that each query to the secure module is valid, in that is comes from an honest execution of the SQIM algorithm. If the a query is not proved to be valid, the secure module refuses to respond.

First we present a homomorphic IND-CCA1 cryptosystem due to Boneh et. al [5]. This is the maximum level of security possible for a homomorphic cryptosystem. We then discuss how to modify the G-SQIM system, in part using the IND-CCA1 cryptosystem, so the secure module will only respond to valid queries, and will therefore be secure against active adversaries.

The set of plaintexts used in watermarking is small (compared with $N$), and there are active attacks which make use of this fact. We will next present one such attack, and will introduce simple modifications to G-SQIM which will thwart all attacks based on the small set of plaintexts.

Finally, we analyze the efficiency and security of VG-SQIM.

### 3.2.1 IND-CCA1 Paillier Cryptosystem

The Paillier and Goldwasser-Micali cryptosystems presented in Chapter 2 are IND-CPA secure, that is they achieve semantic security against a chosen plaintext attack. In order to make G-SQIM secure against active adversaries, we need a homomorphic cryptosystem that is secure against active adversaries. We use the IND-CCA1 Paillier Cryptosystem presented by Boneh et al. [5].

#### IND-CCA1 Security

IND-CCA1 security, also known as a lunchtime chosen ciphertext attack, is defined as follows. Define a cryptosystem as (KeyGen, Encrypt, Decrypt).

- **KeyGen** takes as input a security parameter $t$ and outputs a public key $K_{\text{pub}}$ and a private key $K_{\text{priv}}$.
- **Encrypt** takes as input a message $m$ and a public key $K_{\text{pub}}$ and outputs ciphertext $\alpha$ which is an encryption of $m$ with $K_{\text{pub}}$.

- **Decrypt** takes as input a ciphertext $\alpha$ and a private key $K_{\text{priv}}$. If $\alpha$ is an encryption of message $m$, then the output is $m$, otherwise the output is `INVALID`.

We say a function $f : \mathbb{Z} \to \mathbb{R}$ is negligible is for any positive $\alpha \in \mathbb{Z}$, $|F(x)| < \frac{1}{x^\alpha}$ for all sufficiently large $x$. A cryptosystem is secure against a lunchtime chosen ciphertext attack (IND-CCA1) if no adversary $\mathcal{A}$ can win the following game with non-negligible probability. The game is run by a challenger $\mathcal{C}$.

1. **Setup Stage:** $\mathcal{C}$ runs the KeyGen algorithm with security parameter $t$ and gives $K_{\text{pub}}$ to $\mathcal{A}$.
2. **Query Stage:** $\mathcal{A}$ can adaptively query $\mathcal{C}$ as a decryption oracle. $\mathcal{A}$ is also free to use $K_{\text{pub}}$ in any way it would like.
3. **Challenge Stage:** $\mathcal{A}$ sends $\mathcal{C}$ two equal messages, $m_0$ and $m_1$. $\mathcal{C}$ chooses a random $b \in \{0, 1\}$, encrypts $m_b$, and sends the resulting ciphertext to $\mathcal{A}$.
4. **Decision Stage:** $\mathcal{A}$ outputs $\hat{b}$, its guess as to the value of $b$.

Let $E(t)^b_{\mathcal{A}}$ be the result of playing the game with security parameter $t$, choice $b$ in the Challenge stage, and adversary $\mathcal{A}$, so we can write $\hat{b} = E(t)^b_{\mathcal{A}}$. The cryptosystem is IND-CCA1 secure if

$$|Pr[E(t)^0_{\mathcal{A}} = 0] - Pr[E(t)^1_{\mathcal{A}} = 0]|$$

is a negligible function of $t$ for all $\mathcal{A}$ that run in time polynomial in $t$.

Semantic security against chosen plaintext attacks, also known as IND-CPA security, is defined by the same game, except that in the Query stage, $\mathcal{A}$ can not query $\mathcal{C}$ as a decryption oracle. Thus, $\mathcal{A}$ can adaptively encrypt messages, but can not decrypt ciphertexts.

### IND-CCA1 Paillier Cryptosystem

What follows is an additively homomorphic IND-CCA1 cryptosystem presented by Boneh et al. [5]. It is an IND-CCA1 variant of a variant of the modified Paillier cryptosystem presented in Section 2.2.1, and uses the same complexity assumption.

**Key Generation:** Choose two primes $p$, $q$ such that $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are also prime. Let $N = pq$. Choose $x$ and $y$ randomly from $[1, \lfloor \frac{N^2}{2} \rfloor]$. Choose $h$ randomly from $\mathbb{Z}^*_{N^2}$. Let $g = -h^{2N} \in \mathbb{Z}^*_{N^2}$. Verify that $\gcd(g \pm 1, N) = 1$ or $N$ and that $g^2 \neq 1$. The private key is $(x, y$ and the public key is $K = (N, g, u, v)$, where $u = g^x \in \mathbb{Z}^*_{N^2}$ and, $v = g^y \in \mathbb{Z}^*_{N^2}$.

**Encryption:** Message $m \in \mathbb{Z}^*_N$ and public key $(N, g, u, v)$ are given. Choose a random $r \in [1, \lfloor \frac{N^2}{2} \rfloor]$. Encryption is defined as

$$E_C(m, r; K) = (g^r, (1+N)^m u^r, v^r) \in (\mathbb{Z}^*_{N^2})^3.$$

**Decryption:** Ciphertext $\alpha = (A, B, C)$ and private key $(x, y)$ are given. $\alpha$ is a valid ciphertext if $\left(\frac{A}{N}\right) = 1$, $A^4 \neq 1 \bmod N^2$, $A^y \equiv C \bmod N^2$, and $A^x \equiv B \bmod N$. The last condition verifies that $\frac{B}{A^x} \bmod N^2$ is of the form $(1+mN)$. Decryption is defined as

$$D_C(\alpha; K, x, y) = \begin{cases} \dfrac{(BA^{-x} \bmod N^2) - 1}{N} & \text{if } \alpha \text{ valid} \\ \texttt{INVALID} & \text{otherwise.} \end{cases}$$

For all valid ciphertexts, encrypted as above, the output will be $m$.

**Additive Homomorphism:** Given ciphertext $\alpha = (A, B, C)$ which is an encryption of message $m$ and ciphertext $\beta = (A', B', C')$ which is an encryption of message $m'$, define multiplication of ciphertexts to be

$$\alpha \cdot \beta \triangleq (AA', BB', CC'),$$

where all operations are defined over $\mathbb{Z}^*_{N^2}$. Then,

$$E_C(m_1, r_1; K) \cdot E_C(m_2, r_2; K) = E_C(m_1 + m_2, r_1 + r_2; K).$$

**Multiplicative Homomorphism (multiplication by a constant):** Given ciphertext $\alpha = (A, B, C)$ which is an encryption of message $m$, define exponentiation

of ciphertexts by a constant $k$ to be

$$\alpha^k = (A^k, B^k, C^k),$$

where all operations are defined over $\mathbb{Z}^*_{N^2}$. Then,

$$E_C(m, r; K)^k = E_C(km, kr; K).$$

### 3.2.2 Verification

For a query to the secure module to be valid, it must be have the properties of *legitimacy* and *wholeness*. A query is *legitimate* if all inputs to the secure module are the result of homomorphic operations on ciphertexts from existing watermarking public keys. This condition prevents the detector from inventing new ciphertexts to use as input to the secure module, as in the example above. A query is *whole* if a single signal and single public watermarking key were used to generate it. This prevents the watermark detector from mixing parts of multiple transform matrices and forces it to calculate all transform coefficients using the same signal.

One possible solution is to have the watermark embedder sign the watermarking public key, and have the key and the signature be sent to the secure module as a part of all queries. However, this would also require that the entire signal be sent to the secure module to double-check the watermark detector's calculations and verify validity. This would defeat the entire rational for SQIM and its derivatives.

Instead we embed verification within the query itself, by only accepting queries that have a certain mathematical form which an attacker can replicate with only negligible probability. An invalid query will receive the response `INVALID`.

#### Legitimacy

We ensure legitimacy by using the IND-CCA1 Paillier cryptosystem described in Section 3.2.1 instead of the modified Paillier cryptosystem described in Section 3.1.4. An attacker has access to the ciphertexts in watermarking public keys but does not have

a public (encryption) key. The attacker has two options for generating new ciphertexts: it can operate homomorphically on the ciphertexts available to it, or it can examine those ciphertexts and then try to generate new ciphertexts. Because of the IND-CCA1 security of the cryptosystem, the latter option has a negligible probability of producing valid ciphertexts. Therefore, all ciphertexts given to the secure module must come directly from watermarking public keys or from homomorphic operations on those ciphertexts.

### Wholeness

The notation in this section is taken from Section 3.1. Note that Section 3.2.3 explains how to deal with attacks which take advantage of a small plaintext space.

During initialization, the watermark embedder chooses a random $\theta = \{\theta_1, ..., \theta_n\} \in (\mathbb{Z}_N)^n$. It includes in the watermarking public key $\sigma = E_C(\Delta\delta\theta_1\theta_2 \cdots \theta_n), \rho_0; g, N)$ as well as $\mathbf{\Omega} = \{\Omega_1, ..., \Omega_n\}$, where $\Omega_i = E_C(\theta_i, \rho_i; g, N)$, $\rho_i \in \mathbb{Z}_N^*$ chosen at random.

In Section 3.1.2, the hidden transform was chosen as $\mathbf{S} = \{s_{ij}\}$, for $i = 1...n$ and $j = 1...m$. We now add an extra row, $\mathbf{s}_0$, to the top of the matrix $\mathbf{S}$. $\mathbf{s}_0$ is formed as a random linear combination of the all the other rows, $\mathbf{s}_i$:

$$\mathbf{s}_0 = \sum_{i=1}^{n} \theta_i \mathbf{s}_i.$$

$\mathbf{V} = \{v_{ij}\}$ is defined as before with the addition of row 0: $v_{0,j} = E_P(s_{0,j}, r_{0,j}; g, N)$, with $r_{0,j} \in \mathbb{Z}_N^*$ chosen at random. Finally, we compute

$$\alpha_0 = \sum_{i=1}^{n} \theta_i(\Delta - \alpha_i),$$

and set $k_0 = E_P(\Delta - \alpha_0, r_0; g, N)$ , with $r_0 \in \mathbb{Z}_N^*$ chosen at random.

Queries to the secure module now take the form $(\mathbf{f}, \mathbf{\Omega}, \Psi, \psi, \sigma)$.

When the secure module receives a query, it first decrypts $\sigma$, $\mathbf{\Omega}$, $\Psi$, and $\psi$, and checks if $D_P(\sigma; g, \lambda, N) = \Delta\delta\theta_1\theta_2 \cdots \theta_n \bmod N$. If not, $\theta$ or $\Delta$ or $\delta$ is corrupt and the secure module outputs `INVALID`. Finally, it checks if

$$D_C(f_0; g, \lambda, N) = \sum_{i=1}^{n} \theta_i D_C(f_i; g, \lambda, N). \tag{3.2}$$

If equation 3.2 does not hold, then either the transform or the quantization was corrupt and the secure module outputs `INVALID`. Once all wholeness conditions have been satisfied, the secure module performs as normal (see Section 3.1.3).

**Theorem 3.2.1** *The secure module will respond* `INVALID` *iff a query is invalid.*

**Proof** The watermark detector can not mix and match $\Omega_i$, $\Delta$ or $\delta$ from different public watermarking keys, because $\sigma$ would not match $\mathbf{\Omega}$, $\Psi$ or $\psi$. The homomorphic properties of the IND-CCA1 Paillier cryptosystem do not allow the multiplication of two encrypted values, so an attacker can not homomorphically manufacture ciphertexts which would be valid. An attack algorithm which can otherwise do this could be used to solve the computational Diffie-Hellman problem[1], which we assume is impossible.

In any watermark detection, all the $v_{ij}$ must come from the same $\mathbf{V}$, and that $\mathbf{V}$ is the one associated with $\mathbf{\Omega}$. Likewise, all the $k_i$ must come from the same $\mathbf{k}$, and that $\mathbf{k}$ is the one associated with $\mathbf{\Omega}$. If a watermark detector mixes and matches coefficients from different $\mathbf{V}$ or from different $\mathbf{k}$, it can not fulfill equation 3.2:

$$D_C(f_0; g, \lambda, N) = \sum_{i=1}^{n} \theta_i D_C(f_i; g, \lambda, N).$$

Encrypting both sides of equation 3.2, we see

$$f_0 = \prod_{i=1}^{n} {f_i}^{\theta_i} = \prod_{i=1}^{n} {\Omega_i}^{D_C(f_i; g, \lambda, N)}.$$

A mix-and-match attack would have to generate $f_0$, but neither of these equalities can be formed homomorphically without the knowledge of $\theta$ or $D_C(f_i; g, \lambda, N)$, neither of which are known to the detector. Again, an attack algorithm which could do this

[1]The computational Diffie-Hellman problem is: given $a = g^x \bmod N$ and $b = g^y \bmod N$, find $c = g^{xy} \bmod N$. This is assumed to be impossible to calculate in polynomial time.

could be used to solve the computational Diffie-Hellman problem, which we assume is impossible.

We showed that for a given $\mathbf{\Omega}$ there is only one possible $\mathbf{V}$ and $\mathbf{k}$ that will be accepted, and that no mixing and matching is possible. Therefore, the wholeness property holds. Recall that a valid query is one that satisfies both the wholeness and legitimacy conditions. Since the legitimacy property also holds, then by definition, the secure module will respond `INVALID` if a query is invalid. If a query is valid, the secure module will not respond `INVALID` by design. □

Note that there are a variety of attacks which will not be caught but seem as if they violate the wholeness constraint, usually attacks on columns of $\mathbf{V}$. For example, an attacker could remove column $i$ from the encrypted transform matrix $\mathbf{V}$ and remove sample $i$ from the signal, and the watermark detection query will be valid. However, the attacker could achieve the same result by using the original signal and setting coefficient $i$ to 0. More generally, any linear combination of columns of $\mathbf{V}$ (homomorphically generated) can be simulated by careful choice of signal. A proposed attack that can be simulated by the choice of the signal is not considered to be a valid attack, because these "attacks" do not reveal any information which could not easily be discovered with a valid query.

### 3.2.3 Attacks on Small Plaintext Space

First we present an attack on a small plaintext space. Let $\mathbf{S}$ and $\mathbf{V}$ be as defined in Section 3.1.2. Assume, for this example, that all $s_{ij}$ come from a set of $k$ possibilities. For $x = 1...n$ and $y = 1...m$, the attacker forms $\mathbf{V}'$ such $v'_{xy} = v_{1,1}$ and for all $(i, j) \neq (x, y)$, $v'_{ij} = v_{ij}$. $\mathcal{A}$ performs a standard VG-SQIM watermark detection for each $(x, y)$. If the secure module responds `INVALID`, then the attacker knows that the query broke the wholeness constraints, and that $s_{xy} \neq s_{1,1}$. Conversely, if the secure module responds "watermarked" or "not watermarked" then the attacker knows that $s_{xy} = s_{1,1}$. After $nm$ iterations, the attacker will know which elements of $\mathbf{S}$ are equal to $s_{1,1}$.

While the example attack found the elements equal to $s_{1,1}$, it is obvious how to expand this to all elements of $\mathbf{S}$. Only $\min((mn)^2, (k-1)mn)$ queries of the secure module are necessary to do so.

To prevent this attack, and others which take advantage of the small plaintext space, we randomize the plaintexts so as to increase the set of possible plaintexts. Instead of using the transform matrix $\mathbf{S}$, we will use $\mathbf{S}'$, where

$$s'_{ij} = s_{ij} + r_{ij}p,$$

where $r_{ij} \in \mathbb{Z}_q$ is chosen randomly and $p$ is a factor of the modulus $N$. The encrypted matrix $\mathbf{V}$ in the watermarking public key is now formed from $\mathbf{S}'$ instead of $\mathbf{S}$.

The wholeness constraints are checked as normal. If a query is verified as valid, the plaintexts are taken modulo $p$, and the resulting values will be equal to what they would have been if $\mathbf{S}$ had been used.

This is also done with the secret quantization values. Define

$$\alpha'_i = \alpha_i + r_i p,$$

where $r_i \in \mathbb{Z}_q$ is chosen randomly. The encrypted vector $\mathbf{k}$ in the watermarking public key is now formed from $\mathbf{alpha}'$ instead of $\alpha$.

Finally, this is done with $\Delta$ and $\delta$ as well:

$$\Delta' = \Delta + r_a p,$$

$$\delta' = \delta + r_b p,$$

where $r_a, r_b \in \mathbb{Z}_q$ are chosen randomly. $\Psi$ is now formed from $\Delta'$ and $\psi$ is now formed from $\delta'$.

The method presented in this increase the number of possible plaintexts by a factor of $q$, rendering VG-SQIM safe from the specific attack presented and attacks on the small plaintext space in general. It contributes minimal computational load

and no communications overhead for the secure module. The only requirement is that all possible transform coefficients be smaller than $p$, which we will now show is a reasonable assumption.

Consider the following (large) signal. Assume $2^{30}$ samples, each with 24 bits of precision, and transform coefficients with 48 bits of precision. Then the maximum possible transform coefficient is $2^{30+24+48} = 2^{102}$. In the Paillier cryptosystem, if $|N| = 1024$ bits then $|p| = 512$ bits. Even in the case of such a large signal, the assumption easily holds.

### 3.2.4 Efficiency

Recall that signals are of length $m$ and there are $n$ transform coefficients. Samples of the signal are k-bit numbers and encryptions modulo $N$ have $\ell = \log_2 N$ bits.

With the IND-CCA1 Paillier cryptosystem, each encryption contains 3 $2\ell$-bit components, $6\ell$ bits total. A VG-SQIM query contains $2n + 5$ encryptions, for a total of $6\ell(2n+5)$ bits. In comparison, G-SQIM sends $2\ell(n+2)$ bits and SQIM sends $\ell n$ bits. Standard QIM sends approximately $mnk + mk + n$ bits. VG-SQIM requires 6 times more bandwidth than G-SQIM and 12 times more bandwidth than SQIM. However, in the example given in Section 2.6, VG-SQIM still requires 200 times less bandwidth than standard QIM.

Assuming that modular exponentiations dominate the running time, the wholeness constructions do not significantly affect the running time. The IND-CCA1 Paillier cryptosystem required by the legitimacy constraint, however, requires 3 modular exponentiations for every decryption, instead of the 1 modular exponentiation per decryption required for the standard Paillier cryptosystem. VG-SQIM will therefore have running time approximately 3 times longer than that of G-SQIM.

### 3.2.5 Security of VG-SQIM

VG-SQIM is secure against active adversaries assuming the IND-CCA1 security of the cryptosystem presented in Section 3.2.1.

The security of VG-SQIM is based on the security of G-SQIM. Because of the

validation of VG-SQIM, the secure module will only respond to honest-but-curious watermark detectors. Therefore, the proofs of security and zero knowledge (or minimal information) for G-SQIM naturally extend to VG-SQIM against active watermark detectors.

## 3.3 Verified Secure Spread Spectrum (V-SSS)

Kalker [22] presented a secure Spread Spectrum (SSS) system which utilizes a secure module in a semi-public key watermarking scheme. We will briefly summarize the Spread Spectrum watermarking scheme and introduce SSS, which is secure versus passive adversaries. We will present Verified SSS (V-SSS), a modified version of SSS which is secure against active adversaries.

### 3.3.1 Spread Spectrum Watermarking

The watermark secret is a vector $\mathbf{w} = \{w_1, ..., w_n\}$ where each $w_i = \pm\Delta$. $\mathbf{T}$ is a public transform.

**Watermark Embedding:** To embed a watermark into a signal $\mathbf{x} = \{x_1, ..., x_m\}$, the signal is first transformed:

$$\mathbf{Tx} = \mathbf{c} = \{c_1, ..., c_n\}.$$

The watermark is added to the transform coefficients, and the inverse transform is performed on the result to generate the watermarked image $\hat{\mathbf{x}}$:

$$\begin{aligned} \hat{\mathbf{x}} = & \ \mathbf{T}^{-1}(\mathbf{c} + \mathbf{w}) \\ = & \ \mathbf{T}^{-1}(\mathbf{Tx} + \mathbf{w}) \ . \\ = & \ \mathbf{x} + \mathbf{T}^{-1}\mathbf{w} \end{aligned}$$

**Watermark Detection:** The watermark detector is given $\mathbf{w}$, $\mathbf{T}$, a threshold $\tau$, and an input signal $\mathbf{y}$. It correlates $\mathbf{w}$ with $\mathbf{Ty}$, and if the correlation is greater than $\tau$ announces that $\mathbf{y}$ is watermarked. That is, $\mathbf{y}$ is considered watermarked if

$$\frac{\mathbf{Ty} \cdot \mathbf{w}}{\sqrt{\mathbf{Ty} \cdot \mathbf{Ty}}} \geq \tau.$$

### 3.3.2 Secure Spread Spectrum (SSS)

Only watermark detection is changed in Secure Spread Spectrum; watermark embedding is unaltered.

Let $\omega = \{\omega_1, ..., \omega_n\}$, where $\omega_i = E_P(w_i, r_i; g, N)$ be an encryption of the watermarking secret which is stored in the watermarking public key. To perform SSS watermark detection on a signal $\mathbf{y}$, the watermark detector first transforms the signal to find $\mathbf{Tx} = \mathbf{c}$. The Hidden Transform methods of Section 2.3 are then used to find the encrypted inner product of $\mathbf{c}$ and $\mathbf{w}$, using $\omega$:

$$A = E_P(\mathbf{c} \cdot \mathbf{w}, r; g, N).$$

The watermark detector then computes

$$B = \tau\sqrt{\mathbf{Ty} \cdot \mathbf{Ty}}$$

in the clear. It sends the query $(A, B)$ to the secure module, which decrypts $A$ and announces that $\mathbf{y}$ is watermarked if

$$D_P(A; g, \lambda, N) \geq B. \tag{3.3}$$

### 3.3.3 Verified Secure Spread Spectrum (V-SSS)

The security of SSS versus honest-but-curious watermark detectors follows immediately from the security of the Paillier cryptosystem. Notice, however, that the secure module has a similar vulnerability to that in SQIM. Suppose an adversary has a Paillier ciphertext $\alpha = E_P(m, r'; g, N)$ for which $m$ is unknown. If the adversary submits $(\alpha, \gamma)$ to the secure module, it will reveal if $m \geq \gamma$. Furthermore, by performing this

test multiple time with intelligently chosen $\gamma$, the adversary can discover the value of $m$ in $\log_2 N$ queries.

Again, the solution to active adversaries is to have the secure module reject queries which are not valid. As with VG-SQIM, attacks which can be simulated by careful choice of signal are not considered to be valid attacks. To enforce legitimacy, it is again sufficient to change from the semantically secure Paillier cryptosystem (Section 2.2.1) to the IND-CCA1 Paillier cryptosystem (Section 3.2.1) and enforce the wholeness constraints.

As before (Section 3.2.3), we can eliminate attacks on the small plaintext space by replacing all $w_i$ with $w_i' = w_i + r_i p$ where $r_i \in \mathbb{Z}_q$ is chosen randomly, and having the secure module take the decryption in Equation 3.3 modulp $p$ . In the case when $w_i = \pm\Delta$, this is especially important.

### Wholeness

The wholeness construction, as in VG-SQIM, is more involved. First, a random $k \in \mathbb{Z}_N$ is chosen, and included in the watermarking private key as $\kappa = E_P(k, r_k; g, N)$. Let $\omega' = \{\omega_1', ..., \omega_n'\}$, where $\omega_i' = \omega_i^k$, so $\omega_i' = E_P(kw_i, r_i; g, N)$. Finally, define $A' = E_P(k(\mathbf{c} \cdot \mathbf{w}), r; g, N)$, which can easily be computed with $\omega'$ in the same manner as as $A$ is computed with $\omega$.

Queries to the secure module are of the form $(A, A', B, \kappa)$. When the secure module receives a query, it first decrypts $\kappa$ to find $k$, and verifies that

$$kD_P(A; g, \lambda, N) = D_P(A'; g, \lambda, N). \tag{3.4}$$

If this condition does not hold, then the secure module returns `INVALID`. Otherwise it continues with watermark detection as in Section 3.3.2.

For V-SSS, violating the wholeness constraint means combining different watermarking secrets. For example, $\{\omega_1, ..., \omega_{\frac{n}{2}}\}$ could be drawn from one watermarking public key, while $\{\omega_{\frac{n}{2}+1}, ..., \omega_n\}$ could be drawn from another. There are only two conditions where this will not result in a response of `INVALID`.

The first condition is if the attacker could either encrypt its own $\kappa$ or guess the

decryption of an existing $\kappa$. Then, knowing $k$, it could calculate it's own $\omega' = \omega^k$. Note that this calculation is not possible without knowledge of $k$, because homomorphic multiplication involves multiplication by a known constant, and both $\omega$ and $\kappa$ are encrypted. Since the IND-CCA1 Paillier public key is not known, the attacker can't encrypt it's own $\kappa$, and the probability of it guessing at the decryption of an IND-CCA1 Paillier ciphertext is negligible, so this attack is not possible

Second, the attacker could fool the secure module by getting very lucky by choosing a signal and mixing watermarking secrets so that Equation 3.4 holds. Again, since the attacker doesn't know the value of $k$ or the watermarking secrets, this happens with negligible probability.

Therefore, it is not possible to violate the wholeness property without the secure module responding `INVALID`.

#### Security

With the legitimacy and wholeness constraints satisfied, the secure module will only respond to valid queries. Because the SSS system is secure against all adversaries who only submit valid queries (honest-but-curious adversaries), the V-SSS system is secure against active adversaries. Furthermore, it reveals as little the minimum information possible with each watermark detection, which is the single bit revealing whether or not a signal is watermarked.

## 3.4 Conclusion

In Chapter 3 we presented two extensions to SQIM, and one to SSS. For SQIM, we first show in G-SQIM how to use arbitrary quantizer step sizes, as opposed to the step size of 2 which is required by SQIM. Next we showed how to make G-SQIM, which is secure only against honest-but-curious watermark detectors, into VG-SQIM, which is secure versus any watermark detector. We also introduced a method to avoid attacks that take advantage of small sets of possible plaintexts. Finally, we showed how to modify SSS, which is secure only against honest-but-curious watermark detectors, into V-SSS, which is secure against any adversary.

# Chapter 4

# Image Identification with Randlets

The randlet transform is a new transform composed of randomly-chosen basis functions. This randomization gives the randlet transform distinct advantages which are important from signal processing and security points of view.

There are two types of randomization, explicit and implicit. With implicit randomization, input signals are assumed to be stochastic, and probabilistic claims are made about the output based on these assumptions. Signal processing applications generally use implicit randomization. Explicit randomization, like that used in the randlet transform, is randomization that is inherent to an algorithm and is independent of the input. Explicit randomization allows algorithms to avoid worst case performance and eliminates the need to make statistical assumptions about the input.

Because randlet transform basis functions are chosen at random from a large set of basis functions, the worst case performance of the transform (say, missing an important image feature) occurs with extremely low probability. This is similar to the idea of universal hash functions (See [12]). A universal hash function is chosen at random from a large family of hash functions. They are constructed so that for any two distinct inputs the probability of a hash collision, when taken over the choice of hash functions, is close to optimal, even without making any stochastic assumptions about the inputs. The main idea behind universal hash functions is that one may dispense with statistical assumptions about the input and for most choices of keys expect good performance. This is crucial since in the case of an intelligent, malicious

attacker it is not always possible to make statistical assumptions about the input.

The randomness of the randlet transform also means that without knowledge of the key, the transform coefficients are independent. It also allows analysis of the entropy of transform coefficients without making assumptions about the statistical nature of the input. Furthermore, there are benefits in security applications such as hashing and watermarking because an attacker does not know which basis functions are used in the transform, making attacks much more difficult.

It is natural to wonder why a new transform is necessary to introduce randomness. For example, a DCT can be multiplied by a random matrix to randomize the output. For many applications, though, randomization is not sufficient. In the case of image identification, the transform also must be *perceptual* to achieve good performance. A perceptual transform is one where two images that are perceptually similar will have similar transforms. The randlet transform is both randomized and perceptual, whereas the example of the randomized DCT is randomized but not perceptual.

Furthermore, a deterministic transform which is susceptible to an attack will still be susceptible to the same attack when randomized, because an attacker will still know a great deal about the portions of an image which are important. For example, if a certain attack works well in some sense versus the low-low band of a wavelet transform, then it will also work well against a version of that transform where the coefficients are subsequently randomized. An attacker would know, at a minimum, that the transform is based on the low-low band of the wavelet transform, and not at all on the other 3 bands, even if it could not predict the exact transform coefficients as a result of the randomization.

Section 4.1 introduces the randlet transform. It is a family $\mathcal{F} = \{f_K\}$ of transforms indexed by a key $K$. Each transform uses a set of basis functions that are chosen pseudo-randomly based on the key. The key specifies which member of the family will be used for a particular instance of the transform.

Basis functions in the randlet transform are called *randlets*. Randlets are generally formed by taking a small number of localized functions known as *mother randlets*, and scaling, translating, rotating, and otherwise modifying them. The result is a large number of possible randlets. Each randlet transform uses a small subset of

all possible randlets as its basis functions. While they offer some multi-resolution properties, they do not have the fine structure (e.g., the nested vector spaces) of standard multi-resolution analysis.

Section 4.2 discusses the performance of the randlet transform in image hashing and image identification. We write that an image identification system $H(\cdot)$ takes as input an image $I$ and outputs a vector $v$ which represents the image: $v = H(I)$. Let $\mathcal{A}$ be a family of attacks against which $H$ is meant to operate, and let $A \in \mathcal{A}$ be a specific attack. $H$ has the property that when $I$ is perturbed by an attack, $v$ is not changed much, so $\|H(I) - H(A(I))\| \leq \epsilon$ with high probability. Also, if two distinct images are compared, then their vectors should be distinct as well, so $\|H(I) - H(I')\| > \epsilon$ with high probability. We define the following two terms:

$$\text{Probability of false positive: } P_\epsilon^+ = \Pr[\,\|H(I) - H(I')\| \leq \epsilon]$$

$$\text{Probability of false negative: } P_\epsilon^- = \Pr[\,\|H(I) - H(A(I))\| > \epsilon]$$

The goal of an image identification system is to achieve low $P_\epsilon^+$ and $P_\epsilon^-$, but lowering one will generally cause the other to rise. As epsilon is varied, $P_\epsilon^+$ and $P_\epsilon^-$ describe an ROC curve[1]. The ROC curve itself is defined by the attack and the transform. An application will choose a point along the ROC curve according to its own particular needs. In general, it is more important to have a low $P_\epsilon^+$ because in most systems there are many more distinct images than there are attacked images. Additionally, an adversary who is attacking images may be deterred by a detection probability of even 50%.

Systems already exist that manipulate transform coefficients to perform image identification and image hashing (see section 4.0.1), but one of the motivations for the randlet transform was to generate a transform that was itself resistant to attacks. Additionally, the randlet transform can be used as a component in many of these other systems. This paper is concerned with the performance of transforms in identifying

[1]Generally, an ROC curve is defined as $P_\epsilon^+$ versus $1 - P_\epsilon^-$. However, for our purposes it is more useful to have ROC curves of $P_\epsilon^+$ versus $P_\epsilon^-$.

images, and so compares the randlet transform to the wavelet transform and not to these other, higher-level systems. We note, however, that the randlet transform taken alone already achieves excellent performance in relation to other systems.

In this paper our transforms are discrete and the focus is on algorithmic details and experimental results. We consider attacks based on rotation, cropping, scaling, additive noise, and JPEG compression. The randlet transform produces ROC curves superior to those of the DWT against most attacks, and especially against rotation attacks.

Figure 4.1 shows a model of an image identification system. An attacker attacks an image, which is then given to the system. The image is transformed and the transform coefficients are compared to a database of transform coefficients. The output is the likelihood that the image is an attacked version of one of the images in the library.

### 4.0.1 Previous Work

The Matching Pursuits [31] transform uses an overcomplete basis to transform images, searching at each position to find a basis function with sufficiently large power, and using residuals to compensate for the overlapping nature of the basis functions. This is similar to randlets, except that the transform is not explicitly randomized. Ventura et al. [45] use matching pursuits with two-dimensional Gaussian and Mexican hat basis functions to compress images. Curvelets [9] are an overcomplete transform with a very complicated structure.

Mihcak and Venkatesan [34] and Venkatesan et al. [44] use randomized methods for image hashing and identification, but they achieve their results by manipulating transform coefficients, as opposed to randlets, where the randomization in inherent to the transform itself. Fridrich uses randomness in image hashing [16, 17], but the resulting transforms are not capable of inversion, and are susceptible to certain attacks, like rotation. Furthermore, these transforms are not localized and so can not take advantage of localized features of documents. A common tactic is to attempt to identify images by extracting image features such as edges and corners. See, for example, Johnson et al. [21].

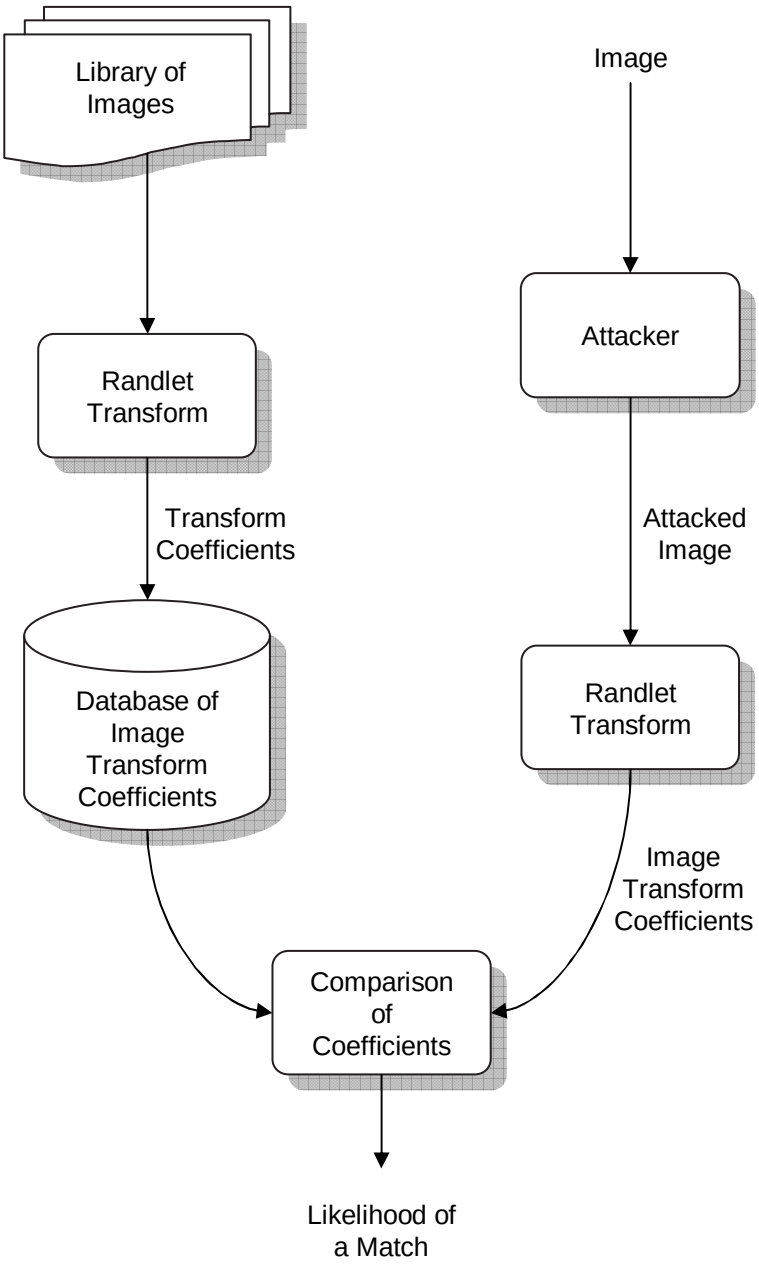


Figure 4.1: Model of an image identification system.

## 4.1 The Randlet Transform

The main idea behind the randlet transform is to randomly generate a set of basis functions and project them onto an image. Without knowing the basis functions it is hard for an adversary to predict what the projections will be.

The randlet transform can be inverted if a sufficient number of coefficients are used. For identification or hashing purposes, 100 3-bit coefficients are generally sufficient to survive the attacks we tested. If reconstruction will be performed, or if it is important that the projections are orthogonal to each other, each projection is subtracted from the image before the next projection is made. Successive projections provide a closer approximation to the original image. The reconstruction is then simply the sum of each randlet multiplied by its transform coefficient. In the case of image hashing, no reconstruction will be performed, so the transform is simply the projection of each basis function onto the image.

### 4.1.1 Randlets

Any two-dimensional function with compact or effectively-compact support can be a randlet, and any collection of such functions can be considered a randlet basis. In practice, however, the basis must be close enough to complete to capture the important features of all signals to be transformed. Furthermore, the addition of some amount of structure greatly increases the speed and lowers the memory cost of the transform and inverse transform, while experimentally imposing no cost. All of the randomness used to form a randlet basis is generated pseudo-randomly with the secret key.

For image watermarking applications, randlets are discrete, two-dimensional functions. First, a set of *mother randlets* is chosen. These are the templates from which other randlet are chosen. The mother randlets are randomly chosen, then randomly scaled, rotated, and otherwise modified to form *base randlets*, which are then normalized. The base randlets are randomly translated to form the randlets, sometimes called *final randlets*, which may overlap. All of the random decisions (choice of mother randlets, rotations, translations, etc.) are made pseudo-randomly using a secret key

and according to user-defined distributions. Note that a randlet basis will not necessarily span the set of all images.

The most important randlets in applications involving images are *perceptual* randlets, which are randlets that have few high-frequency components. In watermarking applications, these randlets result in less perceptual distortion, and are more robust to desynchronization attacks, especially geometric attacks like rotation and cropping, because the they are smoothly varying.

Some example randlets are:

**Gaussian Randlet:** Translations and scalings of: $e^{x^2+y^2}$.

**Mexican Hat Randlet:** Translations, scalings, and rotations of: $y^2 e^{\sigma_x x^2+\sigma_y y^2}$.

**Random Randlet:** Translations of elliptical regions where each pixel value is chosen from a normal distribution with standard deviation 1, and the result is normalized. This randlet is not rotated, but many versions with different random pixels are generated.

**Smooth Random Randlet:** A random randlet that has been low-pass filtered.

**Curvelet Randlet:** Curvelets [9] may be used as randlets.

**Wavelet Randlet:** Translations, scalings, and rotations of: $w(y)e^{\sigma_x x^2}$, where $w(y)$ is a wavelet.

Figure 4.2 shows some pictures of some randlets.

Because randlets have compact support (or effectively compact support), they can emphasize large-scale and small-scale aspects of images, depending on the randlet scaling. Note that some of the functions above have infinite support, but all are approximately zero at enough of a distance from the origin. Finally, randlets are normalized so that the inner product of a randlet with itself is 1.

Randlets of different types can be mixed together in the same basis. A basis with a single type of randlet is called a *pure* randlet basis, while a basis with multiple types of randlets is called a *mixed* randlet basis. The type of randlet or randlets used

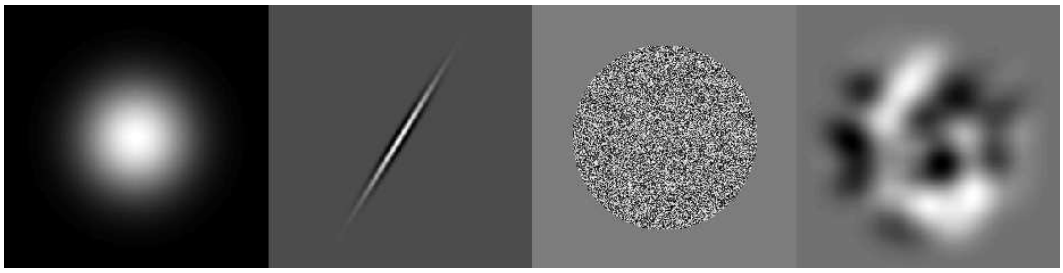

Figure 4.2: Examples of randlets. All images are normalized for greater visibility. From left to right: a $200 \times 200$-pixel Gaussian randlet, a $200 \times 14.14$-pixel Mexican Hat randlet, a $200 \times 200$-pixel random randlet, and a $200 \times 200$-pixel smooth random randlet.

depends on the task. For image identification, Gaussian randlets perform the best. For edge detection, however, randlets with oscillations, such as Mexican Hat randlets or wavelet randlets, are better. For watermarking, random randlets perform well (See Chapter 5).

### 4.1.2 Generating a Randlet Transform

It would be a simple matter to generate each randlet at random according to some distribution. In practice, however, this is prohibitively slow if done in real time, and would use excessive memory resources if performed as preprocessing. To avoid these problems the randlet basis is constrained. First, we choose a small number of mother randlets, which will determine the general properties of the randlet basis that we are generating. Next, instead of allowing, for example, arbitrary rotations and scalings, a finite number of scalings and rotations are chosen for each mother randlet. These are used to form the base randlets, copies of which are then translated across an image. This is fast, since each base randlet is only generated a single time, and requires less memory since only a single base randlet has to be stored, along with a list of translation values.

We now present an example of a randlet basis where base randlets are scaled and rotated versions of a single mother randlet. The randlet basis is represented in an

algorithm by a library of base randlets, and a list of translations for the randlets from the library. Let $m(x, y)$ be the mother randlet. We generate a base randlet by scaling by $\alpha$ horizontally and $\beta$ vertically, and then rotating by $\theta$ degrees, to produce the base randlet $b[x, y]$.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$b[x, y] = m(x', y')$$

Now we will form the list of translations. Assume that the transform will be applied to images of height $h$ pixels and width $w$ pixels[2]. Let $n$ be the number of randlets in the randlet basis, and $F_K(\cdot)$ be an algorithm which selects a randlet from the library based on the secret key $K$. The list of randlets is generated by randomly sampling from the library using $F(\cdot)$, and translating those base randlets uniformly at random $[0, h-1]$ vertically and $[0, w-1]$ horizontally. If a randlet partially falls off the edge of the image, it is cropped to the edge of the image and renormalized during the transform and inverse transform. Generally, the list of randlets is sorted so that those with the largest support come first in the list.

### Example

As an example, consider generating a basis for $256 \times 256$ pixel images. Our example basis will be composed entirely of Mexican Hat randlets with width equal to the square root of the height. 3 heights are randomly chosen, 95, 48, and 22, and for each height a number of rotations equal to half the height is randomly chosen. Therefore, the base randlets are: $95 \times 10$ Mexican Hat randlets with 48 different rotations, $48 \times 7$ Mexican Hat randlets with 24 different rotations, and $22 \times 5$ Mexican Hat randlets with 11 different rotations. The we choose the distribution for the final randlets to select size 95 randlets with probability 0.2, size 48 randlets with probability 0.3, and

[2]Section 4.1.3 describes how to apply the transform on images of different sizes.

size 22 randlets with probability 0.5, and after the size is chosen each rotation is chosen uniformly at random. After the size and rotation are chosen, the randlet is randomly translated across the image.

### 4.1.3 Transform and Inverse Transform

This section describes the randlet transform and the inverse randlet transform as performed on an image of size $h \times w$. Section 4.1.3 describes how to apply the transform on images of different sizes.

The transform is done by projecting each randlet onto the image in turn. After each projection is taken, it is subtracted from the image. What remains of the image after the subtraction is called the *residual*. The transform continues in the manner, projecting each randlet onto the residual of the previous randlet, for all $n$ randlets. When applying the randlet transform, it is important that the randlets are sorted approximately by the size of their support. The largest randlets should be projected and subtracted first, followed by the smaller randlets. Otherwise, the subtraction of the large randlets overwhelms the residual of the smaller randlets that were previously subtracted.

Transform coefficient $c_j$ is computed by finding the inner product between randlet $j$ and the residual of randlet $j-1$. If the residual of randlet $b_j$ is denoted by $r_j$, then

$$c_j = \sum_{y=1}^{h} \sum_{x=1}^{w} r_{j-1}[x,y] \cdot b_j[x,y] \tag{4.1}$$

In practice, the coefficients are quantized. The quantized value is stored as the coefficient in the transform, and it is the quantized value that is used to compute the residual. If $Q(\cdot)$ is the quantizer, then

$$r_j[x,y] = r_{j-1}[x,y] - Q(c_j) \cdot b_j[x,y] \tag{4.2}$$

The inverse transform is extremely simple. Because residuals are used when taking the transform, the projection of each randlet is orthogonal to the previous randlets. Therefore, the original image can be reconstructed by simply adding together each

randlet multiplied by its corresponding transform coefficient. If $\widetilde{I}[i,j]$ is the reconstructed image, then

$$\widetilde{I}[x,y] = \sum_{j=1}^{n} c_j \cdot b_j[x,y] \tag{4.3}$$

For both the transform and the inverse transform, the running time a basis of $n$ randlets, each with area $A$, is $O(nA)$.

In both the image identification and watermarking contexts, a variant randlet transform, called the *forward* randlet transform, is used:

$$c_j = \mathbf{A} \cdot \mathbf{r}_j, \qquad j \in \{1, ..., n\}.$$

No residuals are involved in the forward randlet transform. In the context of image identification, residuals are not necessary because the transform coefficients are used only to find information about an image, never to perform an inverse transform and reconstruct the original image. In the context of image watermarking, the use of residuals would cause the transform coefficients to be unstable, since a small perturbation in a part of the image covered by the first randlets could cause changes in the coefficients of all later randlets. For watermarking, other methods are used to compensate for the non-orthogonality of the randlets.

### Practical Considerations

In application, the randlets are not projected across the entire image. The effective footprint of the randlets is computed in the preprocessing phase by considering only the area where the randlets have non-negligible value. The randlets are then "windowed" to this area.

Since the center of each randlet ranges uniformly across the whole image, the extremities of the randlets can reach outside of the image, which results in edge-effects. The edge effects can be eliminated by padding the image with a mirror-image of itself at each edge, and increasing $h$ and $w$ to accommodate the padding. Experimentally, a padding of 5 to 10 pixels per side is sufficient.

After a number of projections have been taken, the residual often contains small details that are unlikely to be found by randomly-placed randlets. Large randlets will overlap with these details, but will not be able to detect them because of the difference in scale. Smaller randlets could detect them, but will overlap them with very low probability. We have found that an effective way to solve this dilemma is to perturb each randlet and store the projection with the largest power, along with the location of the perturbation. These perturbations can include small translations and rotations, increases and decreases in frequency, etc. Perturbing randlets has more of an effect on smaller randlets and randlets with high frequency components, while having less of an effect on larger and low-frequency randlets.

Such perturbations introduce side-information that increases the size of the transform data, but can dramatically speed the transform's convergence. For a transform of a given size, the version formed by testing small perturbations will generally have a smaller distortion. On the other hand, perturbing randlets greatly slows the transform. For example, If the randlets are perturbed $\delta$ pixels in both the horizontal and vertical directions, running time increases by a factor of $O(\delta^2)$.

Figure 4.3 shows, for a sample image, the distortion between the inverse transform and the original image as a function of the number of randlets projected. The results are typical for the case when the randlets are perturbed. Preliminary, unoptimized tests have shown that when quantized and compressed, transform size is within a factor of two to four of JPEG compression.

With the Randlet Transform, much like with Matching Pursuits [31], lossy compression is easy to do to an arbitrary level by throwing out coefficients corresponding to the randlets with the least power. In terms of the list of randlets in section 4.1.2, lossy compression can generally be achieved by using only the randlets near the beginning of the list, ignoring the randlets at the end of the list.

The rate of lossy compression can be varied dynamically as well, so the amount of bandwidth spent on an image can be made equal to the bandwidth available. An image can be sent iteratively so that the basic representation appears first, and the details are filled in later.

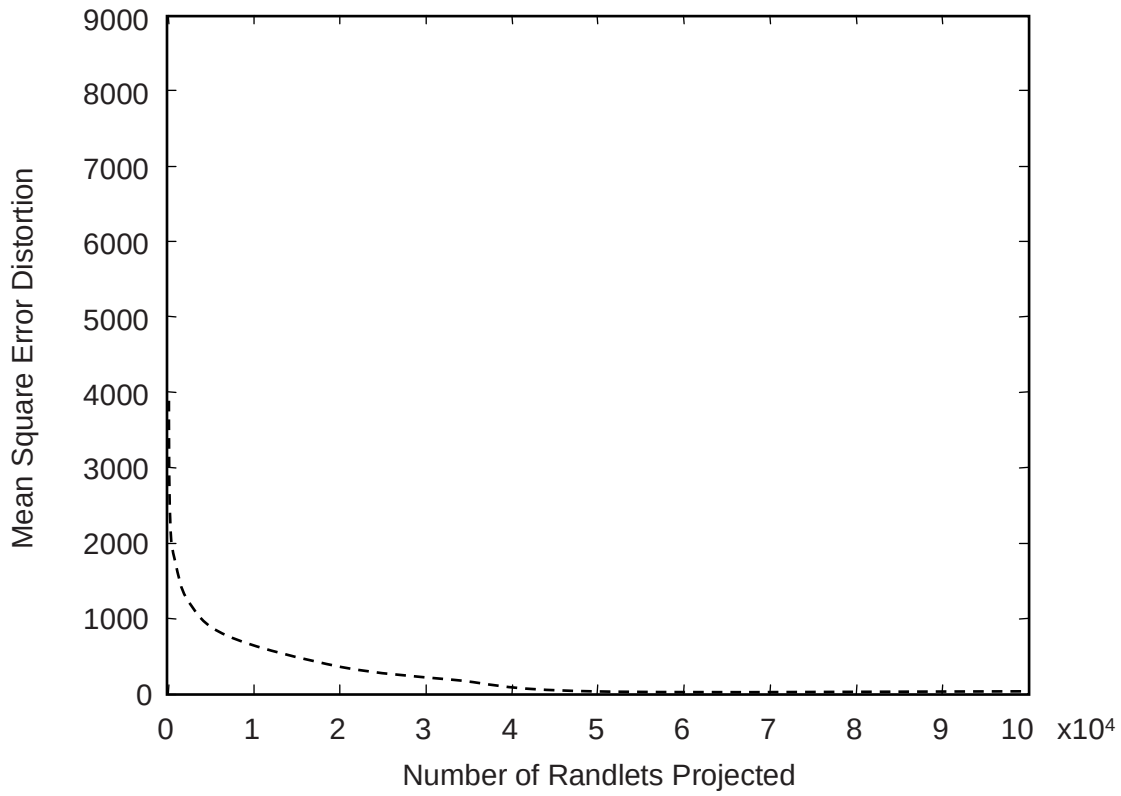


Figure 4.3: Convergence of the randlet transform for a sample image.

### Extending the Transform

A specific instance of the randlet transform is generated to work with a specific image size. To extend its usefulness, the randlet transform should be applicable to images of any size. There are two ways to achieve this.

First, the randlets can be chosen with centers in the range $[0, 1]$, and the transform can be expanded to the actual size of whatever image is chosen. In this method, the randlets themselves must also be scaled to match the size of the image. The advantage of this method is that many coefficients of the transform will become scaling-invariant. However, in order to ensure that this transform will scale well for large images, the randlet basis must be enormous. Many randlets will be redundant on small images, but are necessary to enable the transform to be taken to large scales.

The second method is to generate the transform for a relatively small image, say $128 \times 128$ or $256 \times 256$. Images larger than this size are decomposed into blocks and the transform is performed separately on all blocks. This is the preferred method, because it avoids the redundant basis functions of the previous two methods. However, the first methods can be useful for applications of the randlet transform such as hashing and watermarking, where a small number of basis functions are used, and the basis used is not expected to be complete

In order to reduce edge effects between blocks, each block can be is padded as in section 4.1.3, or the "padding" can be the first 5 or 10 pixels from the adjacent blocks. Additionally, most images will not come in integer multiples of the block size. To deal with this, the image itself is padded to achieve a size that is a multiple of the block size. This padding is removed as part of the inverse transform.

## 4.2 Image Identification and Hashing

An image identification system $H(\cdot)$ takes as input an image $I$ and outputs a vector $v = H(I)$ which represents the image. Distances between vectors are computed as the Euclidian distance. $v$ is computed by converting the image to a grayscale, scaling it to $256 \times 256$ pixels, and then computing the transform of the resulting image. We

tested many randlet transform families and all of the wavelet transforms available in Matlab.

In an image identification system it is important that the identification vectors be kept secret, because an attacker with enough pairs $(I_i, v_i)$ will be able to determine the basis functions that are being used, making it much easier to mount attacks.

For a randlet-based system, each image is transformed with a randlet transform, and $v$ is assigned to be the transform coefficients. Residuals are not taken, so each randlet is projected directly onto the original image and randlets do not need to be sorted by size. If residuals were used, a change in an image that affected one coefficient would affect later coefficients not only through the change in the image, but also in the change in the residual. Furthermore, the power of later coefficients becomes small when residuals are taken, making them less useful in identifying images. By not using residuals, each transform coefficient is made to depend only on the image, not on the ordering of randlets in the randlet basis. Since inversion is not necessary in image hashing is it possible to use relatively few basis functions to generate each image hash. 100 coefficients is sufficient for most purposes (see section 4.2.1).

Figures 4.4, 4.5, 4.6, 4.7, 4.8, and 4.9 show the distribution of coefficient values for two different randlet transforms, a Gaussian randlet transform with effective support of $200 \times 100$, and a Mexican hat randlet transform with effective support of $200 \times 200$. See Section 4.1.1 for descriptions of these randlets.

Figures 4.4 and 4.7 show histograms of the values of 500 coefficients from 500 transformed images. Figure 4.4 used a randlet basis of $200 \times 100$ Gaussian randlets, while Figure 4.7 used a randlet basis of $200 \times 200$ Mexican Hat randlets. Since Gaussian randlets are strictly positive, and since residuals are not being used, all of the coefficients from the first transform are positive. As the histograms show, the transform using Gaussian randlets produces very few coefficients with low power, while the transform using Mexican Hat randlets produces many coefficients with low power.

Figures 4.5 and 4.8 show overlayed histograms from three separate images. Figure 4.5 uses the same transform as Figure 4.4, and Figure 4.8 uses the same transform as Figure 4.7. For both transforms, the histograms of each image are quite distinct,

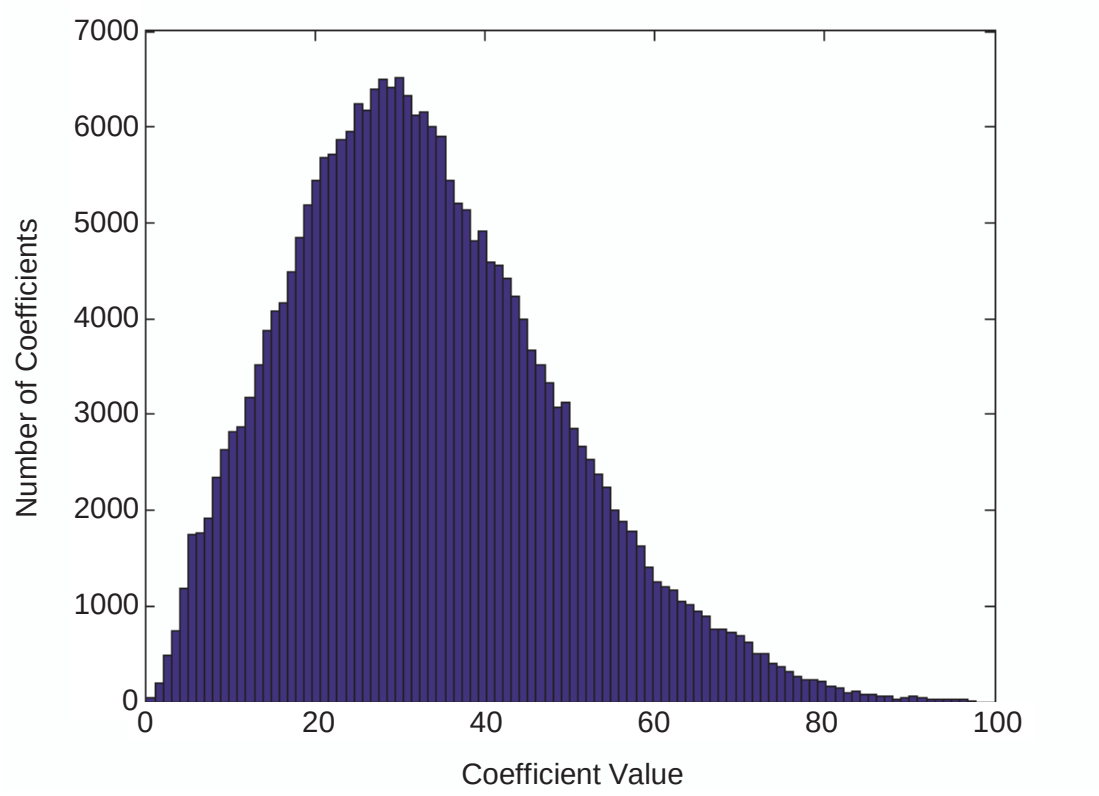


Figure 4.4: A histogram of coefficient values. 500 images were transformed with a randlet basis made up of 500 $200 \times 100$ Gaussian randlets, for a total of 250,000 coefficients in the histogram. The x-axis is coefficient values, the y-axis is the number of coefficients with each value.

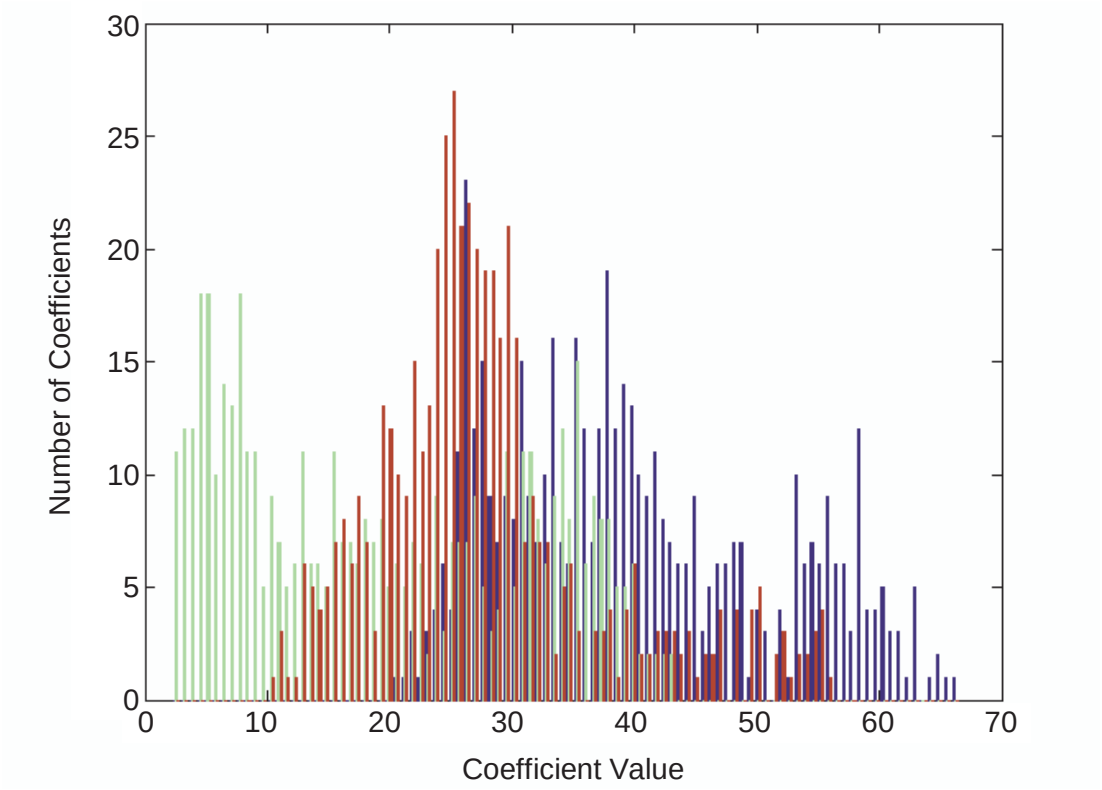


Figure 4.5: Three overlayed histograms, each of the coefficient values from a different image. 3 images were transformed with a randlet basis made up of 500 $200 \times 100$ Gaussian randlets, for a total of 1500 coefficients in the histogram. The x-axis is coefficient values, the y-axis is the number of coefficients with each value.

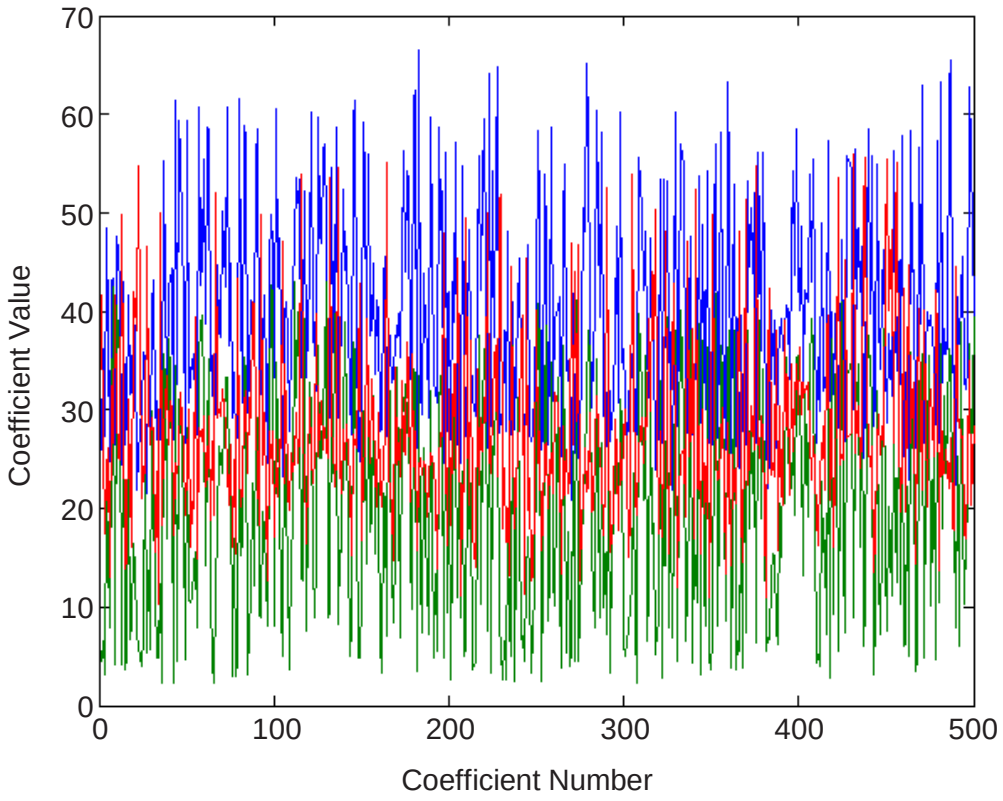


Figure 4.6: Three overlayed plots, each showing the coefficient values for a different image. 3 images were transformed with a randlet basis made up of 500 $200 \times 100$ Gaussian randlets. Each position on the x-axis corresponds to one of the 500 transform coefficients, while the y-axis shows coefficient values.

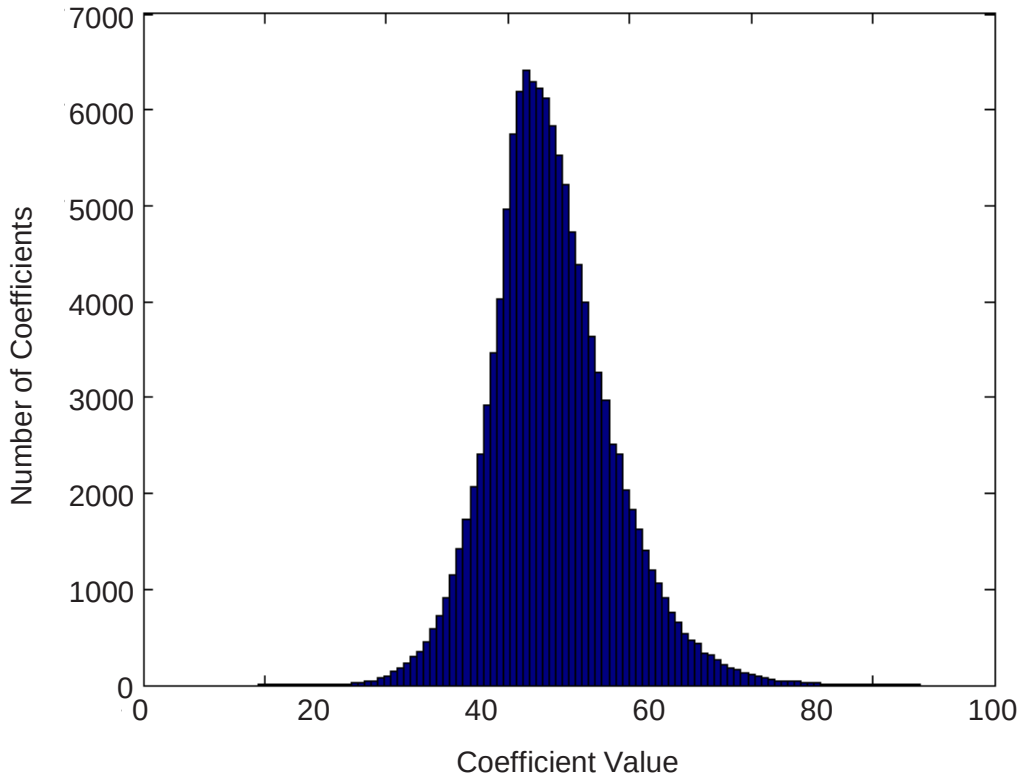


Figure 4.7: A histogram of coefficient values. 500 images were transformed with a randlet basis made up of 500 $200 \times 200$ Mexican Hat randlets, for a total of 250,000 coefficients in the histogram. The x-axis is coefficient values, the y-axis is the number of coefficients with each value.

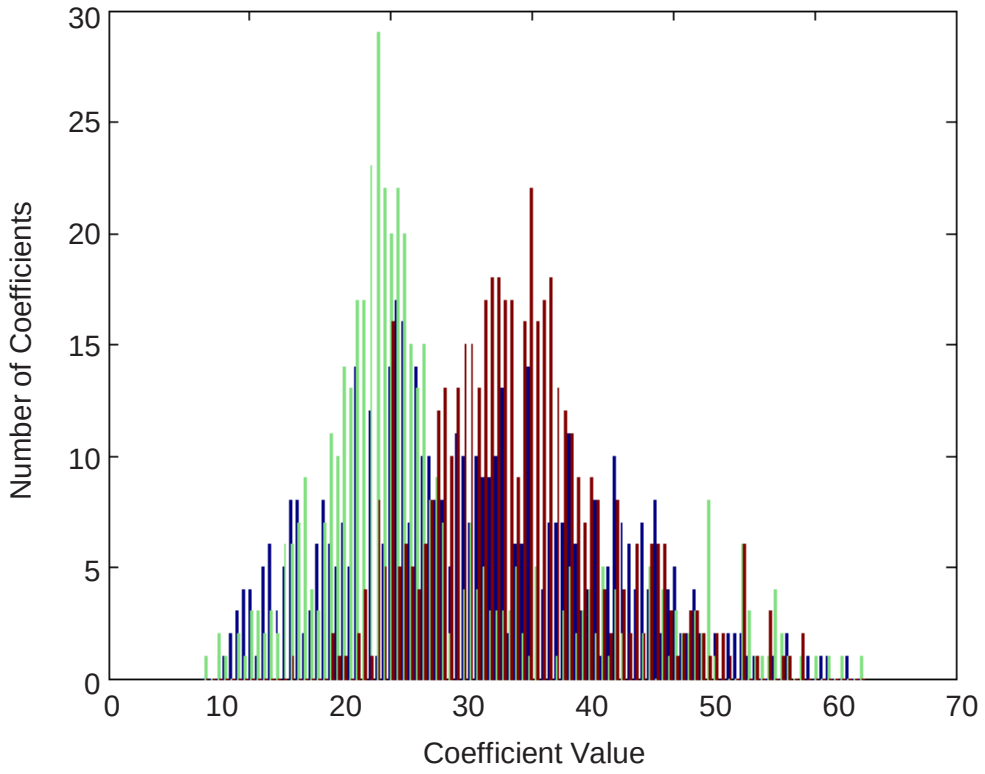


Figure 4.8: Three overlayed histograms, each of the coefficient values from a different image. 3 images were transformed with a randlet basis made up of 500 $200 \times 200$ Mexican Hat randlets, for a total of 1500 coefficients in the histogram. The x-axis is coefficient values, the y-axis is the number of coefficients with each value.

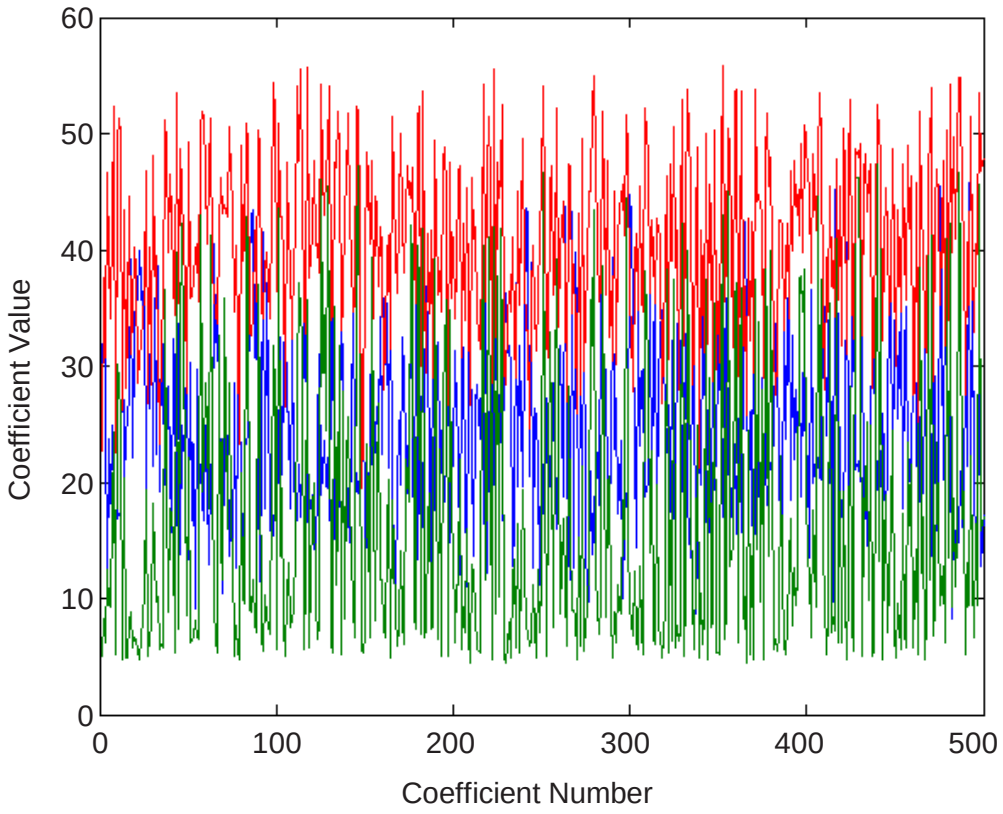


Figure 4.9: Three overlayed plots, each showing the coefficient values for a different image. 3 images were transformed with a randlet basis made up of 500 $200 \times 100$ Gaussian randlets. Each position on the x-axis corresponds to one of the 500 transform coefficients, while the y-axis shows coefficient values.

but there is substantial overlap in their distributions.

Figures 4.6 and 4.9 show overlayed plots of coefficients from three separate images. Figure 4.6 uses the same transform as Figure 4.4, and Figure 4.9 uses the same transform as Figure 4.7. Again, for both transforms, the coefficients of each image are distinct, but there is substantial overlap in their distributions.

For a DWT-based system, each image is transformed several times with the DWT, and the very lowest band is reshaped into a vector and assigned to $v$. It was determined experimentally that performance falls significantly when coefficients other than the low-low band are included in $v$. However, with each application of the DWT, the number of coefficients in the low-low band decreases by a factor of 4, which means that a 6-level DWT transform would result in only 16 coefficients. The randlet transform produces slightly superior performance to the 6-level DWT when only 16 randlet coefficients are used, but the randlet transform also has the option of increasing the number of coefficients to produce superior performance, an option the DWT does not have.

Quantization can be used to increase the entropy per bit of a hash. For example, instead of assigning each coefficient 12 bits, it may be beneficial to quantize the coefficients to 3 bits and generate 4 times as many coefficients. This is discussed in more detail in Section 4.2.1.

### 4.2.1 Experimental Results

All tests were performed on images taken from a library of 10,000 images. The images were of various sizes, but most were approximately $375 \times 265$ pixels in size. Unless otherwise stated, we ran our tests on 500 randomly-chosen images from the library. All of the wavelets present in the Matlab Wavelet Toolkit were tested. The Daubechies "db9" wavelet performed as well or better on image identification than any other wavelet tested, so we use it as the exemplar wavelet in all the DWT tests we present here.

The top chart in Figure 4.11 shows the results of cropping and rotating attacks. ROC curves are given for a $200 \times 200$ Gaussian randlet transform and for the 6-level

DWT transform, both of which performed well for their respective transform types. Neither transform had any trouble with scaling (both enlarging and shrinking), additive Gaussian noise with standard deviations of 0.1 and 0.2 (pixels take values from 0 to 1), and JPEG compression of 50% and 5%. These attacks are not shown because they would not be visible as they are too close to the axes. The randlet transform significantly outperforms the DWT against cropping, and vastly outperforms the DWT against rotation attacks.

Note that these results do not hold true for all randlet transform and wavelet transforms. For example, DWT transforms using 5 or fewer levels do significantly worse in all tests, some other wavelet families perform very badly, and various randlet bases also perform poorly (see below).

To simplify and clarify the presentation, a strong, canonical composite attack was used for the results which follow. Unless otherwise stated, all ROC charts illustrate image identification performance against this canonical attack. The attack is, in order:

**Rotation:** Rotation of 5 degrees around a point 45% of the way from the top of the image and 45% of the way from the left of the image. Detecting a rotation is more difficult when the center of rotation is not the center of the image.

**Cropping:** The bottom of the image is cropped by 5% and the right edge by 4%. Cropping the image asymmetrically is a stronger attack than cropping symmetrically.

**Scaling:** The image is shrunk to 80% of its original width and 90% of its original height.

**JPEG compression:** JPEG compression with quality of 10 is applied.

**Additive Gaussian Noise:** Additive Gaussian noise with a standard deviation of 0.2 is added. Pixel values range from 0 to 1, and if the noise pushes a pixel below 0 or above 1, it is set to 0 or 1, respectively.

Figure 4.10 shows the effects of the canonical attack on a sample image.

Figure 4.11 illustrates the performance of a randlet transform consisting of $200 \times 200$ Gaussian randlets and a 6-level Daubechies db9 wavelet transform against 10 and 20 degrees of rotation, and against 10% and 20% cropping. Figure 4.12 shows how several randlet and wavelet transforms fared against the canonical attack.


Figure 4.10: Illustration of the canonical attack. From left to right: the original image, the attacked image before adding Gaussian Noise, and the final attacked image

Experimentally, random randlets, Mexican Hat randlets, and the DWT with 1 to 5 levels all perform very poorly. The 7-level DWT also performed poorly, although not as horribly as the previous examples. Next are 300×300 Gaussian randlets, smooth random randlets, and the 6-level DWT, which performed decently. The best performers, by far, are $100 \times 100$ $200 \times 100$ and $200 \times 200$ Gaussian randlets.

Figure 4.13 shows how the number of coefficients affects image identification performance under the canonical attack. For each test a basis of 500 $200 \times 100$ Gaussian randlets was used. A random subset of all 500 coefficients was chosen and distances between images were computed with that subset of coefficients. Each test was run 20 times, except for the test with all 500 coefficients which was only run once.

Performance increases as the number of coefficients in a randlet transform increases, but there is a decreasing benefit to adding more coefficients. Using 5 or fewer coefficients gives poor performance under the canonical attack. 10, 20, and 50 coefficients perform increasingly well, although not near the asymptotically optimal performance. 100 coefficient performs about as well as well as 200 coefficients, and both perform almost as well as 500 coefficients, which experimentally seems to be about the asymptotically optimal performance level. Against the composite attack, 100 coefficients is very close to optimal, and 200 coefficients give performance almost equivalent to optimal.

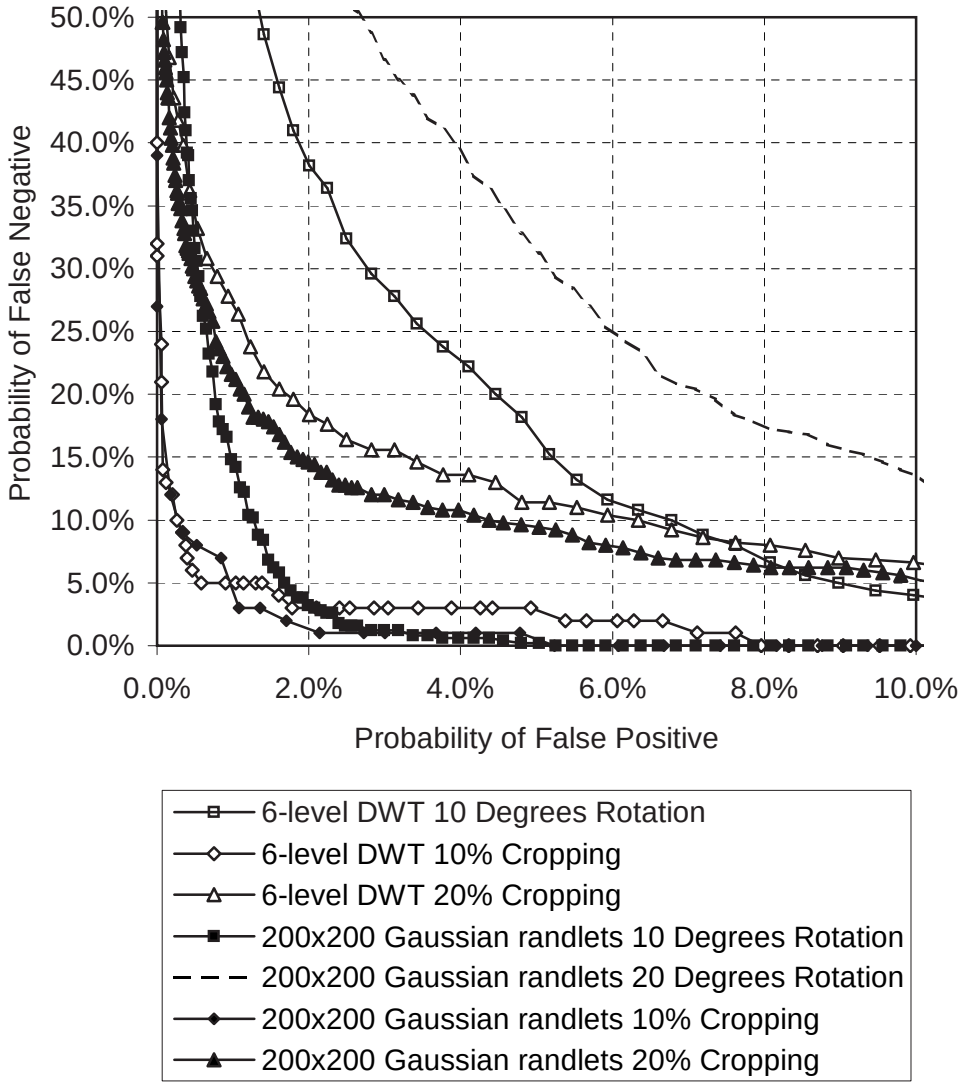


Figure 4.11: ROC curves for image identification against cropping and rotation attacks. A randlet transform consisting of $200 \times 200$ Gaussian randlets and a 6-level Daubechies db9 wavelet transform were tested.

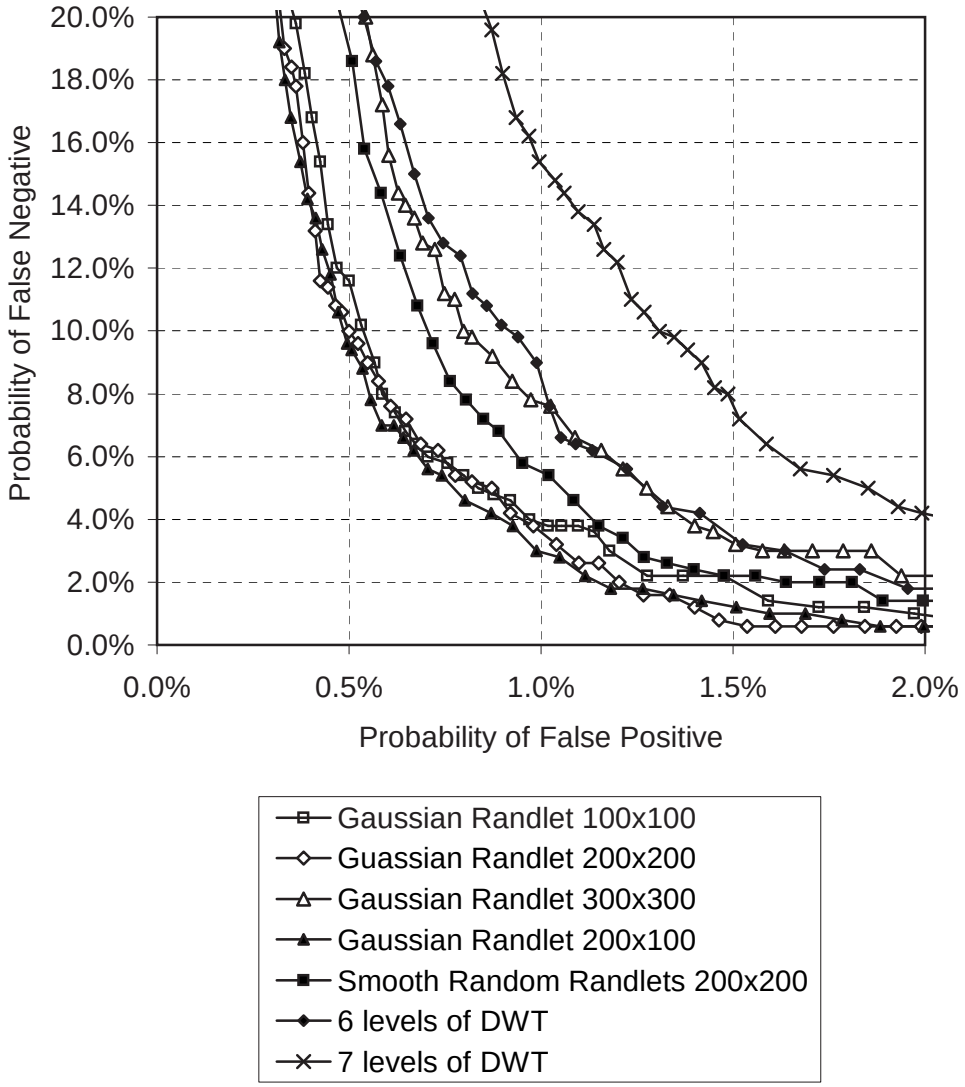


Figure 4.12: ROC curves for many different transforms against the canonical attack.

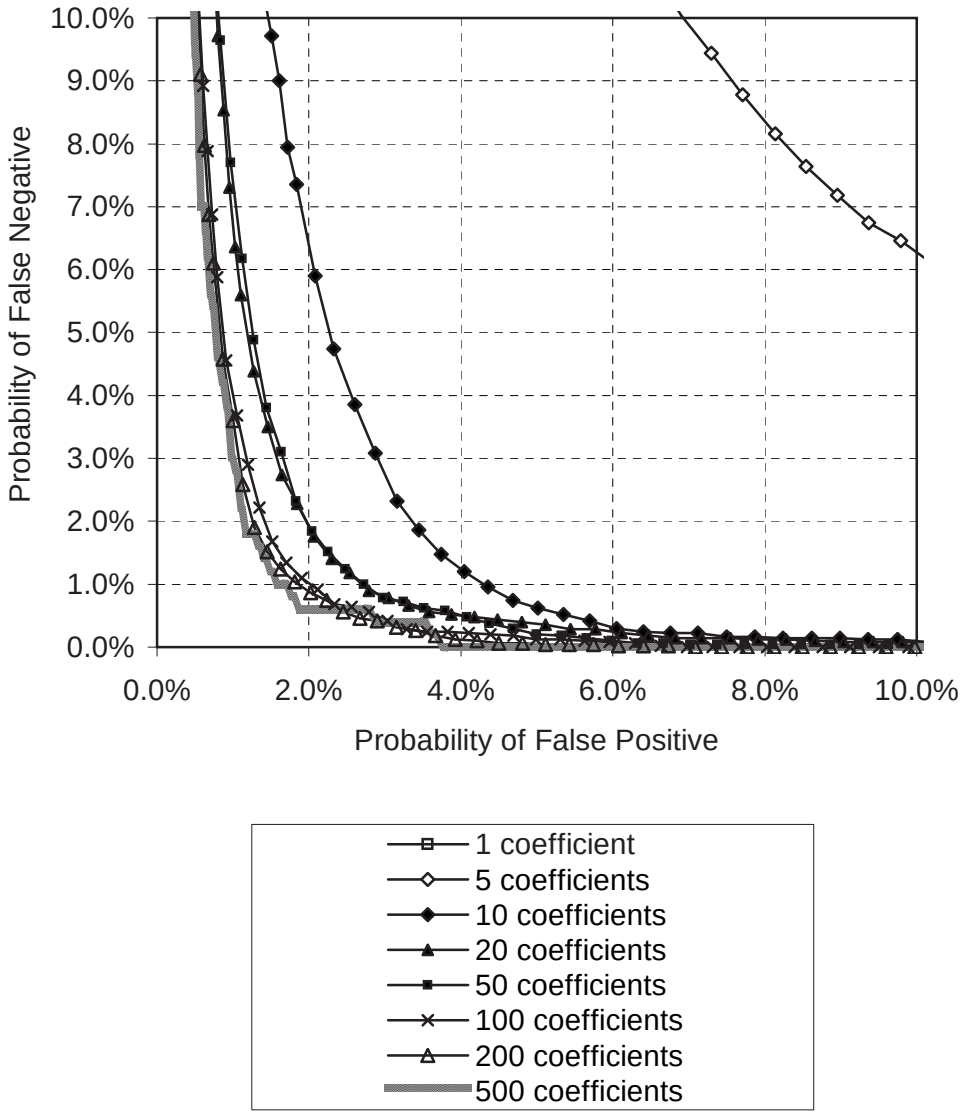


Figure 4.13: ROC curve showing how the number of coefficients affects the performance of image identification with the randlet transform. A basis of 500 $200 \times 100$ Gaussian randlets was used. Performance was so poor with a single coefficient that it does not even show up on the graph.

With 16 coefficients, a $200 \times 100$ Gaussian randlet transform performs slightly better than the 6-level DWT, which always has 16 coefficients. Not only does the randlet transform give better performance per coefficient than the DWT, it also has the ability to increase the number of coefficients arbitrarily, while the DWT does not.

Sometimes a basis with fewer randlets can outperform a basis with more randlets, even when the smaller basis is a subset of the larger one. This is because not all randlets are equal. Some randlets will tend to be better image identifiers with certain groups of images. To take advantage of this fact, an image identification system may generate a large randlet basis, run tests on a subset of images to find out which randlets give the best performance, and then choose only the best bases for use in the full system.

It is interesting to note that in our tests it was often possible to identify images with only a single coefficient. We tested this by projecting 20 $200 \times 100$ Gaussian randlets onto the entire 10,000 images at our disposal. Each randlet, by itself, was sufficient to distinguish all images in the library from each other. Of course, this only works in the absence of attacks, as figure 4.13 shows that a single coefficient has a horrible false negative probability. Also, these tests took advantage of the high precision of floating point numbers in Matlab.

### Quantization

We tested a uniform quantizer over a range of quantization step sizes. A reasonable step size was one where the coefficients would be well distributed among various values. Quantization with a reasonable step size only slightly affected the performance of the randlet transform. Once the step size became so large that the coefficients were no longer well distributed, performance fell dramatically.

We also tested a non-uniform quantizer. It generated bins that were equiprobable over all transform coefficients in all un-attacked images. For a specific image the coefficients would be more likely to fall in certain bins than others, but a coefficient randomly chosen over *all* images would be equally likely to fall into each bin. For each bin, the quantization point was the mean value of all coefficients in that bin.

Figure 4.14 shows the effects of quantizing randlet transform coefficients with the

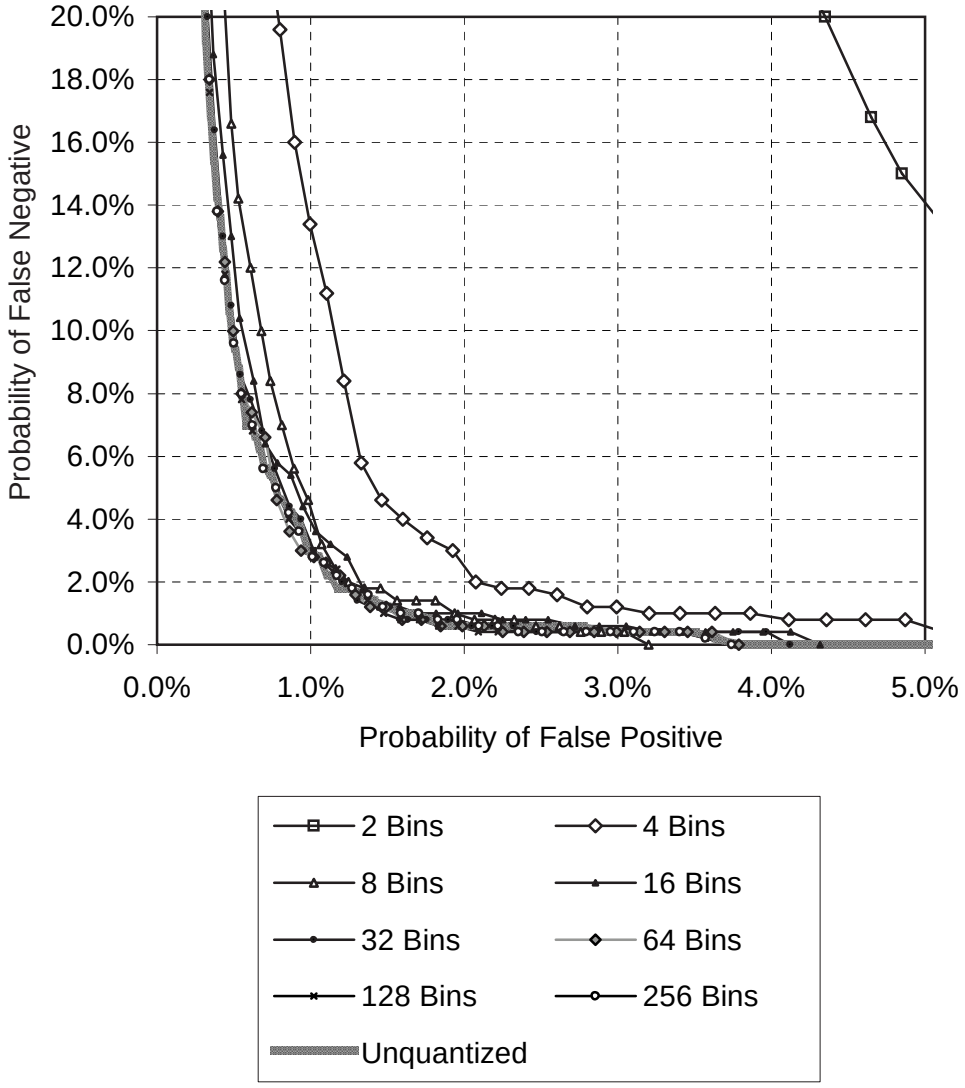


Figure 4.14: The results of quantizing the transform coefficients with a non-uniform quantizer. A basis of 500 $200 \times 100$ Gaussian randlets was used.

non-uniform quantizer. A basis of 500 $200 \times 100$ Gaussian randlets was used. The quantizer worked very well with the randlet transform. Performance suffered with too few bins, but with 8 bins performance was close to the unquantized performance, and with 16 bins it was almost equivalent. In these tests, therefore, 3 bits per coefficient is reasonable and 4 bits per coefficient achieves approximately the maximum performance possible with the randlet transform. Combining this with the results above about the required number of coefficients, we can see that against the canonical attack 300 bits per image almost achieves optimal performance, and 800 bits per image achieves optimal performance.

It would be desirable to have as few coefficients as possible change after an attack. To test this, transform coefficients were compared before and after the canonical attack. Figure 4.15 illustrates the probability that a particular coefficient will be changed by our canonical attack for several different transforms. The randlet transform performs significantly better than the DWT. The probability of the canonical attack changing a particular coefficient is low when there are few bins, but grows quickly as the number of bins grows.

### Entropy

For a deterministic transform and a given image, there is no uncertainty as to any of the image's transform coefficient values. As a consequence, it doesn't make sense to talk about the entropy of coefficients for a fixed image when taking a deterministic transform. With the randlet transform, if the secret key is unknown then there is still uncertainty as to an image's transform coefficient values, even for a fixed image. With the randlet transform it is possible to talk about the entropy of coefficient values for a fixed image by calculating over all possible key values. Of course, as a linear transform there are limits to the entropy of randlet transform coefficients. For example, a completely blank image would not result in any entropy in the coefficients.

When taken over a distribution of images, the uncertainty of coefficient values with the randlet transform comes from both the distribution of images and the randomization of the transform. On the other hand, the entropy of coefficients with deterministic transforms, such as the DWT, depends completely on the properties of

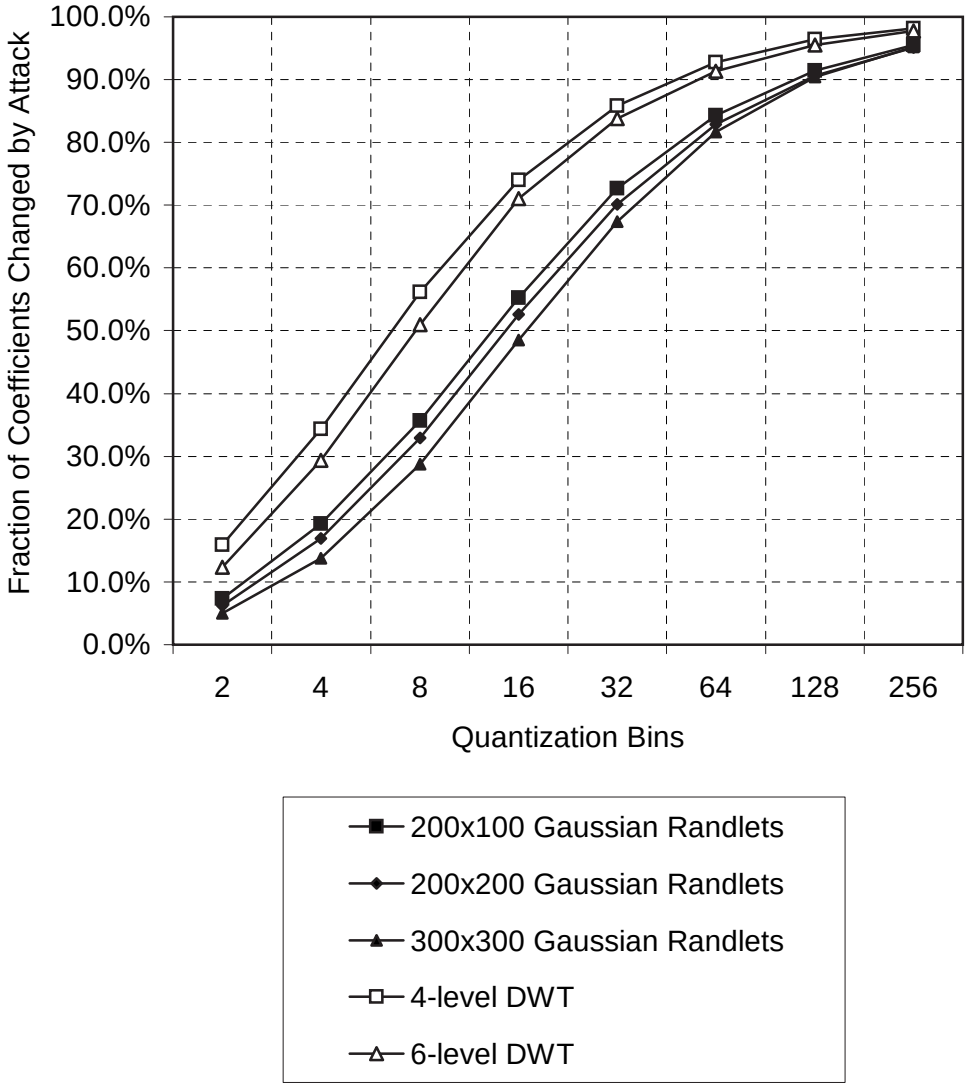


Figure 4.15: The probability that a coefficient will be changed by an attack, as a function of the number of quantization bins.

the image being transformed. Because it is only meaningful to talk of the entropy of DWT coefficients over a distribution of images, all entropy comparisons below do so.

Because the entropy of coefficients depends in part on the complexity of the images being transformed, we tested the variance of images and subsets of images to determine if there was sufficient complexity in most images. 500 randomly chosen images were scaled to $256 \times 256$ pixels and decomposed into 200 randomly chosen, overlapping rectangular subsets of random size and position. The variance was calculated for the pixels in each rectangle.

Figure 4.16 is a histogram displaying the number of rectangles with each level of variance when taken over all 200 rectangles in all 500 images tested. There is a spike of variances very close to 0, which presumably corresponds to areas of images which are solid-colored.

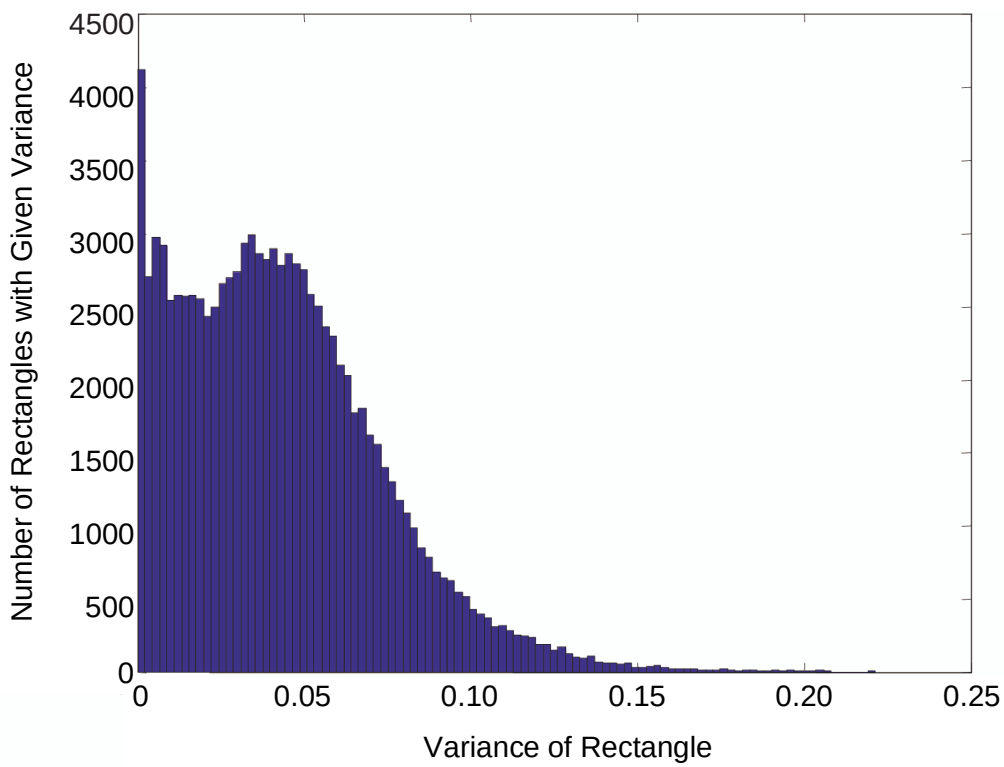


Figure 4.16: A histogram of the number of rectangles with each level of variance.

An arbitrary threshold was chosen so that over all rectangles from all images, a quarter of variances were less than the threshold and three quarters were greater than the threshold. Figure 4.17 is a histogram which orders images by the number of low-variance regions, defined as regions with variance less than the threshold. For each image, the number of rectangles exceeding the threshold was computed. This histogram shows how many images had each possible total. For example, at 100 on the x-axis the result is 2. That means that two images had exactly 100 rectangles with variance above the threshold and 100 rectangles with variance below the threshold. Note that most of the images are on the right of the graph, which corresponds to images with few low -complexity regions.

-

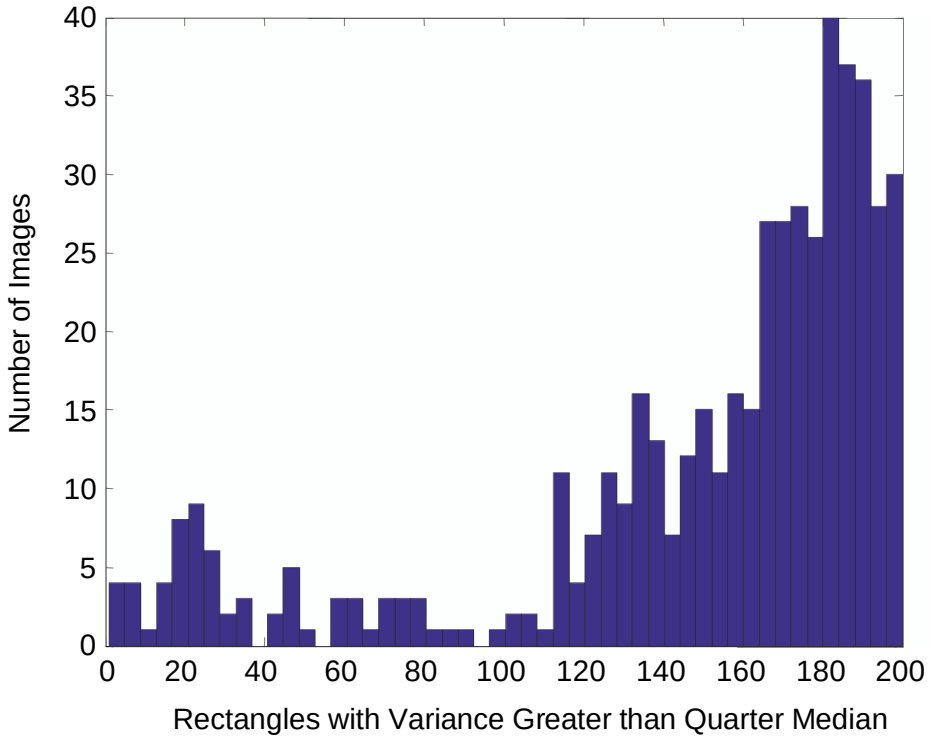


Figure 4.17: A histogram which orders images by the number of low-variance regions.

---

We conclude that most images contain a significant amount of variance relative to this threshold. In our tests approximately 5% to 10% of images had low variance

relative to this threshold in a large number random rectangles. For example, an image of a far-away boat on the ocean could have very little variance in the water and the sky, and only have variance near the boat and the line of the horizon.

Rectangles filled with uniformly random pixel values (in [0,1]) have an average variance of $8.3 \times 10^{-2}$. Of the images we tested, 96.4% had at least 75% of their subsets with a variance of at least $8.3 \times 10^{-3}$, and all of the images had at least 75% of their subsets with a variance of at least $8.3 \times 10^{-4}$. While some images do not have much variance, and many images have subregions of low variance, all the images we tested had enough variance for our purposes.

Figure 4.18 shows the entropy of transform coefficients for four transforms when taken over 500 images. All the coefficients for each transform were quantized with the non-uniform quantizer using various numbers of quantization bins. Then, each image was examined in turn and the entropy of each image's coefficients was calculated. The values shown in the chart are the entropy of each coefficient divided by the number of bits it would take to specify the coefficient's bin. We refer to this as the *normalized entropy*. The closer this value is to one, the closer the coefficients for that image are to being uniformly distributed across all bins, and the harder it is to predict the coefficients. Normalized entropy is used so that it is possible to meaningfully compare quantization with varying numbers of bins.

$200 \times 100$ Gaussian randlets can achieve an average normalized entropy per coefficient of 0.849 bits with 200 bins, while the $200 \times 200$ smooth random randlets can achieve an average of 0.998 bits with 2 bins. The 4-level DWT can achieve 0.880 bits with 52 bins, and the 6-level DWT can achieve 0.699 bits with 10 bins. Although in certain cases the DWT produces more entropy than the randlet transform, this does not detract from the randlet transform. There are many functions which produce a lot of entropy (for example, XOR) which are not very good as image hashes. It is sufficient that the randlet transform produces *enough* entropy per coefficient that the coefficients are difficult for an adversary to predict.

The 4-level DWT generates a reasonable amount of entropy. Unfortunately, it is very poor at image identification. The DWT with the best image identification performance was the 6-level DWT, but it is significantly worse than randlets at generating

entropy in coefficients.

The uniform quantizer implicitly puts the data into bins as well, however the number of bins is not known ahead of time. Entropy tests using the uniform quantizer showed that the average normalized entropy for the randlets was slightly diminished from nonuniform quantizer. For example, the $200 \times 100$ Gaussian randlets achieved about 0.816 bits, while the 6-level DWT achieved 0.722 bits.

## 4.3 Conclusions

The key point of the randlet transform is that it is built with structured randomness; structured so as to give good performance, random so as to avoid worst-case performance and to thwart attackers. It may seem strange, for example, that the DWT performed reasonably well against many attacks but performed so poorly against rotation attacks, and one may suspect that the attacks were carefully chosen to produce the given results[3]. However, this just illustrates the point that randomization can help to avoid worst case performance. Evidently, rotation attacks induce worst case performance in the DWT.

[3]One would, of course, be incorrect in that suspicion. The attacks were chosen because they seemed like reasonable attacks, and were chosen before the results were known. All of the attacks that were tested are presented.

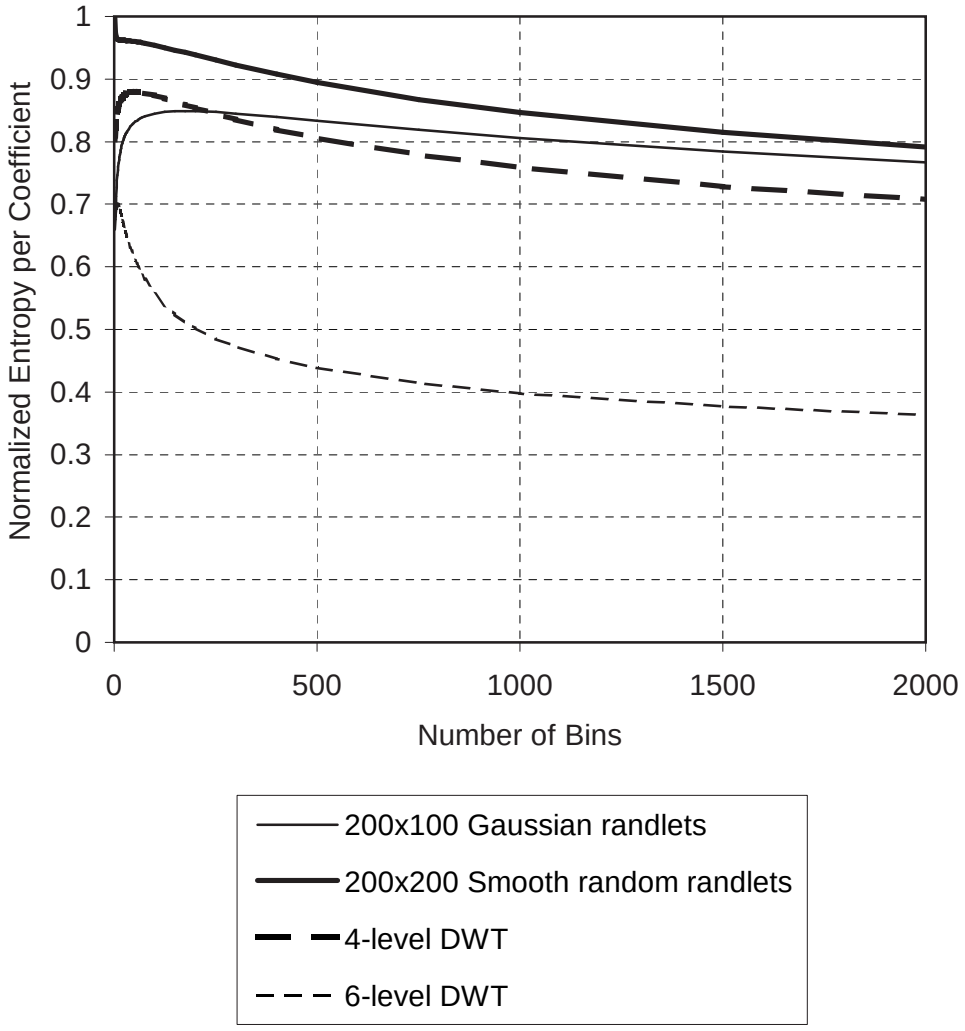


Figure 4.18: Normalized entropy of transform coefficients for various transforms.

# Chapter 5

# Image Watermarking with Randlets

An image watermarking is data hidden in a preexisting image in such a way that it does not significantly compromise the visual quality of the image and is still able to be extracted even if the watermarked image undergoes minor modifications. This chapter explores the application of the randlet transform to image watermarking. Mother randlets are chosen to be robust to attacks on the watermark and at the same time minimally perceptible when embedded in a watermark. Furthermore, an adversary who does not know the randlets is at a great disadvantage when attacking the watermark. We use a linear optimization algorithm to modify the randlet transform coefficients to improve detectability while inducing a minimum of noise in the image. At the watermark detector we consider both hard and soft decoding.

We consider two types of watermarking. With *verification* watermarking, the problem is to recognize if a given image has been watermarked with a given randlet basis. In this way, for example, ownership of an image can be verified by proving that a specific, secret randlet basis has been used to watermark an image. The second type of watermarking we consider is *data* watermarking, where bits of data are embedded into an image and are extracted by the watermark detector. This type of watermarking can also be used to prove ownership, but it can also be used to embed sideband information into an image.

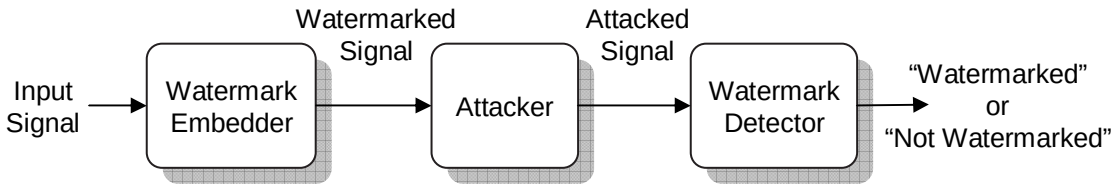


Figure 5.1: Verification watermarking attack model.

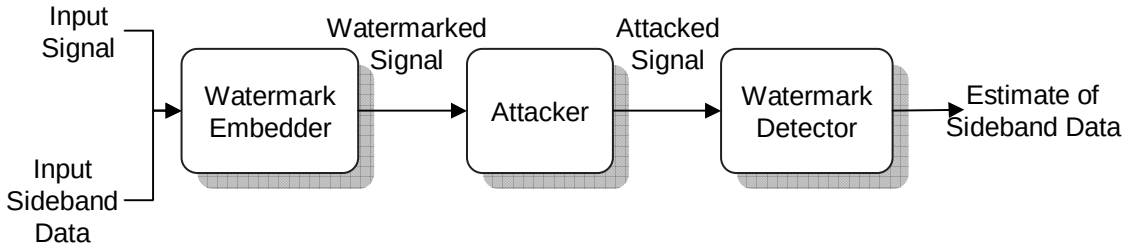


Figure 5.2: Data watermarking attack model.

Figure 5.1 shows a generic attack model for verification watermarking. An image is given to the watermark embedder, which embeds a watermark into the image. Then an attacker modifies the image, and the watermark detector must attempt to discover if its input was watermarked. Figure 5.2 shows a generic attack model for data watermarking. It is very similar to the model for verification watermarking, except that the watermark embedder receives input data in addition to the image, and embeds that data into the image. The watermark detector attempts to extract that data from the image that it receives.

The security of a randlet watermark is based on a secret key which is used to pseudo-randomly generate the randlet basis and randomized quantizers that are used. An attacker who knows the randlet basis has a much better chance of erasing the watermark. An attacker with knowledge of the randlet basis and the random quantizers can forge their own watermarks.

Section 5.1 contains a full description of our watermarking system and Section 5.2 describes real-world tests of the system.

### 5.0.1 Previous Work

Our scheme is similar to Quantization Index Modulation (QIM) as developed by Chen and Wornell [11], but we use non-orthogonal basis functions which introduces difficulties when quantizing transform coefficients. Mihcak et al. [35] use a watermarking scheme that is also quantization-based with non-orthogonal constraints, but they have much less structure in defining their watermark. Fridrich [16] uses QIM with a randomized orthogonal transform which is highly constrained.

## 5.1 Randlet Watermarking

The basis of our watermarking algorithm is the quantization of randlet transform coefficients, similar to ideas presented by Mihcak et al. [35] and Chen and Wornell [11]. We choose quantization watermark embedding over spread spectrum watermark embedding so that the image itself does not act as interference with the watermark.

To watermark an image, the randlet transform of the image is found using a secret randlet basis. The transform coefficients are quantized using secretly shifted uniform quantizers, and the image is modified so that transforming the watermarked image will yield the quantized coefficients. Because the randlet basis functions are nonorthogonal, there will be conflicting goals in this process. For example, changing the image so that one coefficient is quantized may affect the value of another coefficient. We use a linear optimization algorithm is to achieve the desired quantization while reducing the embedding distortion. Watermark detection involves taking the randlet transform and looking at how close each coefficient is to a quantization point on the secret quantizers.

As mentioned above, we consider verification watermarking, used only to tell if an image is watermarked or not; and data watermarking, where the watermark contains bits of side-band information. The two types of watermarks differ mainly in how they quantize the coefficients, although detection is also slightly different in each case.

For verification watermarks, a separate randomly shifted uniform quantizer is chosen for each transform coefficient. To embed a watermark, the randlet transform is

taken of an image and each transform coefficient is quantized with its corresponding quantizer. Detection is done by taking the randlet transform of an image and examining how close each coefficient is to any quantization point on its corresponding quantizer. If a threshold of coefficients are close to quantization points, the image is considered watermarked.

For data watermarks, each randlet embeds a single bit. A randomly-shifted pair of uniform quantizers with alternating quantization points is chosen. One quantizer corresponds to a bit of 0 and the other to a bit of 1. To embed a watermark, a randlet transform is taken of an image and each coefficient is quantized to the nearest quantization point on either the "0" or "1" quantizer, depending on whether a 0 or a 1 is to be embedded. To detect a watermark, the randlet transform is taken of an image, and a bit is assigned a value based on which quantizer has the closest quantization point.

The randlet basis that is used for watermarking depends only on the secret key, not on the specific image being watermarked. For an image-dependent watermark, an image hash can be used to supplement the secret key. Chapter 4 has details on producing a image hashes with randlets.

### 5.1.1 Definitions

We will consider watermarking an $n \times m$ pixel image with $k$ randlets. The image is stored internally as an $N \times 1$ column vector, where $N = nm$. This vector is formed by stacking the columns of the image on top of each other. Let $A$ be an image, and let the pixel at row $y$ and column $x$ be referenced as $A[x, y]$, $x \in [1, ..., n]$, $y \in [1, ..., m]$. Then the column-vector representation of $A$ is:

$$\mathbf{A} = (A[1,1], ..., A[1,m], A[2,1], ..., A[2,m], A[n,1], ..., A[n,m])^T.$$

The randlets and watermark are also $n \times m$ pixel images stored as $N \times 1$ column vectors as above. For a randlet transform with $k$ randlets, let $\mathbf{r}_1, ..., \mathbf{r}_k$ be randlets. For image watermarking, we use the forward randlet transform (Section 4.1.3), where each transform coefficient is the inner product of a randlet with the image. The

forward randlet transform produces a column vector of $k$ coefficients, $\mathbf{c} = (c_1, ..., c_k)^T$ such that $c_j = \mathbf{r}_j^T \mathbf{A}$. Therefore, we can write the randlet transform $T$ as a matrix of row vectors of randlets,

$$T = \begin{bmatrix} \mathbf{r}_1^T \\ \vdots \\ \mathbf{r}_k^T \end{bmatrix} \quad \text{and} \quad \mathbf{c} = TA.$$

$T$ is a $k \times N$ matrix.

### 5.1.2 Choosing Quantization Values

The first step in the randlet watermarking algorithm is to choose the a watermark. The randlet watermark exists in the randlet transform coefficients. If an image's randlet transform coefficients are close to the appropriate quantization points on secret watermarking quantizers then the image is considered watermarked. A watermark is embedded in an image by modifying the image in such a way that the randlet transform coefficients are close to quantization points.

We consider the case of scalar quantizers, where each coefficient is quantized separately, but note that it is also possible to use vector quantizers.

Recall that we consider two types of watermarking, verification watermarking and data watermarking. For verification watermarking, a randomly-shifted uniform quantizer is chosen for each coefficient. Define $Q_j$ to be the quantizer for coefficient $j$. Let $\Delta_j$ be the quantization step size and $\alpha_j \in [-\frac{\Delta_j}{2}, \frac{\Delta_j}{2}]$ be a random shift. Then the quantization points of $Q_j$ are

$$Q_j = \{..., -2\Delta_j + \alpha_j, -\Delta_j + \alpha_j, \alpha_j, \Delta_j + \alpha_j, 2\Delta_j + \alpha_j, ...\}.$$

Figure 5.3 shows an example quantizer for verification watermarking with $\alpha_j = 2$ and $\Delta = 6$.

Let $Q(\mathbf{c})$ denote operation of quantizing every coefficient in $\mathbf{c}$ with its corresponding quantizer. For verification watermarking, then, the quantization operation is


Figure 5.3: Example quantizer for verification watermarking. In this case, $\alpha_j = 2$ and $\Delta = 6$.

$Q(\mathbf{c}) = (Q_1(c_1), ..., Q_k(c_k))^T$.

For data watermarking, the quantizer above is split into two quantizers, $Q_j^0$ and $Q_j^1$, corresponding to embedding a 0 bit and a 1 bit, respectively. This is done by alternately assigning quantization points from the verification watermarking quantizer to $Q_j^0$ and $Q_j^1$. The result is that both are uniform quantizers with alternating quantization points, and each quantization point on one is midway between quantization points on the other. Furthermore, the pair of quantizers is randomly shifted. As with verification watermarking, for randlet $j$, let $\Delta_j$ be the quantization step size and $\alpha_j \in [-\Delta_j, \Delta_j]$ be a random shift. Then quantization points for $Q_j^0$ and $Q_j^1$ are

$$Q_j^0 = \{..., -4\Delta_j + \alpha_j, -2\Delta_j + \alpha_j, \alpha_j, 2\Delta_j + \alpha_j, 4\Delta_j + \alpha_j, ...\},$$

$$Q_j^1 = \{..., -3\Delta_j + \alpha_j, -\Delta_j + \alpha_j, \Delta_j + \alpha_j, 3\Delta_j + \alpha_j, ...\}.$$

Figure 5.4 shows an example of dithered quantizers for data watermarking with $\alpha_j = 2$ and $\Delta = 5$.

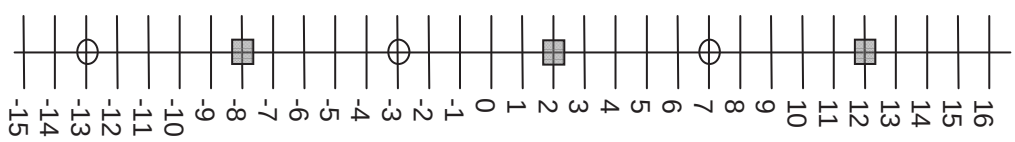


Figure 5.4: Example of dithered quantizers for data watermarking. In this case, $\alpha_j = 2$ and $\Delta = 5$. Shaded squares correspond to $Q_j^0$ while circles correspond to $Q_j^1$.

Let $\mathbf{v} \in \{0, 1\}^k$ be the secret watermarking data to be embedded and let $v_j$ be bit $j$

of $\mathbf{v}$. Given quantizers $Q_j^0$ and $Q_j^1$ for all $j \in \{1, ..., k\}$, define $Q(\mathbf{c})$ to be the operation of applying $Q_j^{v_j}$ to $c_j$, so the quantization operation is $Q_{\mathbf{v}}(\mathbf{c}) = (Q_1^{v_1}(c_1), ..., Q_k^{v_k}(c_k))^T$.

In further sections, the type of quantization, either verification or data watermarking, will be obvious from the context.

### 5.1.3 Watermark Embedding

Because the forward randlet transform is linear, it is possible to represent it in matrix form. In practice such a representation is inefficient in both time and space, prohibitively so with large images or many randlets. However, the watermarking algorithm is conceptually related to the matrix formulation, so we will begin by considering the watermarking algorithm in matrix form, and then relate that to a more practical algorithm.

Image $A$ has randlet transform $\mathbf{c} = \mathbf{TA}$. We choose the watermark to be the smallest $\mathbf{W}$ such that $\mathbf{T}(\mathbf{A} + \mathbf{W}) = Q(\mathbf{c})$. It is generally not possible to inver the matrix $\mathbf{T}$, so we choose $\mathbf{W}$ to be the min-norm solution. Because $\mathbf{T}(\mathbf{A} + \mathbf{W}) = \mathbf{c} + \mathbf{TW} = Q(\mathbf{c})$, we can write

$$\mathbf{TW} = (Q(\mathbf{c}) - \mathbf{c}).$$

**Lemma 5.1.1** *The solution to minimize* $\mathbf{W}$ *such that* $\mathbf{T}(\mathbf{A}+\mathbf{W}) = Q(\mathbf{c})$ *is given by*

$$\mathbf{W} = \mathbf{T}^T(\mathbf{TT}^T)^{-1}(Q(\mathbf{c}) - \mathbf{c}). \tag{5.1}$$

As the min-norm solution in Lemma 5.1.1 implies, the watermarked image, $\hat{\mathbf{A}}$, is

$$\hat{\mathbf{A}} = \mathbf{A} + \mathbf{W} = \mathbf{A} + \mathbf{T}^T(\mathbf{TT}^T)^{-1}(Q(\mathbf{c}) - \mathbf{c}) \tag{5.2}$$

This is a simple solution, but since each randlet is the size of $\mathbf{A}$, $\mathbf{T}$ is a very large matrix. For example, to embed 500 coefficients into an image of size $512 \times 512$, $\mathbf{T}$ would be $500 \times 262144$ and have over 131 million elements. In order to be practical, it is necessary to speed up the calculation of $\mathbf{W}$.

Since randlets have compact support, a large fraction of the elements of are zero. Furthermore, the non-zero entries are not scattered randomly around the matrix, but are highly ordered. We should, therefore, be able to improve the algorithm. We will do so by examining the nature of the min-norm solution.

Consider the randlet transform matrix:

$$\mathbf{T} = \begin{pmatrix} \leftarrow & \mathbf{r}_1^T & \rightarrow \\ \leftarrow & \mathbf{r}_2^T & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{r}_k^T & \rightarrow \end{pmatrix}, \qquad \mathbf{T}^T = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{r}_1 & \mathbf{r}_2 & \dots & \mathbf{r}_k \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}.$$

Multiplication of an image by $\mathbf{T}$ is simply the randlet transform, $\mathbf{TA} = \mathbf{c}$, where each transform coefficient is the inner product of a randlet (one row of $\mathbf{T}$) with the image vector. Now consider multiplying an arbitrary vector $\mathbf{c}$ with the transpose of the transform matrix. $\mathbf{B} = \mathbf{T}^T\mathbf{c}$ is an image which is formed as a linear combination of randlets, with randlet $j$ being weighted by $c_j$. In other words, this is the inverse randlet transform,

$$\mathbf{B} = \sum_{j=1}^{k} c_j \mathbf{r_j}.$$

Now let us consider the watermark, $\mathbf{W} = \mathbf{T}^T(\mathbf{TT}^T)^{-1}(Q(\mathbf{c}) - \mathbf{c})$. The first step in computing the watermark is to find $\mathbf{S} = \mathbf{TT}^T$. If the element of $\mathbf{S}$ at row $i$, column $j$ is denoted $s_{ij}$, then $s_{ij} = \mathbf{r}_i^T\mathbf{r}_j$. That is, the element at row $i$ and column $j$ is the inner product of randlet $i$ and randlet $j$,

$$\mathbf{S} = \mathbf{TT}^T = \begin{pmatrix} & \vdots & \\ \dots & s_{ij} & \dots \\ & \vdots & \end{pmatrix}, \qquad s_{ij} = \mathbf{r}_j^T\mathbf{r}_j.$$

Examining $\mathbf{S}$ can greatly speed up the watermarking process. Since the randlets are normalized, $s_{ii} = 1$, and since the inner product is commutative, $s_{ij} = s_{ji}$. We therefore have to compute fewer than half of the elements in the matrix. Of the

matrix elements that must be computed, $s_{ij} = 0$ if randlet $i$ and randlet $j$ do not overlap. Furthermore, the inner product of $\mathbf{r}_i$ and $\mathbf{r}_j$ only needs to be computed where randlet $i$ and randlet $j$ overlap, instead of computing the inner product of two length-$N$ vectors, where $N$ is very large.

Multiplying the $k \times 1$ vector $(Q(\mathbf{c}) - \mathbf{c})$ by the $k \times k$ matrix $\mathbf{S}^{-1} = (\mathbf{T}\mathbf{T}^T)^{-1}$ gives an $M \times 1$ vector of weights, which we will call $\mathbf{v}$:

$$\mathbf{v} = (\mathbf{T}\mathbf{T}^T)^{-1}(Q(\mathbf{c}) - \mathbf{c})$$

As was shown earlier, $\mathbf{T}^T\mathbf{v}$ is simply the inverse randlet transform of $\mathbf{v}$. That is, if $\mathbf{v} = (v_1, w_2, ..., w_k)^T$, then the result is an image formed by the sum of randlet $j$ multiplied by $w_j$.

$$\mathbf{T}^T\mathbf{v} = \mathbf{T}^T(\mathbf{T}\mathbf{T}^T)^{-1}(Q(\mathbf{c}) - \mathbf{c}) = \sum_{i=1}^{k} v_j \mathbf{r_j}$$

We can find the inverse transform by multiplying each randlet by its corresponding weight and adding the results together. Since each randlet has compact support, and conversely because $\mathbf{T}^T$ is so large, this is generally much faster than performing the matrix multiplication.

So to find the optimum watermark, we first calculate $\mathbf{S} = \mathbf{T}\mathbf{T}^T$ by finding the inner product of every randlet with every other randlet. We form a vector of weights by inverting $\mathbf{S}$ and multiplying the desired change in transform coefficients by that inverse. Finally, we perform a randlet-by-randlet inverse randlet transform on vector of weights to produce the watermark, which is added to the original image.

Note that $\mathbf{T}\mathbf{T}^T$ must have a small condition number in order for the watermark embedding to be successful. If the condition number is too large there will be too much embedding distortion. Generally, the condition number becomes large when too many very large randlets are used in the randlet basis, so using fewer or smaller randlets will prevent large condition numbers. Experimentally, this has not been problematic. Also note that it is possible to use linear regularization conditions with the linear optimization, or to use non-linear optimizations. However, we have found that when the condition number is small these steps are unnecessary.

Finally, it is possible to use a system similar to the distortion compensation mentioned by Chen and Wornell [11]. The quantization step size is increased, but the watermark is only partially embedded in the image. If $\Delta$ is the quantization step size, and $\alpha < 1$ is the distortion compensation parameter, then define $\Delta' = \Delta/\alpha$ to be the new step size and $\mathbf{W}' = \alpha\mathbf{W}$ to be the new watermark. Distortion compensation generally improves performance.

### 5.1.4 Watermark Detection

We consider the case of blind watermark detection. The detector knows the randlet basis and the quantizers used. The detector is given an image, possibly watermarked, possibly attacked, to perform detection upon. To detect a watermark, the randlet transform is taken of an image, and the resulting coefficients are checked to see how close they are to quantization points.

There are several different permutations of watermark detection to consider. First is verification watermarks versus data watermarks, and second is hard detection versus soft detection.

**Verification versus data watermarks:** As described above, this is the distinction between a watermark which carries sideband information, and one which only identifies an image as being watermarked. In both cases, detection will put each transform coefficient into two categories. For verification watermarking, the categories are "watermarked" and "unwatermarked". For data watermarking, the categories are 0 and 1.

**Hard versus soft detection:** Just as in coding, watermark detection can be hard or soft. With hard detection, each coefficient is examined individually and put into one of the two categories independent of the other coefficients. With soft detection, coefficients are assigned probabilities of being in each category, and groups of coefficients are considered together.

Error correcting codes can be used with data watermarking, and in fact must be used in order to do soft detection. The secret data bits are encoded with an

error-correcting code, and the bits of the resulting codeword are embedded as the watermark. For hard decoding, all bits of the watermark are determined individually and are decoded to find the secret data bits. When soft decoding, each bit is assigned a probability of being a 0 and a 1, and these probabilities are given to a soft decoder.

### Verification Watermark, Hard Detection

For hard detection of a verification watermark, a coefficient is considered *marked* if it is within a given radius $\delta$ of any quantization point. If a threshold $\tau$ of coefficients are marked, then the image is assumed to have been watermarked. Otherwise, the image is considered unwatermarked.

Let $\mathbf{c} = (c_1, ..., c_k)^T$ be the randlet transform coefficients of the image being considered. Choose as system parameters $\delta_j \in (0, \frac{\Delta_j}{2})$, and $\tau \in [0, 1]$. Recall that coefficient $c_j$ is quantized with quantizer $Q_j$. Define

$$F_\delta(x) = \begin{cases} 1 & \text{if } |x| \leq \delta \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Then an image is considered watermarked if

$$\frac{1}{k} \sum_{j=1}^{k} F_{\delta_j}(Q_j(c_j) - c_j) \geq \tau. \tag{5.4}$$

### Verification Watermark, Soft Detection

The soft detector for verification watermarks is similar to the hard detector, but instead of using the function $F_\delta$ to reduce each coefficient to a 0 or 1, it estimates the probability that each coefficient is marked and uses those probabilities to decide if the image is watermarked or not.

Given $\mathbf{c} = (c_1, ..., c_k)^T$ as before, let $P_j : [0, \frac{\Delta_j}{2}] \rightarrow (0, 1)$ take as input the distance of coefficient $c_j$ to the nearest quantization point on $Q_j$ and return the probability that the coefficient was marked. $P_j(x)$ is an estimate of the probability that coefficient $c_j$ was marked if it was distance $x$ from $Q_j(c_j)$. $P_j$ can be chosen based on theoretical models, or determined experimentally by examining the effects of attacks

on watermark coefficients. Note that even if a coefficient is on a quantization point, there is a chance that $c_j$ was not originally marked ($P_j(0) \neq 1$), and even if a coefficient is maximally far from a quantization point, there is a chance that $c_j$ was originally watermarked ($P_j(\frac{\Delta_j}{2}) \neq 0$).

Let $p_j$ be our estimate of the probability that $c_j$ was marked, $p_j = P_j(Q_j(c_j) - c_j)$. The soft verification detector considers an image to be watermarked if the likelihood of being watermarked is sufficiently larger enough than the likelihood of not being watermarked. Letting $e^\tau \geq 1$ be a threshold constant, we write that the image is watermarked if

$$\prod_{j=1}^{k} p_j > e^\tau \prod_{j=1}^{k} (1 - p_j). \tag{5.5}$$

Taking the natural log of Equation 5.5, we see that an image is considered watermarked if

$$\sum_{j=1}^{k} \ln \frac{p_j}{1 - p_j} > \tau, \tag{5.6}$$

$$\sum_{j=1}^{k} \ln \frac{P_j(Q_j(c_j) - c_j)}{1 - P(Q_j(c_j) - c_j)} > \tau. \tag{5.7}$$

**Data Watermark, Hard Detection**

With data watermarks, each randlet (and therefore each transform coefficient) encodes a single bit. The goal is that the data extracted from an image, $\mathbf{w} = (w_1, ..., w_k)$ is equal to the secret data that was embedded in the image, $\mathbf{v} = (v_1, ..., v_k)$. As described in Section 5.1.2, there are two uniform quantizers for each coefficient. $Q_j^0$ is used to encode 0 bits and $Q_j^1$ is used to encode 1 bits, each with step size $2\Delta$.

$$Q_j^0 = \{..., -4\Delta_j + \alpha_j, -2\Delta_j + \alpha_j, \alpha_j, 2\Delta_j + \alpha_j, 4\Delta_j + \alpha_j, ...\},$$

$$Q_j^1 = \{..., -3\Delta_j + \alpha_j, -\Delta_j + \alpha_j, \Delta_j + \alpha_j, 3\Delta_j + \alpha_j, ...\}.$$

Note that the quantization points on $Q_j^0$ and $Q_j^1$ alternate, so any coefficient $c_j$

that is not directly on a quantization point on one of the quantizers will be between a quantization point on $Q_j^0$ and a quantization point on $Q_j^1$. With hard detection, coefficient $c_j$ is assigned a value of 0 if it is closer to the quantization point on $Q_j^0$ and is assigned a value of 1 if it is closer to the quantization point on $Q_j^1$. Given $\mathbf{c} = (c_1, ..., c_k)^T$ as before, let

$$w_j = \begin{cases} 0 & \text{if } |Q_j^0(c_j) - c_j| < |Q_j^1(c_j) - c_j| \\ 1 & \text{otherwise} \end{cases} \tag{5.8}$$

After each bit has been assigned a value, an error correcting code is generally used to further reduce bit errors.

### Data Watermark, Soft Detection

Instead of assigning a bit value to each coefficient, the soft decoder calculates for each coefficient the probability that it corresponds to a 0 and the probability that it corresponds to a 1. An error correcting decoder then takes these probabilities and performs soft decoding to find the output bits. Soft decoding inherently uses error correcting codes, and therefore so does soft watermark detection. Error correcting codes can also be used to perform hard decoding after the soft decoding.

Given $\mathbf{c} = (c_1, ..., c_k)^T$ as before, let the embedded codeword be $\mathbf{v} = (v_1, ..., v_k)$. Similarly to soft decoding of verification watermarks, let $P_j : [0, \Delta_j] \rightarrow (0, 1)$ take as input the distance of coefficient $c_j$ to the nearest quantization point and return the probability that the coefficient was originally quantized to that quantization point. As before, $P_j$ is determined experimentally by examining the effects of attacks on watermark coefficients. Note that as before, $P_j(0) \neq 1$ and $P(\Delta_j) \neq 0$.

To perform soft detection, each bit is estimated to be a 0 with probability $P_j(Q_j^0(c_j) - c_j)$ and a 1 with probability $P_j(Q_j^1(c_j) - c_j)$, and these probabilities are passed to a soft decoder. As an example, we present a soft decoder for a repetition code.

Let $\mathbf{s} = (s_1, ..., s_n)$ be the secret data bits, and use a repetition rate of $r$, defining the total number of embedded bits to be $k = nr$. The embedded watermark is $\mathbf{v} = (v_1, ..., v_k)$, where $v_i = s_{\lceil \frac{i}{r} \rceil}$. The detected watermark is $\mathbf{w} = (w_1, ..., w_n)$. $w_j$ is

determined by examining coefficients $c_{(j-1)r+1}, ..., c_{jr}$.

Let $p_j^0$ and $p_j^1$ be our estimates of the probability that $c_j$ was originally quantized to 0 and 1, respectively. Then $p_j^0 = P_j(Q_j^0(c_j) - c_j)$ and $p_j^1 = P_j(Q_j^1(c_j) - c_j)$. Comparing likelihoods, we estimate that $s_j = 0$ if

$$\prod_{i=(j-1)r+1}^{jr} p_j^0 > \prod_{i=(j-1)r+1}^{jr} p_j^1,$$

$$\sum_{i=(j-1)r+1}^{jr} \ln p_j^0 - \ln p_j^1 > 0,$$

$$\sum_{i=(j-1)r+1}^{jr} \ln \frac{P_j(Q_j^0(c_j) - c_j)}{\ln P_j(Q_j^1(c_j) - c_j)} > 0.$$

So the final decision is that

$$s_j = \begin{cases} 0 & \sum_{i=(j-1)r+1}^{jr} \ln p_j^0 - \ln p_j^1 > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{5.9}$$

## 5.2 Experimental Results

Experimentally, attacks on an image produce very specific distributions of errors on the randlet transform coefficients. Desynchronization attacks, such as rotation and cropping, induce error distributions similar to a Laplacian distribution, while attacks that do not affect synchronization induce errors according to a normal distribution.

The standard method of comparing images is mean square error. However, this measure does a poor job of determining how visible a watermark is in an image. In our tests, we measured mean square error and also evaluated visible distortion, and found that often a randlet watermark with less mean square error would be more visible than another randlet watermark with higher mean square error. For our tests, a level of distortion was chosen that was just noticeable.

Since the watermarks are fairly robust against desynchronization attacks, it is feasible to perform searches over the inverses of these attacks to find the watermark. For example, since the watermark extremely robust against rotations of 1.5 degrees, we have implemented a search which can successfully find the watermark at arbitrary rotations by incrementally rotating the watermarked image and testing for the watermark.

In our tests we found that small randlets were not as robust against attacks, while larger randlets begin to overlap too much, greatly limiting the number of randlets that could be used. The frequencies present in the randlets were also important. Randlets with many high-frequency components were not be as robust against desynchronization attacks (for example, random randlets), and randlets with only low-frequency components raised the condition number of $\mathbf{S}$ and greatly increased embedding distortion. Soft detection was found to be much more robust than hard detection, as expected. Smooth random randlets (see Section 4.1.1) were the best compromise and had the best robustness.

Figure 5.5 shows the results of randlet watermarking for a single image. In the upper-left quadrant is the original image, while in the upper-right quadrant is the watermarked image. The lower-left quadrant is the absolute value of the watermark, which is the absolute value of the difference between the original and watermarked images. Finally, the lower-right quadrant is a version of the watermark which has been linearly equalized so that the pixel values span the maximum range.

Figure 5.6 shows the performance for verification watermarking with soft detection. The y-axis is the probability of a true positive, the x-axis is the probability of a false positive. 15 images of size $512 \times 512$ were watermarked with an average watermark to signal ratio of 30db. 3 different randlet bases of $80 \times 70$ smooth random randlets were tested with 401 coefficients were embedded per image. Finally, distortion compensation was used with $\alpha = 0.5$. Performance was perfect for 1.5 degrees rotation, 2% cropping, additive Gaussian noise of variance 0.25 (pixel values in [0,1]), and JPEG compression with quality 10. Figure 5.6 shows the performance against rotation and cropping attacks.

Figure 5.5: The upper-left quadrant is the unwatermarked image, the upper-right is the watermarked image, the lower-left is the magnitude of the watermark, and the lower-right is an equalized version of the watermark.

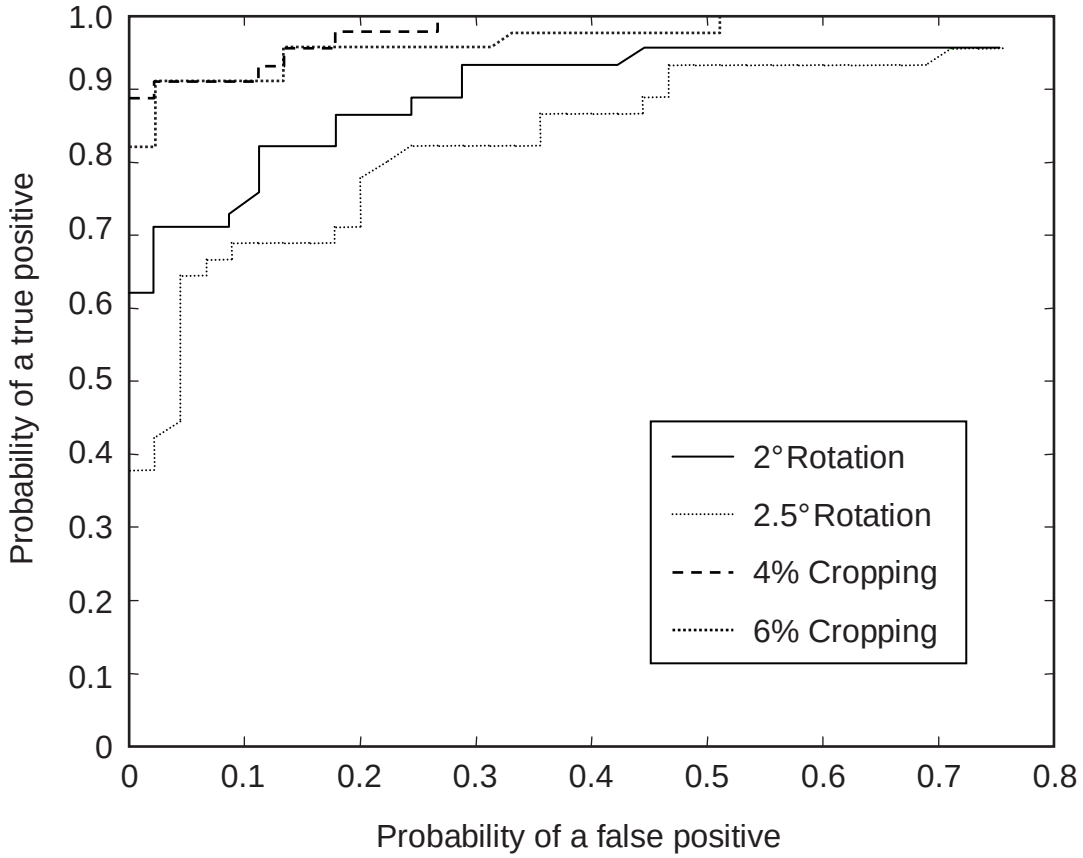


Figure 5.6: Performance of verification watermarking with soft detection.

## 5.3 Conclusion

We have implemented a quantization-based watermarking system using the randlet transform. In contrast to QIM, the image features which are quantized (the randlet transform coefficients) are non-orthogonal, and therefore an optimization routine is used to perform watermark embedding. The randlet watermarking system performs well against a variety of attacks.

# Chapter 6

# Text Sifting

Text sifting is a method of generating document fingerprints. These fingerprints can be used to identify a document even when an intelligent plagiarist attacks the document with the express purpose of changing its fingerprint. In this sense, text-sifting fingerprints are attack-resistant.

Text sifting systems form document fingerprints by extracting a small amount of text from each document, and then use these fingerprints to compare documents. Our initial methods were similar to those of Broder et al. [6, 7] and Heintze [20] (see Section 6.0.2) but were developed independently.

The purpose of a text sifting system is to detect when documents are similar. We assume that there is an intelligent adversary, a plagiarist, who knows that a system will be used to detect similar documents and takes steps to circumvent it. The adversary would like to do so by making as few changes to documents as possible. In reality, of course, there may not be an intelligent adversary; the "adversary" may be human error or chance.

The most complete solution to this problem would be to perform an exhaustive comparison on all pairs of documents, but the number of documents (say, web pages on the Internet) could make this infeasible. Methods of solving this problem have been proposed (see Section 6.0.2), but none of them use an adversarial attack model, and all can fall prey to simple attacks. Text sifting can be used to quickly identify documents while remaining resistant to adversaries who are aware that text sifting is

being used.

A text sifting function takes a document and produces a fingerprint which consists of *clusters*. Clusters are groups of words from the document which are ordered as in the document but are not necessarily contiguous in the document. Text sifting fingerprints have a few important properties:

1. *Random-looking:* The fingerprint is calculated deterministically using a secret key, but the clusters which are output can not be predicted without the key.

2. *Self-synchronizing:* If a cluster appears in two documents and is chosen for the fingerprint of one document, it has a good chance of being chosen for the fingerprint of the other document.

3. *Attack resistant:* The output should remain substantially unchanged when the original document is subject to attacks.

Previous systems have implemented the first and second properties, but remain susceptible to certain attacks. For example, if a system used a sliding window of length 10, an attacker could change a document so that every ten-word span contained a single change. The system would report that the attacked document was completely unrelated to the original document. Text sifting is designed to identify documents in spite of this sort of attack.

The security of text sifting systems comes from the unpredictability of the document fingerprints. The idea is that an attacker without the secret key cannot know what a document's fingerprint will be, so they can only attack the document by making many changes and hoping that the changes significantly alter the document's fingerprint. It turns out that systems which use a sliding window to generate fingerprints are susceptible to powerful attacks. One of our contributions is to consider general attacks on such systems, as well as specific attacks on the sliding window used in other systems. We improve security with a more robust selection process that replaces the sliding window.

Figure 6.1 shows a model of a plagiarism detection system. A library of document fingerprints is first generated. When the system receives a document, presumably

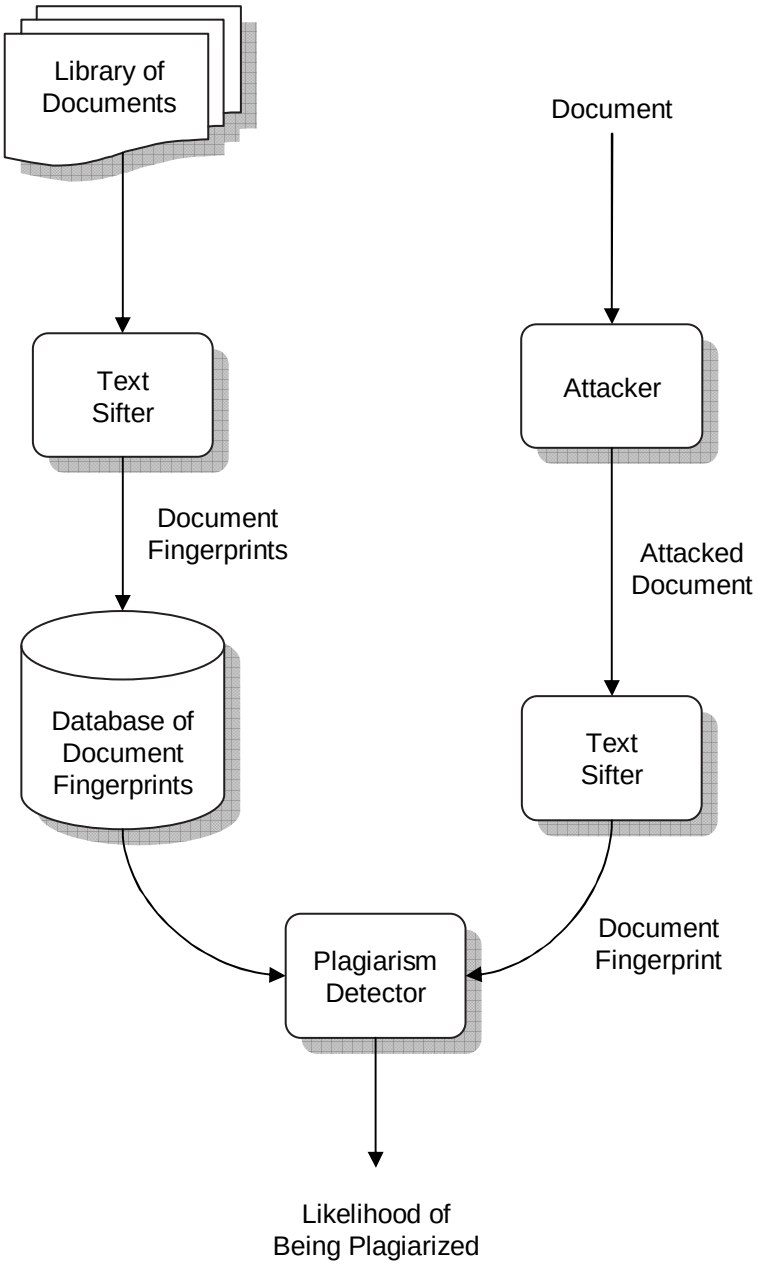


Figure 6.1: Model of a plagiarism detection system.

from an attacker, it generates the documents fingerprint and compares that fingerprint to those in the library. The output is the likelihood that the document was plagiarized.

### 6.0.1 Applications

**Plagiarism and Copy Detection:** The obvious application of text sifting is detecting plagiarism and copyright infringements. This not only means finding exact duplicates of documents, it also means finding close matches that have been changed to avoid copy detection mechanisms. Instead of combing through an entire database to see if a document has been plagiarized, the database might contain a text sifting fingerprint alongside each document. Searching for matches in the fingerprints is much faster than searching through the documents, and the attack-resistance of text sifting means that it is hard for an adversary to "disguise" documents. If a match is found among the fingerprints, further comparisons can be done on the full documents to avoid false matches.

An adversary can mount a plagiarism attack if it has access to a very large number of document fingerprints (see Section 6.2). Therefore, in copy detection systems which employ text sifting, the document fingerprints must be kept secret. Allowing unfettered public access to a copy detection system is also a security risk, as an attacker can iteratively modify a document to find the minimum modification necessary to produce a false negative. Heintze [20] discusses methods for allowing a text sifting system to be used by the public.

**Duplicate Email Detection:** Another application of text sifting is duplicate email detection. Email providers detect spam in part by counting the number of times the same email arrives at their servers. Spammers get around this by changing their email every time it is sent. It would be infeasible for an email service to fully compare every email that is received for every user to every other email that has been recently received, but text sifting can be used to speed up the process and notice the similarities between emails even after attacks by spammers.

Duplicate email detection is essentially the same as plagiarism detection, but optimized for email. Instead of operating on words as is done in plagiarism detection,

operations are on letters. Additionally, the preprocessor is much more aggressive, to take into account the tricks that spammers use. For example, the string "`|<`" (pipe – less-than) could be converted to the letter "`K`", while the letter "`I`" and the number "`1`" could be collapsed to the same token.

Note that a spammer is implicitly trying to perform a plagiarism attack by copying their own email and changing it enough that an email provider will not recognize it.

**Search Engines:** A sifting-enabled search engine would make it more difficult to modify web pages to gain higher page rankings. It would also enable searches for similar documents. Finally, it would make it more difficult to evade a search by changing a document.

### 6.0.2 Previous Work

There are several companies who offer the service of detecting plagiarism on the web, for example Glatt Plagiarism Services [1] and Plagiarism.org [2], but their services are proprietary and they do not reveal their methodology.

The field of natural language processing is devoted to analyzing text by understanding its meaning. See, for example, Manning and Schütze [32].

Stanford Digital Library's SCAM project by Shivakumar and Garcia-Molina [39–41] breaks documents into non-overlapping chunks, stores all of these chunks in a database, and analyzes documents based on the frequency of the chunks it contains. Storage size per document is relatively large.

Heintze [20] considers plagiarism detection. One suggested method is to sort the hash values of groups of characters and use the groups with the smallest hash values as a document fingerprint. They conclude, however, that this method produces too large a probability of false-positive matches, and so use letter frequency to decide which groups of characters are used in document fingerprints.

Broder et al. [6, 7] suggest two methods for selecting groups of words. The first method is to use min-hashing, the second is to select groups of words equal to 0 modulo a constant. These are the methods used in our sifting stage (see Section 6.1.3), but our cluster formation stage avoids the use of sliding windows (see Section 6.1.2). The

focus of their work is on duplicate document detection in order to find groups of similar documents on the web, not plagiarism detection against an intelligent adversary.

Schleimer et al. [38] present the idea of winnowing. They use a sliding window to generate local document fingerprints which can guarantee that short substring matches between documents are ignored, while long substring matches are always found. They also consider plagiarism detection.

## 6.1 Text Sifting

Text sifting consists of three phases. In practice these phases can be combined, but the algorithm is more easily explained as a three-pass process.

1. **Preprocessing:** The document is put into canonical form.
2. **Cluster Formation:** The canonical form is converted into a list of clusters.
3. **Sifting:** A subset of the clusters are chosen and output.

The first phase is common to most plagiarism detection systems. The second phase is unique to text sifting. The last phase is the same as the systems presented by Broder et al. [6, 7]. Figure 6.2 shows the algorithm graphically.

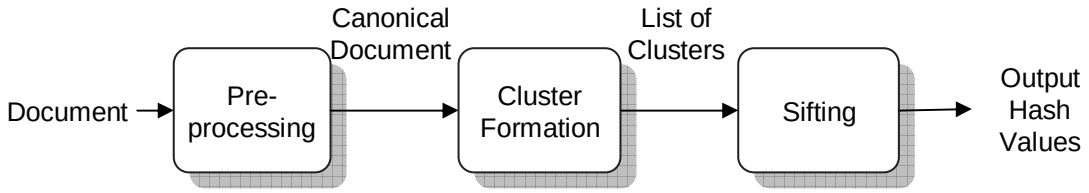


Figure 6.2: The text sifting algorithm.

Text sifting is slower than sliding window systems for two reasons. First, there is the overhead of the cluster formation stage, which is not present in sliding window systems. Second, when using a sliding window, an incremental hash function can

be used in the sifting stage. This is not possible with text sifting, so a slower, non-incremental hash function must be used.

We use the generic term *token*, which is the minimum text element upon which a text sifting program operates. Tokens can be words, letters, or groups of several letters, allowing flexibility in implementation. For plagiarism detection, tokens would be words. For duplicate email detection, tokens would be letters.

A *cluster* is list of tokens ordered as they appear in the document. Similar structures are referred to as *shingles* by Broder et al. [6,7], *substrings* by Heintze [20], and *k-grams* by Schleimer et al. [38]. These three structures are composed of contiguous tokens, while clusters are composed of tokens which are not necessarily contiguous in the document. Table 6.1 shows an example. At an abstract level clusters are lists of tokens, but in practice clusters are stored as hash values.

Many universal hash functions are called for in this section. All such functions are based on the secret key, $K$. In general, any universal hash functions will be sufficient, although faster hash functions are always preferred over slower hash functions. Universal hash functions are discussed by Cormen et al. [12]. Secure cryptographic hash functions are discussed, for example, by Stinson [42]. Broder et al. [7] use Rabin fingerprints [37].

### 6.1.1 Preprocessing Stage

The purpose of the preprocessing stage is to remove superfluous details that do not distinguish one document from another. First, filetype details are removed, for example by extracting the body text from PDF, Postscript, or HTML files.

Next, documents are put into canonical form. The specific form depends on the application. A plagiarism detection program would remove all characters that are not letters, numbers, or spaces, convert all white space into single spaces, and convert all letters to lowercase. A spam detection program might go further, removing all punctuation and white space and converting all characters and pairs of characters into canonical representations. For example, "`I`" and "`1`" could be converted to the same token, as could "`K`" and "`|<`".

At this point "common" words may be removed from the document. In plagiarism detection, this would include words like "a", "the", etc.

Next, all characters are converted into numbers. For plagiarism detection, this would mean assigning each token a value from 0 to 35, with 0 to 9 corresponding to the digits and 10-35 corresponding to the letters `a` to `z`. A permutation on all the possible values ($0 - 35$ in this case) is chosen pseudo-randomly based on the secret key, and all letters are permuted.

Finally, *token hashes* are generated by hashing the tokens with a universal hash function. The output of the preprocessing stage is a list of the tokens hashes.

More complex methods can be used, for example to convert a Postscript of Microsoft Word document to plain text, but they are beyond the scope of this work. For an example of a system that does this, see [20].

### 6.1.2 Cluster Formation Stage

The cluster formation stage is what distinguishes text sifting from other methods. The common method [6, 7, 20, 38] is to generate clusters (or their equivalent) with a sliding window, taking all blocks of contiguous tokens. Text sifting generates clusters of tokens which are not necessarily contiguous. The cluster formation stage takes as input a list of token hashes and outputs clusters of these token hashes.

We present three methods which form clusters while ignoring certain tokens: *random clustering*, *random skipping*, and *random partitioning*. They can be used independently or together. If random partitioning is used without random skipping, then it is used with a sliding window.

#### Random Clustering

The first method of cluster formation is random clustering. Define a sliding window of size $w$ and a number of tokens $r \leq w$. For each position of the sliding window, form clusters of all of the $\binom{w}{r}$ sets of $r$ tokens in the window. Several pairs $(w, r)$ can be used to generate clusters of different lengths. This method is simple to implement and produces a large variety of clusters, but is infeasible because even reasonable

values of $w$ and $r$ produce a prohibitive number of clusters. Random skipping and random partitioning generate a more tractable number of clusters.

### Random Skipping

The basic idea is to keep a cumulative hash of all the tokens in a cluster, and add new tokens to the cluster based on what that would do to the cumulative hash. If adding a given token would put the cumulative hash in the *add* category, then add the token to the cluster. If adding the token would put the cumulative hash in the *stop* category, then add the token to the cluster and stop adding more tokens. Otherwise, skip the token under consideration and move on to the next token in the list.

In more detail, let all token hashes be in the set $G$. Let $g(\cdot,\cdot)$ be a function that takes as input two elements of $G$ and outputs a single element of $G$. $g$ should have the property that if $a$ is known and $b$ is chosen uniformly at random, or vice-versa, then $g(a,b)$ is unpredictable. For example, $g(a,b)$ could be equal to $a \oplus b$ or $a + b \bmod p$, depending on $G$.

Let $h(\cdot)$ be a universal hash function that takes as input an element of $G$ and outputs an element in the set $H$. Let $A$ (meaning *add*) and $S$ (meaning *stop*) be subsets of $H$ such that $A$ and $S$ are disjoint.

For each token hash there will be a cluster that begins with that token hash. When a cluster is started, it contains only the single token hash that started it. It is assigned a cumulative hash $c \in G$ equal this token hash.

Now consider a partially-formed cluster of 1 or more token hashes. Let $c$ be the cumulative hash for this cluster. Let $t$ be the next token hash being considered for inclusion in the cluster. If $h(g(c,t)) \in S$ then add $t$ to the cluster and stop adding tokens. If $h(g(c,t)) \in A$ then add $t$ to the cluster and continue to the next token hash. Otherwise, just continue to the next token hash.

The size of $|A| + |S|$ relative to $|H|$ determines how many token hashes will tend to be skipped between accepted token hashes when forming a cluster. On average, $\frac{|H|}{|A|+|S|} - 1$ token hashes will be skipped before one is accepted. For example, when $|H| = 2(|A| + |S|)$, half of all token hashes are skipped, and there is an average of one token hash skipped between every two accepted token hashes.

If $|S|$ is close to $|A|$ then clusters will tend to be short, while if $|S|$ is small compared to $|A|$ clusters will tend to be long. With random input the expected number of tokens in a cluster is $1 + \frac{1}{x}$. $S$ can be varied as tokens are added to a cluster, for example to result in an equal probability of all cluster lengths from the minimum to the maximum. In practice, minimum and maximum cluster lengths, $l_{min}$ and $l_{max}$, are always set.

$A$ and $S$ may vary with the number of token values already in a cluster. For example, setting $|S| = 0$ when choosing tokens $2, 3, ..., l-1$ and setting $|A| = 0$ when choosing token $l$ ensures that all clusters will consist of exactly $l$ tokens. We have found experimentally that constant-length clusters are superior to variable-length clusters at detecting plagiarism. This is because clusters can only be compared if they are of equal length, so using having multiple lengths decreases the number of pairs which are compared.

For example, if there are $k$ clusters of one length and $k$ of another, then there are $\frac{1}{2}k(k-1)$ pairs of each length which can be compared, or $k(k-1)$ pairs total. On the other hand, if there are $2k$ clusters of the same length, then there are $2k(k-\frac{1}{2})$ pairs which can be compared, which is almost double the case with two lengths. In general, if there are $l$ different lengths with an equal number of clusters of each length, the number of pairs which are compared decreases by a factor of approximately $l$.

A variant is to allow each of the $n$ tokens to start $k$ different clusters, so the output of this stage will be approximately $kn$ clusters[1]. Instead of a single hash function $h$, there would be $k$ hash functions $h_1, h_2, ..., h_k$. This can increase the robustness of the output, but will slow down the cluster formation and sifting phases by a factor of $k$. This variant was experimentally found to not be as effective as combining random skipping with random partitioning, though.

The method of random skipping is less useful when the tokens are letters, because there is very little entropy in the hash values of each token. In this case, random clustering might prove useful.

Note that he parameters of the cluster formation stage are application-specific

[1]It will actually be slightly less because the tokens at the end of the document will not be able to produce clusters which meet the minimum length requirement.

and will be determined experimentally.

### Random Partitioning

In random partitioning, the cluster formation algorithm is run multiple times and each time a subset of token hashes is ignored. First the token hash values are partitioned into $b$ sets of approximately equal size using the secret key $K$. Next the cluster formation algorithm is run $b$ times, each time running as normal but ignoring all token hashes in one of the $b$ sets. If an attacker tries to change a document by adding, for example, the word, "very" in many different positions, at least 1 of the $b$ iterations will ignore the word "very", increasing the chance of recognizing the attack. We have found experimentally that $b = 2$ is optimal.

Consider the tokens in a document to be categorized as "good" or "bad". Good tokens are the ones which help to identify a document. Bad tokens are the ones caused by an adversary either replacing existing tokens or adding tokens. The ideal situation is one where all the bad tokens are ignored and no good tokens are ignored, so the resulting clusters each contain a wide variety of good tokens. Since it is unknown which tokens are good or bad, this leads to a tradeoff in cluster formation. If the set of ignored tokens is very large for an iteration of cluster formation, then most clusters formed from that iteration will contain mostly good tokens, but there will not be that much variety of tokens within clusters. If the set of ignored tokens is very small, there will be a variety of good tokens, but also many bad tokens as well.

If random partitioning is used alone, clusters are formed with a sliding window. The sliding window algorithm is used $b$ times, and each time one of the $b$ subsets of token hashes in ignored.

### Random Skipping with Random Partitioning

To combine random skipping with random partitioning, random partitioning is performed first, and for each of the $b$ iterations of the cluster formation algorithm, random skipping is performed. In practice this is done in parallel, so a document is only scanned a single time.

### 6.1.3 Sifting Stage

This stage takes as input the clusters from the cluster formation stage, hashes them to find the *cluster hashes*, and outputs a small subset of the cluster hashes. Although cluster formation and sifting are explained as two separate processes, in practice each cluster is sifted as it is generated.

There are two methods of sifting, *pure hashing* and *min-hashing*. Both methods use two universal hash functions with key $K$, $D_K(\cdot)$ and $H_K(\cdot)$. Both hash functions take as input a series of token hashes (representing a cluster) and output a single cluster hash.

#### Pure Hashing

For cluster $c_i$, $H_K(c_i)$ is output if $D_K(c_i) = 0 \bmod s$ for some fixed $s$. Afterwards, the clusters are sorted to remove duplicates and speed up later comparisons. Since each cluster is considered independently of all other clusters, a change in one cluster will not affect whether or not another cluster is output. $s$ is generally assumed to be constant, so the output will have length proportional to the length of the input document.

To limit the size of the output, a variant is to allow $s$ to change with the length of the document being sifted. Broder [6] suggests using $s = 2^j$ and using larger $j$ for larger documents ($j$ approximately proportional to the log of the input size). This constrains the size of the output, $m$, while still guaranteeing that a cluster output in a small document be output if it appears in a much larger document.

The size of the output is nondeterministic. For a random input of $n$ clusters with no duplicate clusters, and a constant $s$ for all documents, the output size has a mean of $\frac{n}{s}$ and a standard deviation of $\sqrt{\frac{n}{s} \cdot \frac{s-1}{s}}$.

#### Min-Hashing

For each cluster $c_i$, the hash value $C_i = D_K(\cdot)$ is found. The $C_i$ are sorted and duplicates are removed. For the $m$ $C_i$ with the smallest hash values, $H_K(c_i)$ is output. If there are fewer than $m$ unique cluster hashes then output all unique cluster hashes.

Note that the type of sort required for min-hashing is much faster than a standard sort, since only the smallest $m$ values are returned. A modified . With $n$ clusters and an output size of $m$ ($m \ll n'$), sort time is $O(n \log m + m \log m)$ as opposed to the $O(n \log n)$ of a standard sort.

$m$ can be constant or can depend on the length of the document. If $m$ is constant, it will be difficult to detect a small document that has been embedded inside a larger document; since cluster hash values are distributed uniformly, there is a good chance that the larger document will have many clusters with smaller hash values than those in the smaller document, so many of the clusters which are in the smaller document will be pushed off the end of the sorted list. Allowing $m$ to grow with document size increases the number of clusters in the small document that remain on the sorted list for the large document. The downside of this, of course, is increased storage space to store the extra clusters.

Changing one part of a document can affect the output of a cluster in another part of the document. For example, if cluster $x$ has the $m^{th}$ smallest hash value in a document, and a cluster $y$ is added to the document such that the $H_k(y) < H_k(x)$, then $x$ will be removed from the output and $y$ will be added. This will occur even if $x$ and $y$ are far apart in the document.

### Output Size

The optimal output size is specific to the application. It will be different for a full-web search than for a database of class essays, for example. Since attacks are detected randomly, increasing the output size will help to detect attacks. With pure hashing, the output size is either approximately linear in the size of the input or approximately constant. With min-hashing, the output size can vary arbitrarily with the size of the input, although constant, linear, or sublinear (for example, square root) relationships are best.

A constant-length output performs best when comparing documents of similar sizes. When comparing documents of different sizes, a constant output size must be carefully chosen. If the output is too large, space will be wasted when sifting small files. If the output is too small, fingerprints for large documents will not be very

representative of the document. Heintze [20] and Broder [7] show that a constant-length output can achieve good results at identifying documents in the absence of attacks.

When documents of radically different sizes are to be compared, a linear output size can be useful, especially in plagiarism detection. A common plagiarism attack is to insert a small file into a large file. If the output size is constant, the fingerprint of the large file with the plagiarism might not have many matches with the original, small file. A linear output size will make it more likely that this sort of attack will be detected. The disadvantage of using linear-size outputs is that the total text sifting overhead will tend to be much larger than for systems with constant output size.

### 6.1.4 Comparing Documents

In a text sifting system, documents are compared by comparing their fingerprints. The basis of comparison is the number of cluster hashes that the two fingerprints have in common. However, this number needs to be normalized to convey much meaning.

The most obvious normalization is to divide the number of shared cluster hashes with the total number of cluster hashes in both documents. If $C_1$ is the set of cluster hashes from document 1, and $C_2$ is the set of cluster hashes from document 2, then the first similarity measure is

$$S_1(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

This measure intuitively captures the idea that two documents are similar if they share many clusters in common, but does not do well in the situation where one fingerprint is much larger than the other. For example, it is possible to have a small $S_1$ even though $C_1 \subset C_2$, if $|C_2| \gg |C_1|$.

A similarity measure which avoids this problem is

$$S_2(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1|}$$

Broder [6] refers to this as an estimate of the *containment* of document 1 in document 2 when using pure hashing. This follows the intuition that document 1 is contained in document 2 if a large fraction of the clusters in document 1 appear in document 2. Note that $S_2(C_1, C_2)$ and $S_2(C_2, C_1)$ are not necessarily equal.

Finally, we present a third similarity measure, which is the maximum of $S_2(C_1, C_2)$ and $S_2(C_2, C_1)$.

$$S_3(C_1, C_2) = max\left(\frac{|C_1 \cap C_2|}{|C_1|}, \frac{|C_1 \cap C_2|}{|C_2|}\right)$$

If the fingerprint of either document is a subset of the other document's fingerprint, then $S_3$ will be equal to 1. If $S_1$ is large then $S_3$ will also be large, but $S_3$ may be large even when $S_1$ is small.

$$S_3(C_1, C_2) \geq S_1(C_1, C_2)$$

We prefer $S_3$ as a similarity measure for plagiarism detection.

Note that comparing document fingerprints is very fast. Comparing two sorted lists of hash values is very fast, taking time linear in the lengths of the lists.

## 6.2 Security of Text Sifting

Broder et al. [7] and Heintze [20] have demonstrated that these methods work well in the absence of adversaries. We now consider the robustness of text sifting functions against attacks. First we consider the theoretical performance of text sifting against several categories of attacks, and then describe the experimental results of some of these attacks.

Certain attacks are beyond the scope of text sifting systems. One such class of attacks are attacks which do not leave documents in human-readable form. For example, compression and encryption are not considered to be valid attacks. Attacks which do not produce text files are also not considered, so we ignore, for example, attacks which convert documents into images. Finally, attacks which change all the

words in a document are ignored. This could include translating a document to another language, or writing a new document that conveys similar information to the original document. The attacks we consider are those for which a human, reading both the original document and the attacked document, could reasonably conclude that the second document is a plagiarism of the first.

Text sifting is not necessarily secure if an adversary has direct access to a very large number of document fingerprints. Such an adversary could conceivably discover a small set of clusters which are output by the text sifting program and insert those clusters into documents. This would not necessarily destroy the security of a text sifting system, but could decrease its effectiveness. Therefore, it is assumed that adversaries do not have direct access to document fingerprints. Of course, an adversary with knowledge of the secret key can mount plagiarism attacks and completely break the system with minimal changes to documents.

A text sifting oracle does not reveal document fingerprints, but responds that a given document is an original or a plagiarism. An adversary with access to such an oracle is also a threat, because they can iteratively modify a document and find the minimum modification necessary to produce a false negative. This is not a problem if all users of the oracle are trusted. Heintze [20] discusses methods for allowing public access to such an oracle.

An unaided adversary is one without access to document fingerprints, a text sifting oracle, or the secret key. An unaided adversary can only mount a successful plagiarism attack by making many changes in a document and hoping that the document fingerprint is significantly changed. In the cluster formation phase, an adversary can not predict the clusters which will be output and can only affect the clusters by changing many tokens. An adversary can make a large number of changes in a small region and have a high probability of changing the clusters from that region, but there is no way for the adversary to predict which clusters will be output by the sifting phase, so changes from the small region will probably not significantly change the document fingerprint. The only way to significantly change an entire document fingerprint is to change many tokens in all regions of the document.

A typical text sifting system will choose a similarity measure (usually $S_3$) and a

threshold, and will decide that one of a pair of documents is a copy if the two have a similarity greater than the threshold. When an adversary attacks a document, one way to measure the distortion introduced by the attack is to consider the average similarity, taken over all keys, of the original document fingerprint and the attacked document fingerprint. We refer to this average similarity as the *attack similarity*. The intuition is that two documents with large attack similarity are likely to be very similar to human readers of the documents, for example for detecting plagiarism or copyright violations.

We consider an attack to be successful if the attack similarity is greater than the threshold but the similarity measured by the system is less than the threshold. An attack may result in an attack similarity that is below the threshold, but we do not consider this to be a successful attack because the attacker was forced to distort the document to the point where the attacked document no longer resembles the original document enough to constitute a copy.

The measured similarity is likely to be close to the attack similarity. This is made even more likely with the use of large document fingerprints. Importantly, an adversary can not know if the measured similarity will be higher or lower than the attack similarity. It is therefore difficult for an adversary to launch a successful attack; the attack must be strong enough that the attack similarity is above the threshold, yet the attacker must be lucky enough to have the measured similarity fall below the threshold.

From the perspective of an unaided adversary, the sifting stage of a text sifting system essentially performs a random selection on all clusters from the cluster formation stage. For both pure hashing and min-hashing, if an attack causes $c$ percent of clusters from the cluster formation phase to change, then we expect $c$ percent of the cluster hashes in the document fingerprint to change as well. We will therefore consider the effects of various attacks by looking at how those attacks affect the clusters produced by the cluster formation stage.

### 6.2.1 Cosmetic Attacks

*Cosmetic attacks* are attacks that are eliminated in the preprocessing phase. When tokens are words, cosmetic attacks include changing capitalization, changing formatting, or adding spaces and extra punctuation. When tokens are letters, cosmetic attacks include changing capitalization, inserting punctuation, and substituting symbols for characters (for example, "| <" for "k" and "1" for "I"). Cosmetic attacks do not change the output of a text sifting function at all.

### 6.2.2 Scrambling Attacks

*Scrambling attacks* are attacks which add, delete, or change individual tokens. For example, when words are used as tokens, scrambling attacks correspond to adding words, deleting words, and changing words. Changing a word can mean anything from changing one letter in the word to substituting a new word; in either case, the token hash for the changed word will be completely different.

We will now consider, for several different cluster formation methods, the effects of scrambling attacks.

**Sliding Window:** Assume a sliding window of length $w$. Adding a token removes $w - 1$ old clusters and adds $w$ new clusters. Deleting a token removes $w$ old clusters and adds $w - 1$ new clusters. Changing a token removes $w$ old clusters and adds $w$ new clusters. Note that all these attacks all have approximately the same effect on the clusters that are output.

If an attacker performs a scrambling attack once every $w$ tokens, they can virtually ensure that the sliding window never sees any of the same clusters it saw in the original document, and can therefore (barring hash collisions) ensure that every hash value in the output is different.

The exception to this occurs if an attack accidentally causes a cluster from the original document to appear in the attacked document. These accidents can occur for all three scrambling attacks. For example, when adding tokens, a token can be added next to an identical token, or can be added in such a way that one of the $w$ new clusters is the same as a different cluster in the original document. Similar

coincidences can occur when removing and changing tokens. The probability of such an occurrence is low, but after many attacks on many documents it does occur, as was shown in our experiments.

**Random Skipping:** Assume that all clusters produced by random skipping have a fixed length $l$, since fixed-length clusters were found experimentally to be better than variable-length clusters. Then, ignoring edge effects at the end of the document, a document with $n$ words will have $n$ clusters. Each cluster includes the head token and $l-1$ other tokens, for a total of $n(l-1)$ non-head tokens in clusters. Therefore, each token will be at the head of one cluster and will be in an average of $l-1$ other clusters.

Adding a token causes one cluster to be added, the cluster that starts with that token. No clusters will be removed, but clusters will change if they select this new token, so an average of $l-1$ clusters will change.

Deleting a token causes one cluster to be removed, the cluster that started with that token. Clusters will change if they formerly held the deleted token, so an average of $l-1$ clusters will change.

Changing a token is equivalent to deleting the token and adding a new one in its place. It does not cause any clusters to be added or removed, but it guarantees that the cluster that it heads will change. An average of $2(l-1)$ additional clusters will also be changed, for an average of $2l-1$ total clusters changed. Note that changing tokens has approximately twice the effect of adding or deleting tokens.

**Random Partitioning:** If a token is added, $\frac{1}{b}$ of the clusters will ignore it. If a token is removed, $\frac{1}{b}$ of the clusters were ignoring it anyway and will be unaffected. For both adding and removing tokens, the rest the clusters are affected according to the cluster formation mechanism being used, sliding window or random skipping, but the effect is multiplied by $b-1$. For example, if using random partitioning with a sliding window, adding a token will result in the removal of $(w-1)(b-1)$ old clusters and the addition of $w(b-1)$ new clusters.

If a token is changed, there are two possibilities. If the replacement token is in the same partition as the original token, then as before the effect depends on the cluster formation mechanism being used but is multiplied by $b-1$. If the replacement token

and the original token are in separate partitions, the effect is the same, but multiplied by $b$. Since we use $b = 2$, at least half of the time a changing attack will cause twice the normal effect. Random partitioning, like random skipping, is affected more when tokens are changed than when tokens are added or removed.

### 6.2.3 Large-Scale Attacks

Large-scale attacks are attacks where large numbers of contiguous tokens are manipulated. Attacks are considered to be large-scale when the edge-effects of the attacks are small compared to the main effects. We will discuss large-scale additions of tokens, large-scale deletion of tokens, and large-scale rearranging of tokens.

**Large-Scale Additions:** When a large amount of text is inserted into a location in the middle of a document, it will disrupt the clusters that used to span the location, and will generate new clusters that cross the boundary between the original tokens and the inserted tokens. We assume that these edge effects are small relative to the effects of the large number of new tokens in the document.

We now consider the non-edge effects of a large-scale addition. With pure hashing, all of the cluster hashes from the original document will be present, but there will be a large number of new cluster hashes in the document fingerprint as well. Ignoring edge effects, the $S_3$ similarity from pure hashing will still be 1.

When min-hashing with a constant output size, some of the cluster hashes from the original document will be forced out by cluster hashes from the added tokens. The fraction of original cluster hashes is expected to be equal to the size of the original document as a fraction of the size of the new, expanded document, for both pure hashing and min-hashing. For example, if the new tokens are double the size of the original document, then we expect $\frac{1}{3}$ of the cluster hashes in the document fingerprint to come from the original document.

**Large-Scale Deletions:** Similar to the case of large-scale additions, we assume that edge effects are small relative to the effects of the deletion of a large number of tokens.

With pure hashing, deleting a large number of contiguous tokens will result in

the clusters which correspond to those tokens being removed. These non-edge effects mean that if a fraction $f$ of a document is deleted, only $1 - f$ of the original cluster hashes will remain in the document fingerprint. However, ignoring edge effects, the $S_3$ similarity will still be 1, because the attacked document will be contained within the original document.

With a constant-sized min-hash, $1 - f$ of the original cluster hashes will remain in the document fingerprint, and the rest will be replaced by new cluster hashes which come from the remaining portion of the text.

**Large-Scale Rearranging:** In the previous two cases we ignored edge effects. However, in the case of large-scale rearranging, edge effects are the only effects. If a large region of a document is moved to another spot in the same document, the only clusters that will change are those along the old and new boundaries of the region. Therefore, a small number of large-scale rearranging attacks will have relatively little effect on the document fingerprint with either pure hashing or min-hashing.

### 6.2.4 Experimental Results

As was mentioned before, Broder et al. [7] and Heintze [20] have demonstrated that these methods work well in the absence of adversaries. We now consider the robustness of text sifting functions against simple attacks.

#### Experimental Setup

Attacks were tested on a library of 100 articles from Reuters [3], approximately uniformly distributed in size from 1 kB to 6 kB. These articles were randomly chosen from a set of 9000 articles from Reuters. Both the attacks and the text sifting system were run in MATLAB [33]. MD5 was used as a hash function because a MATLAB implementation by Suter [43] was readily available, but it would generally be too slow to use in a production system. The results of the tests are shown in Table 6.2.

Words were used as tokens. We found that with random skipping it is best to have fixed-length clusters, so all clusters were formed of 10 tokens, both in sliding window tests and random skipping tests. When performing random skipping, clusters were

accepted with a probability of 0.3, as this gave the best performance. We found that the optimal number of partitions for random partitioning was 2. The modulus for pure hashing was 10, so 1 out of every 10 clusters was output. For min-hashing, a constant 100 clusters were requested, although some documents sometimes fell slightly short of this.

### Attacks

We tested eight attacks in two suites. The first suite was "intelligent" attacks and the second suite was random attacks. Both suites consisted of adding attacks, deleting attacks, changing attacks, and combination attacks.

The intelligent attacks consisted of attacking the tokens in positions $9, 19, 29, \ldots$. This ensured that every window of size 10 contained a single change. For the adding attack this meant adding tokens; for the deleting attack this meant deleting tokens; for the changing attack this meant changing tokens; and for the combination attack this meant alternating between adding, deleting, and changing tokens.

To make the tests equivalent, we made exactly as many random attacks of each type as there were intelligent attacks. For each document, if the document consisted of $n$ tokens, the random adding attack added $0.1n$ random tokens to random locations in the document; the random deleting attack removed $0.1n$ tokens at random; the random changing attack changed a random set of $0.1n$ of the tokens into different, randomly-chosen tokens; and the random combination attack performed all three attacks on approximately $0.035n$ tokens each.

### Results

Table 6.2 shows the results of the tests. The tests demonstrate one instantiation of a text sifting system, with a randomly-chosen key. We attacked all 100 documents in our library with the eight attacks and computed the $S_1$ and $S_3$ similarities between the original and attacked versions of each document for four different text sifting systems. The values in the table are the average similarities over all 100 documents.

The table shows that random skipping and random partitioning both offer significant advantages against the intelligent attacks. The intelligent plagiarism attacks succeed against a sliding window, but fail against systems using random skipping, random partitioning, or both. Against random attacks, the record is mixed. Random skipping and random partitioning do slightly better than the sliding window against adding and deleting attacks, but perform worse on changing attacks, and by extension combination attacks, as explained in Section 6.2.2.

The data also illustrates the extra susceptibility of random skipping and random partitioning to changing attacks, as opposed to adding and deleting attacks. The combination attacks, which include all three scrambling attacks, lie somewhere in the middle.

Pure hashing generally performed better than min-hashing, which is surprising because the documents were small enough that only a quarter had fingerprints with size larger than 100, which was the size of fingerprints produced by min-hashing. What seems to happen is that on smaller documents min-hashing wastes space by storing too many clusters, while on longer documents min-hashing doesn't store enough clusters.

One interesting thing to note is that changing only 10% of the tokens in a document lowers the maximum similarity measurement to approximately 0.4. This happens because we used a large number of tokens per cluster. With long clusters, a single change in a document has the potential to affect many clusters. Using shorter clusters will help; using a length of 6 raised the maximum similarity measurement to approximately 0.65. However, decreasing the cluster length increases the number of false positives, so the length that is ultimately chosen depends on the needs of the specific system. A small window might work well with a database of papers, but will result in too many false positives in a full search of the internet.

Note also that the sliding window does not always result in similarities of zero when up against the intelligent attacks. This is the result of a small number of coincidences as explained in Section 6.2.2. They did not occur often, but they did occur occasionally, and that was enough to raise some of the averages above zero. However, those attacks only produced very small similarities (caused by one cluster

matching) in a handful of documents over all the tests we performed.

**False Positives**

In our small library of 100 articles, for all three similarity measures in Section 6.1.4, all pairs of different documents had similarities of zero. Of course, all documents had similarities of 1 with themselves. Occasionally in our tests (although not among our 100 articles), we found unrelated documents that shared a single cluster, but that occurrence was very rare. An $S_3$ similarity of even 0.1 generally arises only from plagiarism attacks, and an $S_3$ similarity of 0.2 almost certainly implies a plagiarism attack.

In a larger library, say all web pages on the internet, there is a significant probability of false positives. It is conceivable that two unrelated documents could end up with similar fingerprints, especially if the documents are short and have small fingerprints, and if they contain a lot of generic or boilerplate text. Having a large number of tokens per cluster and a large number of clusters per fingerprint will significantly decrease the probability of false positives. Both Broder et al. [7] and Heintze [20] address methods for reducing false positives.

A second source of false positives is attack-induced false positives. We do not consider these to be a problem because a plagiarist who induces a false positive has lost.

## 6.3 Conclusions

We presented a system that can withstand a class of plagiarism attacks which are very successful against similar previous systems. Our system achieves this resilience by eschewing the use of sliding windows, instead using cluster formation methods which can not easily be exploited by attackers.

Our system also performs well against random attacks, although slightly worse than sliding window systems. A hybrid text sifting system could generate half of the fingerprint with a sliding window and half with random skipping and/or random partitioning. Compared to straight text sifting systems with the same fingerprint

size, this would result in increased performance against random attacks, especially changing and combination attacks, but would result in reduced performance against intelligent attacks.

The document "quick brown fox jumps" contains four tokens: "quick", "brown", "fox", and "jumps". The following are all possible clusters from this short document:

| **1-token clusters** | **2-token clusters** | **3-token clusters** | **4-token clusters** |
|---|---|---|---|
| (quick) | (quick brown) | (quick brown fox) | (quick brown fox jumps) |
| (brown) | (quick fox) | (quick brown jumps) | |
| (fox) | (quick jumps) | (quick fox jumps) | |
| (jumps) | (brown fox) | (brown fox jumps) | |
| | (brown jumps) | | |
| | (fox jumps) | | |

Table 6.1: All possible clusters are shown for a very short document.

| Pure Hashing | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Slid. Win. | | Rand. Skip. | | Slid. Win. & Rand. Part. | | Rand. Skip. & Rand. Part. | |
| | $S_1$ | $S_3$ | $S_1$ | $S_3$ | $S_1$ | $S_3$ | $S_1$ | $S_3$ |
| Intelligent adding attack | 0 | 0 | 0.184 | 0.339 | 0.152 | 0.287 | 0.182 | 0.332 |
| Intelligent deleting attack | 0.00114 | 0.00264 | 0.187 | 0.354 | 0.157 | 0.296 | 0.190 | 0.346 |
| Intelligent changing attack | 0.000676 | 0.00148 | 0.0557 | 0.115 | 0.0482 | 0.0969 | 0.064 | 0.129 |
| Intelligent combination attack | 0 | 0 | 0.115 | 0.228 | 0.0914 | 0.180 | 0.109 | 0.211 |
| Random adding attack | 0.242 | 0.422 | 0.248 | 0.431 | 0.245 | 0.427 | 0.238 | 0.415 |
| Random deleting attack | 0.205 | 0.379 | 0.216 | 0.389 | 0.205 | 0.375 | 0.218 | 0.387 |
| Random changing attack | 0.194 | 0.347 | 0.109 | 0.213 | 0.127 | 0.243 | 0.106 | 0.203 |
| Random combination attack | 0.205 | 0.370 | 0.151 | 0.287 | 0.174 | 0.317 | 0.155 | 0.285 |
| Min-Hashing | | | | | | | | |
| | Slid. Win. | | Rand. Skip. | | Slid. Win. & Rand. Part. | | Rand. Skip. & Rand. Part. | |
| | $S_1$ | $S_3$ | $S_1$ | $S_3$ | $S_1$ | $S_3$ | $S_1$ | $S_3$ |
| Intelligent adding attack | 0 | 0 | 0.168 | 0.290 | 0.139 | 0.243 | 0.176 | 0.298 |
| Intelligent deleting attack | 0.000513 | 0.00102 | 0.176 | 0.302 | 0.148 | 0.258 | 0.171 | 0.292 |
| Intelligent changing attack | 0.000251 | 0.000500 | 0.0557 | 0.105 | 0.0403 | 0.0768 | 0.0550 | 0.104 |
| Intelligent combination attack | 0.000456 | 0.000901 | 0.106 | 0.191 | 0.0907 | 0.165 | 0.106 | 0.190 |
| Random adding attack | 0.222 | 0.362 | 0.226 | 0.370 | 0.232 | 0.375 | 0.223 | 0.364 |
| Random deleting attack | 0.200 | 0.3322 | 0.213 | 0.355 | 0.205 | 0.342 | 0.206 | 0.339 |
| Random changing attack | 0.213 | 0.349 | 0.104 | 0.188 | 0.132 | 0.230 | 0.110 | 0.196 |
| Random combination attack | 0.178 | 0.300 | 0.147 | 0.254 | 0.155 | 0.266 | 0.141 | 0.245 |

Table 6.2: The effects of scrambling attacks on text sifting.

# Bibliography


[1] Glatt plagiarism services. http://www.plagiarism.com.

[2] Plagiarism.org. http://www.plagiarism.org.

[3] Reuters articles. http://wim.fzi.de:8080/Reuters/Reuters.zip, from http://km.aifb.uni-karlsruhe.de/kaon2/frontpage, no longer available.

[4] A. Adelsbach and A. . Sadeghi. Zero-knowledge watermark detection and proof of ownership. In *Information Hiding Workshop*, 2000.

[5] D. Boneh, E.-J. Goh, and A. Nisgav. Chosen ciphertext secure (IND-CCA1) homomorphic encryption schemes. Unpublished manuscript, 2004.

[6] A. Broder. On the resemblance and containment of documents. In *SEQS: Sequences '91*, 1998.

[7] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the web. In *6th International World Wide Web Conference*, 1997.

[8] D. Brumley and D. Boneh. Remote timing attacks are practical. In *12th USENIX Security Symposium*, 2003.

[9] E. Candès and D. Donoho. *Curvelets: A Surprisingly Effective Nonadaptive Representation of Objects with Edges*. Saint-Malo: Vanderbilt University Press, 1999.

[10] D. Catalano, R. Gennaro, N. Howgrave-Graham, and P. Q. Nguyen. Paillier's cryptosystem revisited. In *CCS '01: Proceedings of the 8th ACM conference on*

*Computer and Communications Security*, pages 206–214, New York, NY, USA, 2001. ACM Press.

[11] B. Chen and G. W. Wornell. Quantization index modulation: A class of provably good methods for digital watermarking and information embedding. *IEEE Trans. on Information Theory*, 47(4):1423–1443, May 2001.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[13] I. J. Cox and J.-P. M. G. Linnartz. Public watermarks and resistance to tampering. In *International Conference on Image Processing (ICIP'97)*, pages 26–29, 1997.

[14] I. Damgard and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Proc. of Public Key Cryptography*, 2001.

[15] J. Eggers, J. Su, and B. Girod. Asymmetric watermarking schemes. In *Sicherheit in Mediendaten, GMD Jahrestagung*, 2000.

[16] J. Fridrich. Key-dependent random image transforms and their applications in image watermarking. *International Conference on Imaging Science, Systems, and Technology*, pages pp. 237–243, 1999.

[17] J. Fridrich. Visual hash for oblivious watermarking. In *Proc. SPIE Photonic West Electronic Imaging*, 2000.

[18] S. Goldwasser and S.Micali. Probabilistic encryption. In *Journal of Computer and Systems Science*, volume 28, pages 270–299. 1984.

[19] G. Hachez and J.-J. Quisquater. Which directions for asymmetric watermarking? In *XI European Signal Processing Conference*, 2002.

[20] N. Heintze. Scalable document fingerprinting. In *Second USENIX Electronic Commerce Workshop*, pages 191–200, 1996.

[21] N. Johnson, Z. Duric, , and S. Jajodia. On "fingerprinting" images for recognition. In *5th Int'l. Workshop on Multimedia Information Systems*, 1999.

[22] T. Kalker. Secure watermark detection. In *43rd Annual Allerton Conference on Communication, Control, and Computing*, 2005.

[23] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer Verlag, 1994.

[24] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis: Leaking secrets. In *Crypto*. Springer Verlag, 1999.

[25] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Lecture Notes in Computer Science*, 1109:104–113, 1996.

[26] Q. Li and E. Chang. Security of public watermarking schemes for binary sequences. In *Information Hiding Workshop*, 2002.

[27] M. Malkin and T. Kalker. A cryptographic method for secure watermark detection. In *Information Hiding Workshop*, 2006.

[28] M. Malkin and R. Venkatesan. Comparison of texts streams in the presence of mild adversaries. In *Australian Information Security Workshop*, 2005.

[29] M. Malkin and R. Venkatesan. Robust image watermarking with the randlet transform. In *43rd Annual Allerton Conference on Communications, Control, and Computing*, 2005.

[30] M. Malkin and R. Venkatessan. The randlet transform: Applications to universal perceptual hashing and image identification. In *42nd Annual Allerton Conference on Communications, Control, and Computing*, 2004.

[31] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[32] C. Manning and H. Schtze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

[33] MathWorks. Matlab. http://www.mathworks.com/.

[34] M. K. Mihcak and R. Venkatesan. New iterative geometric methods for robust perceptual image hashing. In *Digital Rights Management Workshop*, 2001.

[35] M. K. Mihcak, R. Venkatesan, and M. Kesal. Watermarking via optimization algorithms for quantizing randomized statistics of image regions. *Proceedings of the 40th annual Allerton Conference on Communications, Computing and Control*, 2002.

[36] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of Eurocrypt '99*, volume 1592, pages 223–238. Springer-Verlag, 1999.

[37] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard University, 1981.

[38] S. Schleimer, D. Wilkerson, and A. Aiken. Winnowing: Local algoriothms for document fingerprinting. In *SIGMOD 2004*, San Diego, CA, 2003.

[39] N. Shivakumar. *Detecting Digital Copyright Violations on the Internet*. PhD thesis, Stanford University, 1999. http://www-db.stanford.edu/ shiva/.

[40] N. Shivakumar and H. Garcia-Molina. Scam: A copy detection mechanism for digital documents. In *2nd International Conference in Theory and Practice of Digital Libraries*, Austin, Texas, 1995.

[41] N. Shivakumar and H. Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *1st ACM Conference on Digital Libraries*, Bethesda, Maryland, 1996.

[42] D. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

[43] H.-P. Suter. Matlab md5 implementation. http://www.mathworks.com/matlabcentral/fileexchange.

[44] R. Venkatesan, S. Koon, M. Jakubowski, and P. Moulin. Robust image hashing. In *Proceedings of IEEE International Conference on Image Processing*, volume 3, pages 664–666, 2000.

[45] R. M. F. i. Ventura, P. Vandergheynst, and P. Frossard. Highly flexible image coding using non-linear representations. In *Proceedings of VCIP*, 2003.

[46] I. Venturini. Counteracting oracle attacks. In *Proceedings of the 2004 Workshop on Multimedia and Security*, pages 187–192. ACM Press, 2004.