Content-based Video Signatures based on Projections of Difference Images

Papadopoulos Chrysanthi

April 2015

Dissertation submitted in partial fulfilment for the degree of Master of Science in Communication & Information Systems

Department of Informatics & Communications
TEI of Central Macedonia
# Table of Contents

Table of Contents ........................................................................................................ 2  
Authentication .............................................................................................................. 3  
Video tampering ......................................................................................................... 4  
  Authentication Techniques .......................................................................................... 4  
  Classification of authentication Techniques ............................................................... 5  
  Multimedia Authentication sources: Raw Data v.s. Compresses Data .................... 7  
Digital Signatures and Watermarking ........................................................................ 8  
  What is a Digital Signature .......................................................................................... 8  
  How do digital Signatures Work ................................................................................ 9  
  Hash Based Digital Signatures .................................................................................... 11  
    Lamport–Diffie one-time signature scheme .............................................................. 12  
    Winternitz one-time signature scheme .................................................................... 13  
  What is Digital Watermarking .................................................................................... 15  
  Why Digital Watermarking? ...................................................................................... 15  
  Watermarking v.s. Digital Signature .......................................................................... 16  
Robust Hash Functions ................................................................................................ 18  
Our Approach .............................................................................................................. 19  
  Video feature extraction ............................................................................................. 19  
  Robust Hash ............................................................................................................... 21  
The signature sensitivity .............................................................................................. 23  
Our tests ...................................................................................................................... 24  
  Comparing videos with brightness modification and completely different videos .... 25  
  Comparing videos after noise modification and completely different videos ........ 26  
  Comparing videos after rotation and completely different videos ......................... 28  
Matlab Code ................................................................................................................. 30  
References .................................................................................................................. 47
Authentication

By definition, authentication is a process that provides some means to guarantee that the entities are who they claim to be (i.e. entity authentication), or that the information has not been manipulated by unauthorized parties (i.e. data authentication) (Menezes, 1997). In the context of multimedia, video authentication refers to a process that ascertains the integrity of a given digital video and detects if it has been tampered in any way. In other words, a video authentication system ensures the integrity of a digital video, and verifies that the video taken into use has not been corrupted.

As technology continues to invade in our everyday life a lot of issues appear, one of those issues is video authenticity. With the rapid innovation and development in digital technologies, video applications are infiltrating into our daily lives in breakneck speed from traditional television broadcasting to modern vulnerable communication media such as Internet/Intranet, wireless communication and consumer products such as VCD/DV.

Digital video applications have a lot of advantages compared to the custom analog video, better picture quality, sharper images and better colour reproduction, are only some of them. Moreover a video can be easily transported over the Internet and it facilitates easy editing and copying.

There are a lot of cases in our everyday life where video authentication seems to be necessary. Some of those are the following:

- A video may present a widely known person being involved in illegal activities. In this case it is of major interest to be able to distinguish whether or not the video was altered or not.

- A criminal can be set free because the video showing their crime cannot be proved conclusively in the court of law.
Video tampering

When content of information, being produced by a given video sequence, is maliciously altered, then it is called tampering of video data. It can be done for several purposes, for instance to manipulate the integrity of an individual. Since a wide range of sophisticated and low cost video editing software are available in the market that makes it easy to manipulate the video content information maliciously, it projects serious challenges to researchers to be solved.

Authentication Techniques

A video authentication system ensures the integrity of digital video and verifies whether the video has been tampered or not. But in most cases, especially in the court of law, it may be more beneficial if the authentication system can tell where the tampering happens and how the video is tampered.

A typical video authentication system is shown in figure 1. For a given video, authentication process starts with feature extraction. After that, with a specific video authentication algorithm, the authenticated data $H$ is generated using the feature $f$ of the video. This authentication data $H$ is encrypted and packaged with the video as a signature or alternatively it can be embedded into the video content as a watermark. The video integrity is verified by computing new authentication data $H$ for the given video. The new authentication data $H$ is compared with decrypted original authentication data $H$. if both match, the video is treated as authentic else it is considered as tampered video.

In order for a video authentication system, to be effective, it must support the properties such as sensitivity to malicious alteration, localization and self recovery of altered regions, robustness to normal video processing operations, tolerance against some loss of information, compactness of authentication data, sensitivity against false intimation and computational feasibility. In fact in addition to having robustness against benign operations, an ideal video authentication system must make a given video resistant against all possible attacks and must verify whether a given video is tampered or not. Benign operations are those video processing operations that do not modify its content semantically such as geometric transformations, image enhancements and compression. Once the verification is done for the given video, it would be useful to find where and how the tampering has been done.
In the past few years, watermarking and digital signature have been widely used for the purpose of video authentication. Different techniques have their own advantages and disadvantages. In fact fragile watermarking and digital signatures are the two commonly used schemes for authentication.

**Classification of authentication Techniques**

Multimedia authentication technique can be classified into two categories: *complete* authentication and *content* authentication.

*Complete* authentication refers to techniques that consider the whole piece of multimedia data and do not allow any manipulations or transformations. Early works of multimedia authentication were mostly in this category. Because the non-manipulated data are like messages, many existing message authentication techniques can be directly applied. For instance, digital signatures can be placed in the LSB (Least Significant Bit) of uncompressed data, or the header of compressed data. Then, manipulations will be detected because the hash values of the altered message bits will not match the information in the digital signature. In practice, fragile watermarks or traditional digital signatures may be used for complete authentication.

*Content* authentication refers to a different objective that is unique for multimedia data. The meaning of multimedia data is based on their content instead of the bitstreams. In some applications, manipulations on the bitstreams without changing the meaning of content are considered as acceptable. Compression is an example. Today most digital multimedia data are sired or distributed in compressed forms. To satisfy various needs of broadcasting, storage and transmission, transcoding of compressed digital videos may also be required. For instance digital video clips are usually shot and stored in the compressed format with a pre-
determined bitrate, but distributed with a different bitrate in transmission. Transcoding processes change the pixel values of the digital video but not its content, therefore, videos that are obtained by transcoding the original one should be considered as authentic.

With complete verification we have to verify the data at every transmission stage and trust all the internal entities. However, with content verification, we can transmit the robust signature with the data and only verify it at the last stage (Figure 2). Therefore, we do not need to verify the data at each stage and question the trustworthiness of the intermediate people.

A broad meaning of content authentication is to authenticate multimedia content on a semantic level even though manipulations may be perceptible. Such manipulations include filtering, colour manipulation, geometric distortion, etc. We distinguish these manipulations from lossy compression because these predictable changes may be considered as acceptable to some observers but may be unacceptable to others.

![Figure 2](http://www.ee.columbia.edu/ln/dvmm/publications/99/spie99paper.pdf)

Today, most digital multimedia data are stored or distributed in compressed form. Moreover, to satisfy the various needs of broadcasting, storage and transmission, some transcoding of compressed digital images/videos may be required. For instance, digital video clips are usually shot and stored in the compressed format with a pre-determined bit-rate. But the final distributed bit rate of them may be different. Another example is that digital images shot and stored in one format may need to be distributed in different formats. These transcoding processes change the pixel values of the digital image/video but not its content. Therefore, these processes should not alter the authenticity of the data. Robustness is an important concern in developing multimedia authentication techniques. Without robustness, an authentication
method can only verify the images/videos at the final stage of transcoding processes, but not authenticate them. In other words, unless we trust all the transcoders in the processes, the “reality” or the “intactness” of the multimedia data cannot be proven without robust signatures.

Robustness consideration for authentication is different from that for general watermarking techniques. Watermarks used for copyright protection are expected to be robust to most manipulations. But authentication signatures are expected to survive only acceptable transcoding or compression and reject other manipulations.

**Multimedia Authentication sources: Raw Data v.s. Compresses Data**

Multimedia compression standards have been designed and widely adopted by various applications: JPEG in the WWW, MPEG in VCD, MPEG2 in DVD and H.261 and H.263 in video conferencing. The source of a multimedia authentication system may be raw data or compressed data. In particular applications, the raw format of multimedia data may not be available. For instance, a scanner generates temporary raw images but only saves them in their compressed format; a digital camera which captures image/video produces compressed files only, without generating raw data. Therefore, an authentication system which can only authenticate raw data may have limited uses in practice.
Digital Signatures and Watermarking

What is a Digital Signature

Generally a Digital Signature is a mathematical system that is used to prove the authenticity of a digital message or document. A valid digital signature provides the receiver with a certificate that verifies that the digital message or document was not tampered during transmission. Digital signatures use a combination of a hash function combined with asymmetric cryptography for creating the hash. This proves the integrity of the digital document.

In some countries like the USA and some European countries digital signatures can be used in the court of law. Digital signatures in digital media are similar to handwritten signatures in printed documents. When digital signatures are implemented in a right way (using cryptographic algorithms) they are much harder to be tampered compared to printed documents. In addition to this the person who signed cannot claim that it wasn’t him how signed the digital media.

The idea of a "digital signature" first appeared in Diffie and Hellman’s seminal paper, New Directions in Cryptography. They propose that each user A publish a "public key" (used for validating signatures), while keeping secret a "secret key" (used for producing signatures). In their scheme user A’s signature for a message M is a value which depends on M and on A’s secret key, such that anyone can verify the validity of A’s signature using A’s public key. However, while knowing A’s public key is sufficient to allow one to validate A’s signatures, it does not allow one to easily forge A’s signatures. The digital signature shall depend on the content and some secret information which is only known to the signer. Therefore, it cannot be forged, and the appraiser can verify whether the content of video information matches the information contained in the digital signature. In other words, we can trust the signer as well as his/her digital signature to verify the data integrity.

In signature based techniques, the sender first extracts the key features from the original copy of image/video and then the features are encrypted using a private key, resulting in a signature. The receiver can use the senders public key to decrypt the signature in order to authenticate the received image. The signature is usually stored somewhere other than in the media itself. The digital signature is stored individually in user defined field, like, in header of MPEG sequence or in a separate file. Because multimedia data are stored in specific file format, the digital signature can be considered as being “embedded” in the data. Digital signature methods have taken few research directions – message authentication code (MAC, AMAC and AIMAC), visual hash, robust hash and digital signature itself. These techniques
purse a common technique for authentication: feature extraction and subsequent use of the features for later authentication.

A semi fragile way to detect the tampered pixels of an image was proposed by Vhih-Hsuan Tzeng and Wen-Hsiang Tsai, this method is based on a new type of digital signature which works for colour as well as geometric visual appearance, and prevents explosion of the signature size in the mean time. Their proposed technique is based on two processes, signature generation and signature authentication using authentication and tamper localization.

How do digital Signatures Work

From the senders perspective, the signing operation can be as simple as a click of a button. But several things happen with that one click:

- **Step 1: Getting a Private and Public Key**
  
  In order to digitally sign a document, the sender needs to obtain a private and public key, which is a one-time process. The private key, as the name implies, is not shared and is used only by the signer. The public key is openly available and used by those that need to validate the signer’s digital signature.

- **Step 2: Signing an Electronic Document**
  
  **Initiate the signing process** - Depending on the software used, the sender needs to initiate the signing process (e.g., by clicking a “Sign” button on the software’s toolbar).

  **Create a digital signature** - A unique digital fingerprint of the document (sometimes called a message digest or document hash) is created using a mathematical algorithm (such as SHA-1). Even the slightest difference between two documents would create a separate digital fingerprint of each.

  **Append the signature to the document** - The hash result and the user’s digital certificate (which includes the user’s public key) are combined into a digital sig-
nature (by using the user’s private key to encrypt the document hash). The resulting signature is unique to both the document and the user. Finally, the digital signature is appended to the document. The user sends the signed document to the recipient. The recipient uses the sender’s public key (which is included in the digital certificate) to authenticate the sender’s signature and to ensure that no changes were made to the document after it was signed.

- **Step 3: Validating a Digital Signature**

  **Initiate the validation process** - Depending on the software used, the recipient needs to initiate the validation process (e.g., by clicking a “Validate Signature” menu option button on the software’s toolbar).

  **Decrypt the digital signature** - Using the sender’s public key, the recipient decrypts his digital signature and receives the original document (the document fingerprint).

  **Compares the document fingerprint with her calculated one** – The recipient software then calculates the document hash of the received document and compares it with the original document hash (from the previous step). If they are the same, the signed document has not been altered.

![Diagram of signing an electronic document](image-url)
Hash Based Digital Signatures

Like any other digital signature scheme, hash-based digital signature schemes use a cryptographic hash function. Their security relies on the collision resistance of that hash function. The existence of collision resistant hash functions can be viewed as a minimum requirement for the existence of a digital signature scheme that can sign many documents with one private key. That signature scheme maps documents (arbitrarily long bit strings) to digital signatures (bit strings of fixed length). This shows that digital signature algorithms are in fact hash functions. Those hash functions must be collision resistant: if it were possible to construct two documents with the same digital signature, the signature scheme could no longer be considered secure. This argument shows that there exist hash-based digital signature schemes as long as there exists any digital signature scheme that can sign multiple documents using one private key. As a consequence, hash-based signature schemes are the most important post-quantum signature candidates. Although there is no proof of their quantum computer resistance, their security requirements are minimal. Also, each new cryptographic hash function yields a new hash-based signature scheme. So the construction of secure signature schemes is independent of hard algorithmic problems in number theory or algebra. Constructions from symmetric cryptography suffice. This leads to another big advantage of hash-based signature schemes. The underlying hash function can be chosen in view of the hardware and software resources available. For example, if the signature scheme is to be implemented on a chip that already implements AES, an AES based hash function can be used, thereby reducing the code size of the signature scheme and optimizing its running time.

Hash-based signature schemes were invented by Ralph Merkle. Merkle started from one-time signature schemes, in particular that of Lamport and Diffie. One-time signatures are even more fundamental. The construction of a secure one-time signature scheme only requires a one-way function. One-way functions are necessary and sufficient for secure digital signatures. So one-time signature schemes are really the most fundamental type of digital signature schemes. However, they have a severe disadvantage. One key-pair consisting of a secret signature key and a public verification key can only be used to sign and verify a single document. This is inadequate for most applications. It was the idea of Merkle to use a hash tree that reduces the validity of many one-time verification keys (the leaves of the hash tree) to the validity of one public key (the root of the hash tree). The initial construction of Merkle was not sufficiently efficient, in particular in comparison to the RSA signature scheme. However in the meantime, many improvements have been found. Now hash-based signatures are the most promising alternative to RSA and elliptic curve signature schemes.
**Lamport–Diffie one-time signature scheme**

Let n be a positive integer, the security parameter of LD-OTS. LD-OTS uses a one way function

\[ f: \{0,1\}^n \rightarrow \{0,1\}^n, \quad (1) \]

And a cryptographic hash function:

\[ g: \{0,1\}^* \rightarrow \{0,1\}^n \quad (2) \]

**LD-OTS key pair generation.** The signature key X of LD-OTS consists of 2n bit strings of length n chosen uniformly at random,

\[ X = (\chi_{n-1}[0], \chi_{n-1}[1], \ldots, \chi_0[0], \chi_0[1]) \in R \{0,1\}^{(n,2n)} \]

The LS-OTS verification key Y is

\[ Y = (y_{n-1}[0], y_{n-1}[1], \ldots, y_0[0], y_0[1]) \in R \{0,1\}^{(n,2n)} \]

Where

\[ y_i[j] = f(\chi_i[j]), \quad 0 \leq i \leq n-1, j=0,1. \quad (3) \]

So LD-OTS key generation requires 2n evaluations of f. The signature and verification keys are 2n bits strings of length n.

**LD-OTS signature generation.** A document M ∈ \{0,1\}^* is signed using LD-OTS with a signature key X as in equation (1). Let \( g(M) = d = (d_{n-1}, \ldots, d_0) \) be the message digest of M. Then the LD-OTS signature is

\[ \sigma = (\chi_{n-1}[d_{n-1}], \ldots, \chi_0[d_0]) \in \{0,1\}^{(n,m)} \quad (4) \]

This signature is a sequence on n bit strings, each of length n. They are chosen as a function of the message digest d. The i-th bit string in this signature is \( \chi_i[0] \) if the i-th bit in d is 0 and \( \chi_i[1] \), otherwise. Signing requires no evaluation of f. The length of the signature is \( n^2 \).

**LD-OTS Verification.** To verify a signature \( \sigma = (\sigma_{n-1}, \ldots, \sigma_0) \) of M as in (4), the verifier calculates the message digest \( d = (d_{n-1}, \ldots, d_0) \). Then she checks whether

\[ (f(\sigma_{n-1}), \ldots, f(\sigma_0)) = (y_{n-1}[d_{n-1}], \ldots, y_0[d_0]). \quad (5) \]

Signature verification requires n evaluations of f.
Winternitz one-time signature scheme

While the key and signature generation of LD-OTS is very efficient, the size of the signature is quite large. The Winternitz OTS (W-OTS), which is explained in this section, produces significantly shorter signatures. The idea is to use one string in the one-time signature key to simultaneously sign several bits in the message digest. In literature this proposal appears first in Merkle’s thesis. Merkle writes that the method was suggested to him by Winternitz in 1979 as a generalization of the Merkle OTS. Like LD-OTS, W-OTS uses a one-way function

\[ f : \{0,1\}^n \rightarrow \{0,1\}^n \]

And a cryptographic function

\[ g : \{0,1\}^* \rightarrow \{0,1\}^n \]

W-OTS key pair generation. A Winternitz parameter \( w \geq 2 \) is selected which is the number of bits to be signed simultaneously. Then

\[
t1 = \frac{n}{w}, \quad t2 = \left\lfloor \frac{\lceil \log_2 t1+1 \rceil + w}{w} \right\rfloor, \quad t = t1 + t2 \quad (6)
\]

are determined. The signature key \( X \) is

\[
X = (x_{t-1}, \ldots, x_1, x_0) \in \mathbb{R}^{{0,1}(n,t)} \quad (7)
\]

Where the bit strings \( x_i \) are chosen uniformly at random.

The verification key \( Y \) is computed by applying \( f \) to each bit string in the signature key \( 2^w-1 \) times. So we have

\[
Y = (y_{t-1}, \ldots, y_1, y_0) \in \{0,1\}^{(n,i)} \quad (8)
\]

Where

\[
y_i = f_{2w-1}(x_i), \quad 0 \leq i \leq t-1 \quad (9)
\]

key generation requires \( t(2^w-1) \) evaluations of \( f \) and the lengths of the signature and verification key are \( t-n \) bit, respectively.

W-OTS signature generation. A message \( M \) with message digest \( g(M) = d = (d_{n-1}, \ldots, d_0) \) is signed. First, a minimum number of zeros is prepended to \( d \) such that the length of \( d \) is divisible by \( w \). The extended string \( d \) is split into \( t1 \) bit strings \( b_{t-1}, \ldots, b_{t-1} \) of length \( w \). Then

\[
d = b_{t-1} \parallel \ldots \parallel b_{t-1} \quad (10)
\]
where \(\|\) denotes concatenation. Next, the bit strings \(b_i\) are identified with integers in \(\{0, 1, \ldots, 2^w-1\}\) and the checksum

\[
c = \sum_{i=t-1}^{t-1} (2^w - b_i) \quad (11)
\]

is calculated. Since \(c \leq t_2 2^w\), the length of the binary representation of \(c\) is less than

\[
\lfloor \log_2 t_2 2^w \rfloor + 1 = \lfloor \log_2 t_1 \rfloor + w + 1 \quad (12)
\]

A minimum number of zeros is prepended to this binary representation such that the length of the extended string is divisible by \(w\). That extended string is split into \(t_2\) blocks \(b_{t_2-1}, \ldots, b_0\) of length \(w\). Then

\[
C = b_{t_2-1} \| \ldots \| b_0.
\]

Finally, the signature of \(M\) is computed as

\[
\sigma = (f^b_{t-1}(x_{t-1}), \ldots, f^b_1(x_1), f^b_0(x_0)). \quad (13)
\]

in the worst case, signature generation requires \(t(2^w-1)\) evaluations of \(f\). The W-OTS signature size is \(t \ast n\).

W-OTS verification. For the verification of signature \(\sigma = (\sigma_{t-1}, \ldots, \sigma_0)\) the bit strings \(b_{t-1}, \ldots, b_0\) are calculated as explained in the previous section. Then we check if

\[
(f^{2^w-1-bt-1}_{2^w-1-bt-1}(\sigma_{t-1}), \ldots, f^{2^w-1-b0}_{2^w-1-b0}(\sigma_0)) = (y_{t-1}, \ldots, y_0) \quad (14)
\]

If the signature is valid, then \(\sigma_i = f^b_i(x_i)\) and therefore

\[
f^{2^w-1-bt-1}_{2^w-1-bt-1}(\sigma_{t-1}) = f^{2^w-1}(x_t) = y_t \quad (15)
\]

holds for \(i = t-1, \ldots, 0\). In the worst case, signature verification requires \(t(2^w-1)\) evaluations of \(f\).
What is Digital Watermarking

The idea of hiding data in another media is very old. Digital watermarking is a perceptually invisible pattern embedded in a digital image. Nevertheless, the term digital watermarking first appeared in 1993, when Tirkel at al. (1993) presented two techniques to hide data in images. These methods were based on modifications to the least significant bit (LSB) of the pixel values.

A digital watermark is a pattern permanently embedded into digital data (audio, images and text) that can be detected or extracted using computing operations in order to make assertions about the data. The watermark is hidden in digital data in such a way that it is inseparable from the data and so that it is resistant to many operations not degrading the host document. Thus by means of watermarking, the work is still accessible but permanently marked.

Digital watermarking techniques were born from stenography, which means covered writing. Stenography is the science of communicating information while hiding the existence of communication. The goal of stenography is to hide an information message inside harmless messages in such a way that it is not possible even to detect that there is secret message present. Both stenography and watermarking belong to a category of information hiding, but the objectives and conditions for the two techniques are just opposite. In watermarking, for example, the important information is the “external” data (e.g., images, voices etc.). the “internal” data (e.g. watermark) are additional data for protecting the external data and to prove ownership. In stenography, however, the external data are not very important they are just a carrier of the important information. The internal data are the most important.

Watermarking techniques can be divided into two groups depending on whether the original images is required for watermark extraction or not. In non-oblivious watermarking the original image is needed for watermark extraction. Although this makes non-oblivious techniques more robust to attacks, the need for the original image is clearly a disadvantage. In oblivious techniques the watermark can be extracted from the attacked video without having the original image.

Why Digital Watermarking?

Digital watermarking is an enabling technology for e-commerce strategies: conditional and user-specific access to services and resources. Digital watermarking offers several advantages. The details of a good digital watermarking algorithm can be made public knowledge. Digital watermarking provides the owner of a piece of digital data the means to
mark the data invisibly. The mark could be used to serialize piece of data as it is sold or used as a method to mark a valuable image. For example, this marking allows an owner to safely post an image for viewing but legally provides an embedded copyright to prohibit others from posting the same image, watermarks and attacks on watermarks are two sides of the same coin. The goal of both is to preserve the value of the digital data. However, the goal of a watermark is to remove the watermark without destroying the value of the protected data. The contents of the image can be marked without visible loss of value or dependence on specific formats. For example a bitmap (BMP) image can be compresses to a JPEG image. The result is an image that requires less storage space but cannot be distinguished from the original. Generally, a JPEG compression level of 70% can be applied without humanly visible degradation. This property of digital images allows insertion of additional data in the image without altering the value of the image. The message is hidden in unused “visual space” in the image and stays below the human visible threshold for the image.

Watermarking v.s. Digital Signature

Since the meaning of multimedia data is based on its content, we can modify the multimedia bitstream to embed some codes, i.e., watermarks, without changing the meaning of the content. The embedded watermark may represent either a specific digital producer identification label (PIL) or some content-based codes generated by applying a specific rule. In the authenticator, the watermarks are examined to verify the integrity of the data.

For complete authentication of uncompressed raw multimedia data, watermarking may work better than digital signature methods because:

- The watermarks are always associated with the data and can be conveniently examined and
- There are many spaces in the multimedia data in which to embed the watermarks with negligible quality degradation (known as invisible watermarks).

However, there is no advantage to using the watermarking method in compressed multimedia data for complete verification. Compressed standards have user-defined sections where a digital signature can be placed. Because multimedia data are stored or distributed in specific file format instead of pixel values, the digital signature can be considered as being “embedded” in the data. Once the multimedia data is modified, the user-defined section of the original data is usually discarded by editing software. Even if the digital signature can be preserved by the software, we can easily detect the modification, since the hash values of the modified data will not be the same as the original. Moreover, compresses multimedia data
offer less space for hiding watermarks. Visually quality if the data may be compromised in order to ensure that enough watermarking bits for adequately protecting the data.

For content authentication, compression should be distinguished from other manipulations. Previous watermarks are either too fragile for compression or too flexible to detect malicious manipulations. The performance of an authenticator should be simultaneously evaluated by two parameters: the probability of false alarm and the probability of missing manipulations. Fragile watermarks, which have low probability of miss, usually fail to survive compressions such that their probability of false alarm is very high. Previous researchers have attempted to modify the fragile watermark to make it robust with compression however, such modifications failed to distinguish compression and tampering, when they lower the probability of false alarm, the probability of miss in their system increases significantly. On the other hand, robust watermarks are robust to most manipulations, but are usually too robust to detect malicious manipulations. Their probability of miss is usually too high.
Robust Hash Functions

Hash functions are also known as message digest functions. They are used to extract a fixed-length bit string from a computer file or an image; the bit stream can be of any length. In cryptography hash functions are usually used in digital signatures, so that the receiver can verify that the message is authentic and that it was sent from the right person. A hash function has the following rules:

1. Given a message \( m \) and a hash function \( H \), it should be easy to compute the hash \( h = H(m) \)
2. Given \( h \), it is hard to compute \( m \) such that \( h = H(m) \)
3. Given \( m \), it is hard to find another message \( m' \) such that \( H(m') = H(m) \)

From all the above it is easy to see that hash functions are “infinitely” sensitive, this means that only a small perturbation of the message \( m \) will give a completely different bit-string \( h \). In digital image applications the requirements of what should be a digest of an image are different. For example changing one pixel in an image doesn’t make the image different or non-trustable. In these cases it would be useful to have a technique that would return almost the same bit-strings for similar looking images, and at the same time completely different bit-strings for different images. This is called a robust hash function.

One of the problems that arise is when two images are almost the same, and when are they different. An easy answer to this question is that whenever the human eye says that the images are the same we would like the bit-strings to be approximately the same.
Our Approach

The proposed signature extraction consists of the following two steps:

1. Video feature extraction, and
2. Computation of the signature bits from extracted features.

Video feature extraction

The goal of the video feature extraction block is to obtain a set of robust features that are invariant to certain signal processing operations on the video. The steps followed are numbered below:

1. An absolute difference image is computed between adjacent video frames (frame(n) and frame(n+1)). This step ensures that we only extract signature bits based on active moving regions between two frames. If the video sequence is of a static graphic image and there is no activity the method will fail.

2. The absolute difference image computed in the previous step is downsampled and cropped. This step ensures that the extracted features are invariant to interlaced and progressive video formats and addition of graphics and letterboxes on the corners of the individual frames.

3. The cropped absolute difference image is tiled horizontally and vertically (Figure 5).

4. The intensities of the absolute difference image within each tile are summed up to obtain a coarse absolute difference image.

\[ Q_v(k, l) = \frac{1}{W_x \cdot W_y} \sum_{i=(k-1)W_x}^{kW_x} \sum_{j=(l-1)W_y}^{lW_y} \Delta(i, j) \]

\( \Delta \) is the cropped absolute difference image. An example of the matrice produced by applying the function described above is shown in Table 1.

\( Q_v \): a coarse representation of \( \Delta \), which equals to the average of pixel intensities in images blocks of size \( W_x \cdot W_y \) such that \( K \cdot W_x = 120 \), and \( L \cdot W_y = 160 \). \( Q_v \) is of size \( (K \times L) \). A figure of the feature extraction process is shown in Figure 6.
Figure 5: A representation of the matrix Qv. Each block of the matrix is of size 15x18, so the final matrix Qv is of size 8x9.

Table 1: An example of a Coarse absolute Difference image table (Qv)

<table>
<thead>
<tr>
<th>5,1531</th>
<th>4,0457</th>
<th>1,6951</th>
<th>3,8728</th>
<th>2,6321</th>
<th>2,0617</th>
<th>3,6309</th>
<th>2,5432</th>
<th>2,5639</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,4222</td>
<td>2,816</td>
<td>1,6284</td>
<td>7,384</td>
<td>3,6901</td>
<td>1,858</td>
<td>1,7148</td>
<td>2,0235</td>
<td>1,9944</td>
</tr>
<tr>
<td>4,7049</td>
<td>3,1963</td>
<td>4,6432</td>
<td>9,2457</td>
<td>3,1309</td>
<td>2,8741</td>
<td>2,7469</td>
<td>2,0062</td>
<td>2,0319</td>
</tr>
<tr>
<td>2,5284</td>
<td>2,1988</td>
<td>2,042</td>
<td>8,8296</td>
<td>3,0506</td>
<td>3,2679</td>
<td>2,4728</td>
<td>2,8531</td>
<td>1,3681</td>
</tr>
<tr>
<td>2,0901</td>
<td>1,9309</td>
<td>1,8741</td>
<td>2,2593</td>
<td>2,3235</td>
<td>2,5864</td>
<td>2,0444</td>
<td>2,0481</td>
<td>1,9125</td>
</tr>
<tr>
<td>3,4802</td>
<td>2,4852</td>
<td>2,8</td>
<td>2,4247</td>
<td>2,3988</td>
<td>3,2136</td>
<td>2,5593</td>
<td>2,1222</td>
<td>3,6056</td>
</tr>
<tr>
<td>3,0136</td>
<td>2,5481</td>
<td>2,8012</td>
<td>2</td>
<td>2,7</td>
<td>1,7111</td>
<td>2,9556</td>
<td>3,4198</td>
<td>2,5333</td>
</tr>
<tr>
<td>3,3642</td>
<td>2,4852</td>
<td>2,4765</td>
<td>2,0951</td>
<td>2,0951</td>
<td>3,3395</td>
<td>3,3037</td>
<td>2,1642</td>
<td>2,1236</td>
</tr>
</tbody>
</table>

Figure 6
Robust Hash

This block takes as input the matrix \( Q_v \), and generates the signature by generating \( K \) hash bits. We use a robust hash function for this purpose as small perturbations in the video features would not change the hash bits drastically. Also by using a robust hash function instead of sending \((M \times N)\) values we only send \( K \) bits. For this purpose we use the robust hash function shown below:

\[
H_k = \sum_{i=1}^{M} \sum_{j=1}^{N} Q_v(i, j) * P_k(i, j)
\]

Where \( Q_v \) is the coarse absolute difference image. We generate \( K \) \((K=36)\) random matrices \( P_i \) each with the same dimensions as the matrix \( Q_v(M \times N) \). The matrix entries are uniformly distributed random variables in \([0, 1]\). We compute the mean of matrix \( P_i \) and subtract it from each matrix element in \( P_i \) (Table2). Then the matrix \( Q_v \) is projected onto these \( K \) random vectors (one such multiplication is shown in Table3). Where \( H_k \) is the projection of the matrix \( Q \) onto the random vector \( P_k \). Using the hash function mentioned above every such multiplication concludes to a number, so in the end we will have 36 elements . Using the median of these projections as a threshold we generate \( K \) hash bits for the matrix \( Q_v \). We generate a hash bit ‘1’ if the projection \( H_k \) is greater than the threshold, otherwise we generate a hash bit of ‘0’.

| 0.5434 | 0.1367 | 0.8117 | 0.1754 | 0.1051 | 0.7425 | 0.2859 | 0.1781 | 0.9338 |
| 0.2784 | 0.5751 | 0.1719 | 0.3728 | 0.3819 | 0.6302 | 0.8524 | 0.2377 | 0.9464 |
| 0.4245 | 0.8913 | 0.8162 | 0.0057 | 0.0365 | 0.5818 | 0.975 | 0.0449 | 0.6023 |
| 0.8448 | 0.2092 | 0.2741 | 0.2524 | 0.8904 | 0.0204 | 0.8849 | 0.5054 | 0.3878 |
| 0.0047 | 0.1853 | 0.4317 | 0.7957 | 0.9809 | 0.21 | 0.3595 | 0.3763 | 0.3632 |
| 0.1126 | 0.1084 | 0.94 | 0.0153 | 0.0599 | 0.5447 | 0.5989 | 0.5928 | 0.2043 |
| 0.6707 | 0.2197 | 0.8176 | 0.5988 | 0.8905 | 0.7691 | 0.3548 | 0.6299 | 0.2768 |
| 0.8259 | 0.9786 | 0.3361 | 0.6038 | 0.5769 | 0.2507 | 0.3402 | 0.1426 | 0.2465 |

Table 2: An example of a random matrix, with dimensions 8x9 and entries \( \in [0, 1] \). The mean of this specific matrix is 0.464297
Table 3: An example of the matrix shown in table 1 after the subtraction of its mean value from every element.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.079103</td>
<td>-0.3276</td>
<td>0.347403</td>
<td>-0.2889</td>
<td>-0.3592</td>
<td>0.278203</td>
<td>-0.1784</td>
<td>-0.2862</td>
</tr>
<tr>
<td>-0.1859</td>
<td>0.110803</td>
<td>-0.2924</td>
<td>-0.0915</td>
<td>-0.0824</td>
<td>0.165903</td>
<td>0.388103</td>
<td>-0.2266</td>
</tr>
<tr>
<td>-0.0398</td>
<td>0.427003</td>
<td>0.351903</td>
<td>-0.4586</td>
<td>-0.4278</td>
<td>0.117503</td>
<td>0.510703</td>
<td>-0.4194</td>
</tr>
<tr>
<td>0.380503</td>
<td>-0.2551</td>
<td>-0.1902</td>
<td>-0.2119</td>
<td>0.426103</td>
<td>-0.4439</td>
<td>0.420603</td>
<td>0.041103</td>
</tr>
<tr>
<td>-0.4596</td>
<td>-0.279</td>
<td>-0.0326</td>
<td>0.331403</td>
<td>0.516603</td>
<td>-0.2543</td>
<td>-0.1048</td>
<td>-0.088</td>
</tr>
<tr>
<td>-0.3427</td>
<td>-0.3559</td>
<td>0.475703</td>
<td>-0.449</td>
<td>-0.4044</td>
<td>0.080403</td>
<td>0.134603</td>
<td>0.128503</td>
</tr>
<tr>
<td>0.206403</td>
<td>-0.2446</td>
<td>0.353303</td>
<td>0.134503</td>
<td>0.426203</td>
<td>0.304803</td>
<td>-0.2136</td>
<td>-0.1241</td>
</tr>
<tr>
<td>0.361603</td>
<td>0.514303</td>
<td>-0.1282</td>
<td>0.139503</td>
<td>0.112603</td>
<td>-0.2136</td>
<td>-0.1241</td>
<td>-0.3217</td>
</tr>
</tbody>
</table>

Table 4: The projection of the matrix Qv on the random vector shown in Table2. This table is the result of Qv.*Pk.

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.407625</td>
<td>-1.32536</td>
<td>0.588882</td>
<td>-1.11884</td>
<td>-0.94544</td>
<td>0.573571</td>
<td>-0.64774</td>
<td>-0.72786</td>
<td>1.203758</td>
</tr>
<tr>
<td>-0.63618</td>
<td>0.312021</td>
<td>-0.47614</td>
<td>-0.67562</td>
<td>-0.30405</td>
<td>0.308247</td>
<td>0.665519</td>
<td>-0.45852</td>
<td>0.961506</td>
</tr>
<tr>
<td>-0.18724</td>
<td>1.364829</td>
<td>1.633955</td>
<td>-4.24005</td>
<td>-1.33939</td>
<td>0.337715</td>
<td>1.402849</td>
<td>-0.84139</td>
<td>0.280408</td>
</tr>
<tr>
<td>0.962063</td>
<td>-0.56091</td>
<td>-0.38838</td>
<td>-1.87097</td>
<td>1.299869</td>
<td>-1.45061</td>
<td>1.04067</td>
<td>0.11727</td>
<td>-0.10466</td>
</tr>
<tr>
<td>-0.9606</td>
<td>-0.53872</td>
<td>-0.06109</td>
<td>0.748738</td>
<td>1.200327</td>
<td>-0.65771</td>
<td>-0.21425</td>
<td>-0.18023</td>
<td>-0.19335</td>
</tr>
<tr>
<td>-1.19265</td>
<td>-0.88448</td>
<td>1.331968</td>
<td>-1.08868</td>
<td>-0.97007</td>
<td>0.258382</td>
<td>0.344489</td>
<td>0.272709</td>
<td>-0.93745</td>
</tr>
<tr>
<td>0.622015</td>
<td>-0.62326</td>
<td>0.989672</td>
<td>0.269006</td>
<td>1.150748</td>
<td>0.521548</td>
<td>-0.32363</td>
<td>0.566328</td>
<td>-0.47499</td>
</tr>
<tr>
<td>1.216504</td>
<td>1.278145</td>
<td>-0.31748</td>
<td>0.292272</td>
<td>0.235914</td>
<td>-0.71331</td>
<td>-0.40998</td>
<td>-0.69622</td>
<td>-0.46251</td>
</tr>
</tbody>
</table>

From applying the function Hk=∑M∑NQv(i,j) * Pk(i,j) from every matrix produced by multiplying Qv.*Pk a single number is generated. We have 36 random matrices so 36 numbers will be produced by the above multiplication. Using the median of these projections as a threshold we generate K hash bits for the matrix Qv. We generate a hash bit ‘1’ if the projection Hk is grater than the threshold. Otherwise we generate a hash bit ‘0’.
The signature sensitivity

Our goal is to generate signatures in a way that when we compare two completely different videos we would like the percentage of the bits that flip to be large. In addition to signatures of modified videos where the same percentage should be small.

The program made for this paper does exactly that. It compares the histograms produced by the hamming distance of two modified videos (case 1) and the hamming distance of two completely different videos (case 2). What we expect is the histogram of case 1 to be nearer to the hamming distance value of zero, which means that the bit error rate of two modified videos should be smaller than the bit error rate of two completely different videos.
Our tests

In order to prove the above we designed an application that takes 3 videos as entries, and provides a histogram that shows the correlation between videos that have been modified and different videos. The original video has been modified in the following ways:

- Noise increase
- Brightness reduction

The application was designed in MatLab 7.10.0(R 2010a), and it provides a gui interface from which the user can select the videos needed for the comparison.

The above interface takes three videos as it’s entrance and produces a histogram for the comparison of the first and the second, and one histogram for the comparison of the first video with the third. When comparing the first two videos the histogram’s color is blue, and it is red when we compare the first video with the third. All overlapping areas are colored with a mixture of blue and red. In the bottom right corner is the Close Button that closes all opened figures except the GUI interface and clears the histogram area.
Comparing videos with brightness modification and completely different videos

In the next test the first two videos are the same only with a brightness reduction and the third one a completely different video. The GUI the parameters should be as shown in the figure below:

From the histogram we can see that the hamming distances of the original and the modified video are closer to zero than the Hamming distances of the different videos. We can see that the hamming distance of the altered videos are mostly bellow 0.04 whereas for the completely different videos the hamming distance is mostly over 0.05. Moreover the overlapped areas are very small.
Figure 9

Comparing videos after noise modification and completely different videos

In the next test the first two videos are the same but the in the second video we have added more noise, the third one a completely different video. The GUI the parameters should be as shown in the figure below:
From the histogram we can see that the hamming distances of the original and the modified video are closer to zero than the Hamming distances of the different videos. We can see that the hamming distance of the altered videos are mostly below 0.05 whereas for the completely different videos the hamming distance is mostly over 0.05. Moreover the overlapped areas are very small.
Comparing videos after rotation and completely different videos

In the next test the first two videos are the same but the second is rotated by $10^\circ$, the third one a completely different video. The GUI the parameters should be as shown in the figure below:
From the histogram we can see that the hamming distances of the original and the modified video are closer to zero than the Hamming distances of the different videos. We can see that the hamming distance of the altered videos are mostly below 6 whereas for the completely different videos the hamming distance is mostly over 0.06. The overlapped areas are very small as in the other tests described above.

Figure 13
Matlab Code

```matlab
obj=mmreader('Original.avi');
obj2=mmreader('noise_incr.avi');
obj3=mmreader('movie_0001.avi');

nFrames = obj.NumberOfFrames;
n2Frames=obj2.NumberOfFrames;
n3Frames=obj3.NumberOfFrames;

str=sprintf('Number of Frames %d',nFrames)
str2=sprintf('Number of Frames %d',n2Frames)
str3=sprintf('Number of Frames %d',n3Frames)

vidHeight = obj.Height;
vidWidth = obj.Width;

vidHeight = obj2.Height;
vidWidth = obj2.Width;

vidHeight = obj3.Height;
vidWidth = obj3.Width;

%Creating a folder for saving the video images
if isequal(exist('images/diff_images', 'dir'),7)
    display('folder already exists');
else
    mkdir('images/diff_images');
end

%Code for cutting the video into frames
for k = 1 : 10
    this_frame = read(obj, k);
    thisfig = figure();
    thisax = axes('Parent', thisfig);
    image(this_frame, 'Parent', thisax);
    title(thisax, sprintf('Video 1 #%d', k));
end

for l = 1 : 10
    this_frame2 = read(obj2, l);
    thisfig2 = figure();
    thisax2 = axes('Parent', thisfig2);
    image(this_frame2, 'Parent', thisax2);
    title(thisax2, sprintf('Video 2 #%d', l));
end

for m = 1 : 10
    this_frame3 = read(obj3, l);
    thisfig3 = figure();
    thisax3 = axes('Parent', thisfig3);
```
% Code for saving video frames in the folder that was created above and converting them into grayscale images

numFramesWritten=0; % Number of written frames for Video1
numFramesWritten2=0; % Number of written frames for Video2
numFramesWritten3=0; % Number of written frames for Video3

for t = 1 : 10  % t number of frames
    currFrame = read(obj, t);  % reading individual frames
    grayim = rgb2gray(currFrame);
    opBaseFileName = sprintf('%3.3d.png', t);
    opFullFileName = fullfile('images', opBaseFileName);
    imwrite(grayim, opFullFileName, 'png');  % saving as 'png' file
    % indicating the current progress of the file/frame written
    progIndication = sprintf('Wrote frame %4d of %d.', t, nFrames);
    disp(progIndication);
    numFramesWritten = numFramesWritten + 1;
end

for s = 1 : 10
    currFrame2 = read(obj2, s);  % reading individual frames
    grayim2 = rgb2gray(currFrame2);
    opBaseFileName2 = sprintf('%3.3d.png', s);
    opFullFileName2 = fullfile('images', opBaseFileName2);
    imwrite(grayim2, opFullFileName2, 'png');  % saving as 'png' file
    % indicating the current progress of the file/frame written
    progIndication2 = sprintf('Wrote frame %4d of %d.', s, n2Frames);
    disp(progIndication2);
    numFramesWritten2 = numFramesWritten2 + 1;
end  % end of 'for' loop

for mm = 1 : 10
    currFrame3 = read(obj3, mm);  % reading individual frames
    grayim3 = rgb2gray(currFrame3);
    opBaseFileName3 = sprintf('%3.3d.png', mm);
    opFullFileName3 = fullfile('images', opBaseFileName3);
    imwrite(grayim3, opFullFileName3, 'png');  % saving as 'png' file
    % indicating the current progress of the file/frame written
    progIndication3 = sprintf('Wrote frame %4d of %d.', mm, n3Frames);
    disp(progIndication3);
    numFramesWritten3 = numFramesWritten3 + 1;
end

progIndication = sprintf('Wrote %d frames to folder images', numFramesWritten3);
disp(progIndication);

% Calculating the brightness difference between frames, the images have been modified to grayscale, so the difference image will arise from the pixel difference between two frames
sum1=0;

for n=1:30
    bg=read(obj,n);
    currFrame=read(obj,n+1);
    diff=abs(bg-currFrame);

    bg2=read(obj2,n);
    currFrame2=read(obj2,n+1);
    diff2=abs(bg2-currFrame2);

    bg3=read(obj3,n);
    currFrame3=read(obj3,n+1);
    diff3=abs(bg3-currFrame3);

%Here we change the image size by down sampling, and after that we save the image
    dsi = imresize(diff, [120 160]);
    dsi2 = imresize(diff2, [120 160]);
    dsi3 = imresize(diff3, [120 160]);

%Slicing the image into blocks, the blocks are 15X18, and after that we calculate the mathematical function given fro calculating Qv
    Wx=15;
    Wy=18;

    fun=@(block_struct) *mean2(block_struct.data);
    Qv = blockproc(dsi,[15 18],fun);

    fun=@(block_struct) *mean2(block_struct.data);
    Qv2 = blockproc(dsi2,[15 18],fun);

    fun=@(block_struct) *mean2(block_struct.data);
    Qv3 = blockproc(dsi3,[15 18],fun);

%The image size is 852x480,
%Wx*Wy is the size of each block
%K=8, L=9 so K*Wx=120->Wx=15 & Wy=17, so i=57 & j=28. We will have to create a function that will calculate each image element as following:Q(i,j)=1/256.5*Δ(i,j) where Δ(i,j) is the difference image

syma=0;
H=zeros(36,1);
for K=1:36
    %Creating the random matrices 9X8. We want them to start from the same point so that if we have the same image the code will produce the same sequence of random numbers, so we seed the function
    s = RandStream('mt19937ar','Seed',100);
    RandStream.setDefaultStream(s);
out = rand(8,9);
s = RandStream('mt19937ar','Seed',100);
RandStream.setDefaultStream(s);
out2= rand(8,9);
s = RandStream('mt19937ar','Seed',100);
RandStream.setDefaultStream(s);
out3= rand(8,9);

%Computing the mean of the matrices
m=mean2(out);
m2=mean2(out2);
m3=mean2(out3);

%Subtracting the mean from every matrix element
final=out-m;
final2=out2-m2;
final3=out3-m3;

%Calculating the projection of the random matrices onto the random vector
H=sum(Qv.*final);
H2=sum(Qv2.*final2);
H3=sum(Qv3.*final3);
H(K)=sum(H);
H2(K)=sum(H2);
H3(K)=sum(H3);

%Calculating the median of the projections
for kk=1:9
H(kk);
M=median(H);
M2=median(H2);
M3=median(H3);
end

%Assigning a value to the threshold
Thres=M  %Αυτό είναι το threshold
Thres2=M2
Thres3=M3

%If Hk is bigger than the threshold we assign the value 1, else we assign the value 0
for Hk=1:36  %to 36 itan 9
if H(Hk)>Thres Kf(Hk)=1;
else Kf(Hk)=0;
end

if H2(Hk)>Thres2 Kf2(Hk)=1;
else Kf2(Hk)=0;
end

if H3(Hk)>Thres3 Kf3(Hk)=1;
else Kf3(Hk)=0;
% Displaying the signature of the first video
for Hkk=1:36
    disp(Kf{Hkk})
end

sum1=0;

% Displaying the signature of the second video
for Hkkk=1:36
    disp(Kf2(Hkkk))
end

% Displaying the signature of the third video

% Computing the Hamming distance
BER(n) = pdist2(Kf,Kf2,'hamming')
BER2(n) = pdist2(Kf,Kf3,'hamming')

% Plotting
axes(handles.axes1);
hist(BER)
hold on
hist(BER2)
legend('First Two Videos','First & Third Video')
xlabel('Bit Error Rate')
h = findobj(gca,'Type','patch');
set(h(1),'FaceColor','r','EdgeColor','w','facealpha',0.75)
set(h(2),'FaceColor','b','EdgeColor','w','facealpha',0.75)
hold off
end
end
end
function varargout = guitest2(varargin)
% GUI TEST2 M-file for guitest2.fig
% GUI TEST2, by itself, creates a new GUI TEST2 or raises the existing
% singleton*. 
% 
% H = GUI TEST2 returns the handle to a new GUI TEST2 or the handle to
% the existing singleton*. 
% GUI TEST2('CALLBACK', hObject, eventData, handles,...) calls the local
% function named CALLBACK in GUI TEST2.M with the given input arguments.
% GUI TEST2('Property', 'Value', ...) creates a new GUI TEST2 or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before guitest2_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to guitest2_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)". 
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guitest2
% Last Modified by GUIDE v2.5 17-Apr-2015 11:26:27

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @guitest2_OpeningFcn, ...
    'gui_OutputFcn', @guitest2_OutputFcn, ...
    'gui_LayoutFcn', [], ..., ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guitest2 is made visible.
function guitest2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guitest2 (see VARARGIN)

% Choose default command line output for guitest2
handles.output = hObject;

% guidata handles structure
guidata(hObject, handles);

% UIWAIT makes guitest2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
% function varargout = guitest2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;

% Code for the first popup menu
% --- Executes on selection change in video.
% function video_Callback(hObject, eventdata, handles)
% hObject    handle to video (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if ispc && isequal(get(hObject,'video'),...
    get(0,'wg_cs_10.mpg'))
    set(hObject,'obj','wg_cs_10.mpg');
elseif ispc && isequal(get(hObject,'video'),...
    get(1,'brightness_reduction.avi'))
    set(hObject,'obj','brightness_reduction.avi');
elseif ispc && isequal(get(hObject,'video'),...
    get(2,'noise_incr.avi'))
    set(hObject,'obj','noise_incr.avi');
elseif ispc && isequal(get(hObject,'video'),...
    get(3,'rotation.avi'))
    set(hObject,'obj','rotation.avi');
    end
end

if ispc && isequal(get(hObject,'video'),...
    get(5,'Movie_0001.avi'))
    set(hObject,'obj','Movie_0001.avi');
end

% Hints: contents = cellstr(get(hObject,'String')) returns video
% contents as cell array
% contents{get(hObject,'Value')} returns selected item from video
% --- Executes during object creation, after setting all properties.
function video_CreateFcn(hObject, eventdata, handles)
% hObject    handle to video (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

%Code for executing the program when the Compute button is pressed
% --- Executes on button press in button.
function button_Callback(hObject, eventdata, handles)
% hObject    handle to button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%obj=mmreader('Movie_0001.avi');

contents = get(handles.video,'String');
contents2 = get(handles.video2,'String');
contents3 = get(handles.video3,'String');

videovalue = contents{get(handles.video,'Value')};
videovalue2 = contents{get(handles.video2,'Value')};
videovalue3 = contents{get(handles.video3,'Value')};

obj=mmreader(videovalue);
obj2=mmreader(videovalue2);
obj3=mmreader(videovalue3);

nFrames = obj.NumberOfFrames;
n2Frames=obj2.NumberOfFrames;
n3Frames=obj3.NumberOfFrames;
str=sprintf('Number of Frames %d',nFrames)
str2=sprintf('Number of Frames %d',n2Frames)
str3=sprintf('Number of Frames %d',n3Frames)
vidHeight = obj.Height;
vidWidth = obj.Width;
vidHeight = obj2.Height;
vidWidth = obj2.Width;
vidHeight = obj3.Height;
vidWidth = obj3.Width;

%Creating a folder for saving the video images
if isequal(exist('images/diff_images', 'dir'),7)

% What will be displayed if the folder already exists
% Code for cutting the video into frames

for k = 1 : 10  % If we want we can test for more or less frames
    this_frame = read(obj, k);
    thisfig = figure();
    thisax = axes('Parent', thisfig);
    image(this_frame, 'Parent', thisax);
    title(thisax, sprintf('Video 1 #%d', k));
end

for l = 1 : 10
    this_frame2 = read(obj2, l);
    thisfig2 = figure();
    thisax2 = axes('Parent', thisfig2);
    image(this_frame2, 'Parent', thisax2);
    title(thisax2, sprintf('Video 2 #%d', l));
end

for m = 1 : 10
    this_frame3 = read(obj3, l);
    thisfig3 = figure();
    thisax3 = axes('Parent', thisfig3);
    image(this_frame3, 'Parent', thisax3);
    title(thisax3, sprintf('Video 3 #%d', m));
end

% Code for saving video frames in the folder that was created above and converting them into grayscale images

numFramesWritten=0; % Number of written frames for Video1
numFramesWritten2=0; % Number of written frames for Video2
numFramesWritten3=0; % Number of written frames for Video3

for t = 1 : 10  % t number of frames
    currFrame = read(obj, t);  % reading individual frames
    grayim=rgb2gray(currFrame);
    opBaseFileName = sprintf('%3.3d.png', t);
    opFullFileName = fullfile('images', opBaseFileName);
    imwrite(grayim, opFullFileName, 'png');  % saving as 'png' file
    progIndication = sprintf('Wrote frame %4d of %d.', t, nFrames);
    disp(progIndication);
    numFramesWritten = numFramesWritten + 1;
end

for s = 1 : 10
    currFrame2 = read(obj2, s);  % reading individual frames
    grayim2=rgb2gray(currFrame2);
    opBaseFileName2 = sprintf('%3.3d.png', s);
    opFullFileName2 = fullfile('images', opBaseFileName2);
imwrite(grayim2, opFullFileName2, 'png'); %saving as 'png' file
%indicating the current progress of the file/frame written
progIndication2 = sprintf('Wrote frame %4d of %4d.', s, n2Frames);
disp(progIndication2);
numFramesWritten2 = numFramesWritten2 + 1;
end
%end of 'for' loop

for mm = 1 : 10
    currFrame3 = read(obj3, mm); %reading individual frames
    grayim3 = rgb2gray(currFrame3);
    opBaseFileName3 = sprintf('%3.3d.png', mm);
    opFullFileName3 = fullfile('images', opBaseFileName3);
    imwrite(grayim3, opFullFileName3, 'png'); %saving as 'png' file
%indicating the current progress of the file/frame written
    progIndication3 = sprintf('Wrote frame %4d of %4d.', mm, n3Frames);
disp(progIndication3);
    numFramesWritten3 = numFramesWritten3 + 1;
end

progIndication = sprintf('Wrote %d frames to folder images', numFramesWritten3);
disp(progIndication);

%Calculating the brightness difference between frames, the images have been modified to grayscale, so the difference image will arise from the pixel difference between two frames

sum1=0;
for n=1:30
    bg=read(obj, n);
    currFrame=read(obj, n+1);
    diff=abs(bg-currFrame);
    bg2=read(obj2, n);
    currFrame2=read(obj2, n+1);
    diff2=abs(bg2-currFrame2);
    bg3=read(obj3, n);
    currFrame3=read(obj3, n+1);
    diff3=abs(bg3-currFrame3);
    %Here we change the image size by down sampling, and after that we save the image
    dsi = imresize(diff, [120 160]);
    dsi2 = imresize(diff2, [120 160]);
    dsi3 = imresize(diff3, [120 160]);

    %Slicing the image into blocks, the blocks are 15X18, and after that we calculate the mathematical function given for calculating Qv
    Wx=15;
fun=@(block_struct) mean2(block_struct.data);
Qv = blockproc(dsi,[15 18],fun);

fun=@(block_struct) mean2(block_struct.data);
Qv2 = blockproc(dsi2,[15 18],fun);

fun=@(block_struct) mean2(block_struct.data);
Qv3 = blockproc(dsi3,[15 18],fun);

%The image size is 852x480,
%Wx*Wy is the size of each block
%K=8, L=9 so K*Wx=120=>Wx=15 & Wy=17, so i=57 & j=28. We will have
to create a function that will calculate each image element as fol-
lowing:Q(i,j)=1/256.5*Δ(i,j) where Δ(i,j) is the difference image

syma=0;
H=zeros(36,1);
for K=1:36

%Creating the random matrices 9X8. We want them to start from the
same point so that if we have the same image the code will produce
the same sequence of random numbers, so we seed the function
    s = RandStream('mt19937ar','Seed',100);
    RandStream.setDefaultStream(s);
    out = rand(8,9);
    s = RandStream('mt19937ar','Seed',100);
    RandStream.setDefaultStream(s);
    out2= rand(8,9);
    s = RandStream('mt19937ar','Seed',100);
    RandStream.setDefaultStream(s);
    out3= rand(8,9);

%Computing the mean of the matrices
    m=mean2(out);
    m2=mean2(out2);
    m3=mean2(out3);
%Subtracting the mean from every matrice element
    final=out-m;
    final2=out2-m2;
    final3=out3-m3;

%Calculating the projection of the random matrices onto the random
vector
    H=sum(Qv.*final);
    H2=sum(Qv2.*final2);
    H3=sum(Qv3.*final3);
    H(K)=sum(H);
    H2(K)=sum(H2);
    H3(K)=sum(H3);

end
Calculating the median of the projections
for kk=1:9
    H(kk);
    M=median(H);
    M2=median(H2);
    M3=median(H3);
end

Assigning a value to the threshold
Thres=M %Αυτό είναι το threshold
Thres2=M2
Thres3=M3

If Hk is bigger than the threshold we assign the value 1, else we assign the value 0
for Hk=1:36 %to 36 itan 9
    if H(Hk)>Thres Kf(Hk)=1;
        else Kf(Hk)=0;
    end
    if H2(Hk)>Thres2 Kf2(Hk)=1;
        else Kf2(Hk)=0;
    end
    if H3(Hk)>Thres3 Kf3(Hk)=1;
        else Kf3(Hk)=0;
    end
end

Displaying the signature of the first video
for Hkk=1:36
    disp(Kf(Hkk))
end
sum1=0;

Displaying the signature of the second video
for Hkkk=1:36 %to 36 itan 9
    disp(Kf2(Hkkk))
end

Displaying the signature of the third video

computing the Hamming distance
BER(n) =(pdist2(Kf,Kf2,'hamming'))
BER2(n) =(pdist2(Kf,Kf3,'hamming'))

Plotting
    axes(handles.axes1);
    hist(BER)
    hold on
    hist(BER2)
    legend('First Two Videos','First & Third Video')
xlabel('Bit Error Rate')
h = findobj(gca,'Type','patch');
set(h(1), 'FaceColor', 'r', 'EdgeColor', 'w', 'facealpha', 0.75)
set(h(2), 'FaceColor', 'b', 'EdgeColor', 'w', 'facealpha', 0.75)
hold off
end

% --- Executes during object creation, after setting all properties.
function text1_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to text1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called
    contents = get(handles.button, 'String');
    videovalue = contents{get(handles.button, 'Value')};

%disp(Kf);

% --- Executes on selection change in video2.
function video2_Callback(hObject, eventdata, handles)
    % hObject    handle to video2 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    if ispc && isequal(get(hObject, 'video'),...
        get(0,'wg_cs_10.mpg'))
        set(hObject, 'obj', 'wg_cs_10.mpg');
    else
        set(hObject, 'obj', get(hObject, 'video'));
    end
    if ispc && isequal(get(hObject, 'video'),...
        get(1,'brightness_reduction.avi'))
        set(hObject, 'obj', 'brightness_reduction.avi');
    else
        set(hObject, 'obj', get(hObject, 'video'));
    end
    if ispc && isequal(get(hObject, 'video'),...
        get(2,'noise_incr.avi'))
        set(hObject, 'obj', 'noise_incr.avi');
    else
        set(hObject, 'obj', get(hObject, 'video'));
    end
    if ispc && isequal(get(hObject, 'video'),...
        get(3,'rotation.avi'))
        set(hObject, 'obj', 'rotation.avi');
    else
        set(hObject, 'obj', get(hObject, 'video'));
    end
    if ispc && isequal(get(hObject, 'video'),...
        get(5,'Movie_0001.avi'))
        set(hObject, 'obj', 'Movie_0001.avi');
    elseif ispc && isequal(get(hObject, 'video'),...
        get(5,'Movie_0001_incr_noise.avi'))
        %set(hObject, 'obj', 'Movie_0001_incr_noise.avi');
        end
        end
        end

% Hints: contents = cellstr(get(hObject,'String')) returns video2 contents as cell array
% contents{get(hObject,'Value')} returns selected item from video2
% --- Executes during object creation, after setting all properties.
function video2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to video2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

% --- Executes during object creation, after setting all properties.
function text2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
contents2 = get(handles.button,'String');
videovalue2 = contents2{get(handles.button,'Value')};

% --- Executes on button press in button2.

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1

% --- Executes during object creation, after setting all properties.
function BER1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BER1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% --- Executes on selection change in video3.
function video3_Callback(hObject, eventdata, handles)
% hObject    handle to video3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if ispc && isequal(get(hObject,'video'),...
    get(1,'wg_cs_10.mpg'))
    set(hObject,'obj','wg_cs_10.mpg');
if ispc && isequal(get(hObject,'video'),...
get(2,'brightness_reduction.avi'))
set(hObject,'obj','brightness_reduction.avi');
if ispc && isequal(get(hObject,'video'),...
get(3,'noise_incr.avi'))
set(hObject,'obj','noise_incr.avi');
    if ispc && isequal(get(hObject,'video'),...
get(4,'rotation.avi'))
set(hObject,'obj','rotation.avi');
    if ispc && isequal(get(hObject,'video'),...
get(6,'Movie_0001.avi'))
set(hObject,'obj','Movie_0001.avi');
    %if ispc && isequal(get(hObject,'video'),...
    %get(5,'Movie_0001_incr_noise.avi'))
    %set(hObject,'obj','Movie_0001_incr_noise.avi');
end
end
end
end

end

% Hints: contents = cellstr(get(hObject,'String')) returns video3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from video3

% --- Executes during object creation, after setting all properties.
function video3_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to video3 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: popupmenu controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),
                        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

% --- Executes on selection change in pop1.
function pop1_Callback(hObject, eventdata, handles)
    % hObject    handle to pop1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: contents = cellstr(get(hObject,'String')) returns pop1 contents as cell array
    %        contents{get(hObject,'Value')} returns selected item from pop1

    for n=1:30
        choice = get(hObject,'Value'); % Obtaining the selection from pop-up menu.
        switch choice
case 1
    y = n^2;
    
    case 2
    y = n^3;
    
    case 3
    y = exp(n);
end

for Hkk=1:36
    set(handles.text1,'String',num2str(Kf));
disp(Kf(Hkk))
end

sum1=0;

for Hkkk=1:36
    set(handles.text2,'String',num2str(Kf2));
disp(Kf2(Hkkk))
end

for Hkkk=1:36
    set(handles.text3,'String',num2str(Kf3));
end

end

set(handles.output,'string',y);

% --- Executes during object creation, after setting all properties.
function pop1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pop1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
%
% Hint: popupmenu controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in pop1.
function popupmenu5_Callback(hObject, eventdata, handles)
% hObject    handle to pop1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns pop1 contents as cell array
%       contents{get(hObject,'Value')} returns selected item from pop1

% --- Executes during object creation, after setting all properties.
function popupmenu5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pop1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in CloseButton.
function CloseButton_Callback(hObject, eventdata, handles)
% hObject    handle to CloseButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
for i = 1:30
    if ishandle(i)
        delete(i);
    end
end
cla reset;
References


http://el.wikipedia.org/wiki/%CE%A8%CE%B7%CF%86%CE%B9%CE%B1%CE%BA%CE%AE_%CF%80%CE%BF%CE%B3%CF%81%CE%B1%CF%86%CE%AE


