Temporal Synchronization in Video Watermarking

by Eugene T. Lin and Edward J. Delp,

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Temporal Synchronization in Video Watermarking

Eugene T. Lin, *Student Member, IEEE,* and Edward J. Delp, *Fellow, IEEE*

***Abstract*—One of the challenges for blind watermark detection is synchronization. Synchronization is the process of identifying the coordinates of an embedded watermark and is crucial in successful watermark detection. If the detector's input is watermarked but synchronization fails, then the embedded watermark will not be detected. In this paper, temporal synchronization for blind video watermark detection is examined by developing new models for watermark embedding and detection. The structure of the watermark, and specifically its key schedule, dramatically affects the ease of synchronization. The new embedder models the construction of the watermark by using a state machine key generator. The key generator can produce time-invariant, time-independent, and time-periodic key schedules as special cases. The watermark detector uses a queue and a state predictor to perform a search to establish and maintain temporal synchronization. These models are general and can be applied to many symmetric blind video watermarking techniques. It is shown that a watermark without temporal redundancy in its key schedule is vulnerable to attacks such as frame dropping and transposition. Using the models, a watermark more resilient against temporal synchronization attacks is designed by adding temporal redundancy in the watermark construction. Experimental results from an implementation of the models are presented.**

***Index Terms*—Synchronization, video, watermarking.**

I. Introduction

A WATERMARK is a signal that is securely, imperceptibly, and indelibly embedded into original content such as an image, video, or audio signal, producing a watermarked signal. The watermark describes information that can protect or enhance the content, for example identifying the owner, origin, or recipient of the content. Secure embedding implies that an embedded watermark cannot be easily tampered with, forged, or transferred from the watermarked signal to another arbitrary signal [1]. Imperceptible embedding implies that the presence of the watermark is unnoticeable when the watermarked signal is displayed, even though the act of watermark embedding introduces distortion into the audible or visible components of the watermarked signal. Indelible (robust) embedding implies that the watermark cannot be easily removed or separated from the watermarked signal using signal processing, estimation, filtering, lossy compression, or other attacks. Other types of watermarks exist, such as fragile or visible watermarks, however in this paper we shall generally use the term "watermark" to refer to imperceptible, robust watermarks.

Content protection [2]–[4] has been the most prevalent class of proposed applications for robust watermarking, as content owners are wary in an environment which permits the unauthorized reproduction and distribution of their copyrighted materials on a massive scale. The reader is encouraged to review tutorial and overviews in watermarking [5]–[9] for additional background.

There are three principal processes involved in robust watermarking: *watermark embedding*, *attack*, and *watermark detection*. In watermark embedding, a watermark is constructed and then embedded into an original signal to produce the watermarked signal. For security, watermark embedding usually requires knowledge of a secret embedding key. In addition, some watermarks also allow auxiliary information to be encoded in the watermark, known as the message or payload. Once the watermark has been embedded, the watermarked signal may be subjected to attack. There are many different types of attacks, including those which attempt to remove the watermark, make the watermark more difficult to detect, or subvert the security of the watermark. In watermark detection, a test signal is provided to the watermark detector. The test signal may be watermarked and possibly attacked, or may not have been watermarked at all. The watermark detector examines its input signal and reports whether the watermark is present or not, and if applicable, extracts the payload. If the watermark detector does not require access to the original (unwatermarked) signal, the watermarking technique is known as a blind technique.

One of the challenges in robust blind watermark detection is that of synchronization. Synchronization is the process of identifying the correspondence between the spatial and temporal coordinates of the watermarked signal and that of an embedded watermark, or "finding the watermark." During the watermark embedding process, the embedder inserts the watermark into the original signal with some chosen location, orientation, and scale. When the watermarked signal is attacked, the location, orientation, and scale of the embedded watermark may be altered. The goal of a geometric or synchronization attack, such as shifting, rotation, or cropping, is not to remove the watermark directly but force the detector to confront a more difficult synchronization problem [10], [11]. When the watermarked signal is presented to the detector, the detector must determine the location, orientation, and scale of the embedded watermark, which is the process of synchronization. Temporal synchronization attacks in video include frame dropping, insertion, transposition, and averaging (temporal interpolation or temporal scaling).

Synchronization is critical in robust video and audio watermark detection, even in the absence of a malicious attacker. In some applications, processing of a watermarked video by users requires robust synchronization for watermark detection. For example, aspect ratio conversion, frame rate conversion, and cropping are common video editing operations in broadcast video and motion picture applications. Other applications sub-

Manuscript received February 17, 2004. This work was supported by a grant from Digimarc. The associate editor coordinating the review of this paper and approving it for publication was Dr. Ton A. C. M. Kalker.

The authors are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-2035 USA (e-mail: linet@ecn.purdue.edu, ace@ecn.purdue.edu).

Digital Object Identifier 10.1109/TSP.2004.833866

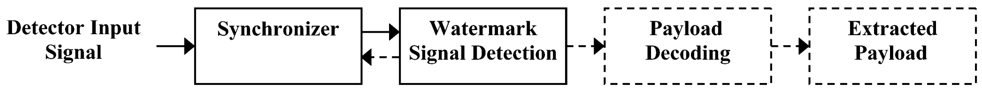


Fig. 1. Robust blind watermark detection with synchronization.

ject the watermarked signal in conditions which may damage parts of the signal and confuse the watermark detector. One example of such an application is streaming video and audio, where the watermarked signal may be damaged as it is transmitted over the network [12], [13]. Network congestion may also cause the watermarked signal to be lost for an indeterminate time. (Even a signal loss lasting 1 s can result in many frames being lost.) If the watermark detector loses synchronization, it is necessary for the detector to resynchronize prior to resuming detection. Lastly, blind watermark detectors must be able to synchronize when the input signal first becomes available to the detector. This process is known as initial synchronization. Initial synchronization is complicated by the fact that any portion of the watermarked signal may be input to the detector, not necessarily the "beginning" of the signal. A robust watermark detector should not rely on observing the "beginning" (or any other specific portion) of a watermarked signal for synchronization. A detection or synchronization scheme that relied on observing the "beginning" part of the watermarked signal would be vulnerable if that portion was cropped.

In this paper, we focus on the problem of temporal synchronization in the detection of video watermarks. We will describe a general model for video watermark embedding which models the construction of the watermark signal, and a model for watermark detection which generalizes the behavior of many video watermark detectors. A framework for efficient temporal synchronization is developed using the models. Spatial synchronization issues are not addressed by our models.

II. Background

An overview of robust blind watermark detection is shown in Fig. 1. The detector input signal is examined by the synchronizer, whose objective is to identify the correspondence (or mapping) between the coordinates of the input signal and that of an embedded watermark. The correspondence information is provided to the watermark signal detector. For example, one way that the watermark signal detector can make use of the synchronization information is to transform the watermark to some "normalized" coordinate system, and then perform signal detection on the normalized watermark. The synchronizer and watermark signal detection are not necessarily independent processes, and may be coupled.

One general approach to synchronization is that of naive search. In this approach, the watermark detector explicitly searches the space of coordinate transformations (or some subspace thereof) to locate the watermark. The search is performed without using any special properties of the watermark or other side information. Two examples of the naive search approach are the exhaustive search and the sliding correlator [14], [15]. Naive search is not practical for many applications because the cardinality of the set of all coordinate transformations (or subsets such as the set of affine transformations or the set of shifts) is an obstacle for computationally efficient search. It has also been shown that the exhaustive search is prone to false positives, which makes this approach questionable even if computational cost is not a constraint [16].

Another approach for synchronization is the construction of templates in the watermarked signal. A template is a pattern which describes the coordinates of the embedded watermark. A coordinate transformation applied to the watermarked signal, such as that arising from attack, will affect the template in the same manner as the embedded watermark. When the watermarked signal is provided to the detector, the synchronizer examines the template to determine the coordinates of the watermark. A template may be thought of as side information about the structure of the watermarked signal which the detector can use to reduce the computational search for synchronization.

Watermarking techniques have generally used three methods for constructing synchronization templates. The first method is the explicit embedding of an auxiliary synchronization signal into the watermarked signal, in addition to the watermark. Video watermarking techniques using explicit template embedding include [17]–[19], as well as the "helper watermarks" in [15] and the orthogonal sequences used for temporal synchronization in [20]. Disadvantages of this method are that the overall distortion in the watermarked signal is increased by the embedding of the synchronization signal, and the synchronization signal itself may be prone to estimation and attack [21]. Another method for constructing a template is to apply constraints on the structure of the watermark signal to generate the synchronization pattern, such as constructing a watermark with a tiled or periodic structure [22]. The structure of the watermark itself provides the template and no auxiliary synchronization signal is embedded. Unfortunately, the same constraints placed on the watermark signal to create the template reduces the capacity and security of the watermark. A second example of this method is [23]. The third method for obtaining a synchronization template is to use features of the original video signal [24]–[27] as the basis for synchronization. This method requires the synchronizer to reliably detect salient features from the original signal, and in contrast with the preceding methods, neither an explicit synchronization signal nor the watermark itself is used as the template.

Some watermarking techniques embed the watermark with invariance properties under transformations [19], [28], [29]. However, it may be difficult to find a transformation with invariance under some forms of attack, such as nonuniform scaling and spatial and temporal cropping. A similar approach is to design the watermark to be less sensitive to synchronization [30].

While spatial synchronization, particularly under affine transformations [31], has been explored for still image watermarks, temporal synchronization is a relatively unexplored area. The temporal dimension is unique from the spatial dimensions in video watermarking. First, there is often a high

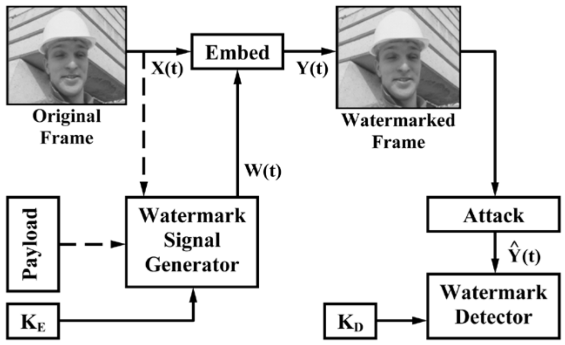


Fig. 2. Classical model of watermarking.

degree of temporal redundancy in successive frames of a video sequence, which can be a vulnerability. For example, if the embedded watermark in successive frames are uncorrelated, frame averaging may eliminate the watermark [32]. It was also demonstrated in [32] that embedding the same (fixed) watermark for all frames of the video can leave the watermark vulnerable to estimation attacks. Second, in some video watermarking applications, the detector has access only to a small portion of the video (a single frame, or a small number of frames) at any given time and may not be able to buffer entire video sequences or scenes. However, spatial synchronization techniques typically rely on having a large portion of the watermarked signal available for analysis.

III. Temporal Synchronization Framework

In this section, the foundation to define and examine issues in the temporal synchronization problem is developed, beginning from the classical model for watermarking. The classical model, shown in Fig. 2, is an elementary model for video watermark embedding and detection. First, a watermark is created and inserted into an original video sequence to produce the watermarked video. The watermarked video may be subjected to attack. Finally, the watermarked, and possibly attacked video is provided to the watermark detector.

The watermark embedder is provided three inputs: 1) the original video $\mathbf{X}$; 2) the embedding key K_E; and 3) payload M. The original video is an ordered sequence $\mathbf{X} = \langle X(0), X(1), X(2), \ldots, X(t), \ldots \rangle$,[1] with $X(t)$ corresponding to the frame displayed at time t. For convenience, t is discrete and expressed in units of frames, so t can also be referred to as the frame index. The first frame of the video is indexed by $t = 0$, the second frame by $t = 1$, and so on, with increasing indices ($t = 2, 3, 4, \ldots$) corresponding to video frames that are displayed later in time. The embedding key K_E is the secret information that is needed to create and embed the watermark signal. Embedding the watermark without knowledge of K_E should be very difficult. This provides security, as only users who know K_E will be able to embed the watermark. The payload M is auxiliary data that is encoded in the watermark signal. The detector, if it successfully detects the watermark, can extract the payload and provide it to the application. Not all watermarking techniques use a payload.

Watermark embedding is a two-step process. First, the watermark signal is constructed by the watermark signal generator. The watermark signal generator is provided with K_E and M, and produces $W(t)$, which is the watermark signal that will be inserted into the video. Some watermarking techniques also provide the original video $X(t)$ to the watermark signal generator for more effective watermark embedding. Providing the original video allows signal-adaptive or perceptual-based watermark embedding to reduce watermark visibility [1]. Watermark robustness can also be improved by considering the characteristics of the original video, which is the concept of *informed embedding* [6], [33]. Once the watermark signal is constructed, $W(t)$ is inserted into the original video frame $X(t)$ to produce the watermarked video frame $Y(t)$. The specific methods by which the watermark is constructed and embedded is dependent on the watermarking technique. The output of the embedder is the watermarked video $\mathbf{Y} = \langle Y(0), Y(1), \ldots, Y(t), \ldots \rangle$.

After the watermark has been embedded, the watermarked video may be attacked. There are many different types of attacks. Any process which causes the watermarked video to be altered is considered an attack, as altering the watermarked video may remove the embedded watermark or make the watermark more difficult to detect. Such attacks include filtering, lossy compression, addition of noise, and geometric attacks. Some other attacks, such as protocol attacks, attempt to subvert the security of the watermark and do not directly impact watermark detection. The primary focus of this discussion will be on temporal synchronization attacks, such as frame dropping, insertion of arbitrary frames, frame transposition, and frame averaging. These attacks alter the temporal structure of the watermarked video, which can confuse the watermark detector.

Let $\hat{\mathbf{Y}} = \langle \hat{Y}(0), \hat{Y}(1), \ldots, \hat{Y}(t), \ldots \rangle$ denote the watermarked and possibly attacked video that is provided to the detector. If the video has not been attacked, then $\hat{\mathbf{Y}}$ is identical to $\mathbf{Y}$ and $\hat{Y}(t) = Y(t)$ for all t. If the video is attacked then it is not necessarily true that $\hat{Y}(t) = Y(t)$ for any t. For example, a frame-dropping attack obtains $\hat{\mathbf{Y}}$ by removing frames from $\mathbf{Y}$, whereas a frame-insertion attack obtains $\hat{\mathbf{Y}}$ by inserting new, arbitrary frames into $\mathbf{Y}$.

The watermark detector examines its input video and determines if the watermark is present. The detector is also provided with the appropriate detection key K_D for detecting the watermark. For a symmetric (private key) watermark, the embedding key and its corresponding detection key are identical ($K_E = K_D$). Asymmetric (public key) watermarks [34] use distinct but related embedding and detection keys, similar in concept to public key cryptography [35]. Without the appropriate detection key, the watermark detector will not be able to detect the embedded watermark. Blind robust watermark detection was described in Section II. Fig. 2 shows the watermarked video $\hat{\mathbf{Y}}$ as the input to the watermark detector, however in the general case any video could be provided to the watermark detector. It is not necessary that the video examined by the detector is watermarked.

The classical model is a general model that is useful for an overview of watermarking, however, it does not provide insight

[1]In this text, the notation $\langle \cdot \rangle$ is used to denote an ordered sequence and $\{\cdot\}$ is used to denote set membership. $\mathbf{X} = \langle x_1, x_2, x_3 \rangle$ is the ordered sequence x_1 followed by x_2, followed by x_3. $\mathcal{X} = \{x_1, x_2, x_3\}$ is a set with members x_1, x_2, and x_3.

into the temporal synchronization problem because the structure of the watermark signal is not considered. We will now extend the discussion from the classical model, leading to new models for watermark embedding and detection. For the remainder of the discussion, it will be assumed that the watermarking technique is symmetric, or $K_E = K_D$.

When a watermark is embedded in a video sequence, one of the parameters that determines its structure is the embedding key K_E. In video watermarking, it is assumed that K_E is used to create a *key schedule* or *key sequence* $\mathbf{K} = \langle K(0), K(1), \ldots, K(t), \ldots \rangle$, which is the ordered sequence of subkeys used for generating the watermark embedded in the individual frames of $\mathbf{Y}$. Specifically, $K(t)$ is the subkey used for generating the watermark signal embedded in frame $Y(t)$. K_E and $K(t)$ are assumed to be members of the key space $\mathcal{K}$ whose cardinality (denoted by $|\mathcal{K}|$) is very large. For security reasons, $|\mathcal{K}|$ is assumed to be sufficiently large that a computational (exhaustive) search over $\mathcal{K}$ is infeasible, preventing an exhaustive search to find K_E or $K(t)$. These assumptions hold for many video watermarking techniques, including [14], [19], [22], [32], [36]. For example, it is not uncommon for a watermarking technique to use K_E as the seed of a pseudorandom number generator which produces the watermark signal. For these techniques, $K(t)$ corresponds to the internal state of the random number generator when frame $X(t)$ is watermarked. Some other watermarking techniques embed the same watermark signal in each frame of video, and for these techniques $K(t) = K_E$ for all t.

Let the ordered sequence $\hat{\mathbf{K}} = \langle \hat{K}(0), \hat{K}(1), \ldots, \hat{K}(t), \ldots \rangle$ denote the keys used to produce the watermark signals embedded in the frames of $\hat{\mathbf{Y}}$, where $\hat{K}(t)$ is the key used to generate the watermark in frame $\hat{Y}(t)$. If the watermarked video has not been attacked, then $\hat{Y}(t) = Y(t)$ and thus $\hat{K}(t) = K(t)$ for all t. However, attacks such as frame dropping, insertion, and transposition alter the sequence of frames in the video, causing changes to $\hat{K}(t)$. Under these temporal synchronization attacks, it is not necessary that $\hat{K}(t) = K(t)$. Signal processing attacks that do not affect the temporal structure of the watermarked signal, such as spatial filtering, affect $\hat{\mathbf{Y}}$ but not $\hat{\mathbf{K}}$. $\hat{K}(t)$ is not defined if $\hat{Y}(t)$ is an arbitrary, unwatermarked frame that has been inserted into the video.

The objective for the watermark detector is to determine $\hat{K}(t)$ when frame $\hat{Y}(t)$ is examined. (A blind detector does not have access to $Y(t)$.) If the detector can determine $\hat{K}(t)$, temporal synchronization is achieved. If the detector cannot determine $\hat{K}(t)$, temporal synchronization is lost. The process of finding $\hat{K}(t)$ when frame $\hat{Y}(t)$ is examined by the detector is temporal synchronization. However, if an attacker can determine $\hat{K}(t)$ for any frame $\hat{Y}(t)$, he will have effectively deduced the structure of the watermark and the watermark security is broken. Unlike the watermark detector, it is assumed that the attacker does not have prior knowledge of K_E.

The key schedule can have a dramatic effect on the ease of synchronization and the security of the watermark. To illustrate the effects of the key schedule on synchronization and security, three classes of watermarks with special key sequences are discussed: the time-invariant key, time-independent key, and periodic key watermarks.

A time-invariant key watermark uses the same key to construct and embed the watermark into each frame of the video. For these watermarks, temporal synchronization is a simple matter because it is known that $K_E = \hat{K}(t) = K(t)$ for any watermarked frame $\hat{Y}(t)$ No search is need to determine $\hat{K}(t)$. However, time-invariant key watermarks may be vulnerable to estimation attack [32], and an attacker that successfully obtains the watermark key for a single frame of the video breaks the security of the watermark.

In a time-independent key watermark, the keys used to construct and embed the watermark signal in successive video frames are nearly independent. Strictly speaking, the watermark keys $K(t_1)$ and $K(t_2)$, $t_1 \neq t_2$, are not truly independent because a single key K_E is used to produce all the watermarking keys in the key schedule. The keys in the time-independent key schedule, however, do not repeat or repeat with an extremely long period. Such a key schedule may be produced by using a pseudorandom number generator to create the watermark signal for each frame, whose internal state is never reset after seeding with K_E. Knowledge of the key used to embed the watermark in one frame yields little information about the key used to embed the watermark in other frames, however, the detector must generally search $\mathcal{K}$ to find $\hat{K}(t)$ when synchronization is lost. Watermark robustness may also be an issue [32].

In a time-periodic key watermark, the keys of the key schedule form a repeating sequence (with relatively short period). In such a key schedule, every $K(t)$ (and thus every $\hat{K}(t)$) is a member of a small set of keys $\mathcal{K}' \subset \mathcal{K}$ (with $|\mathcal{K}'| \ll |\mathcal{K}|$) that are repeated. A search $\mathcal{K}'$ may not be infeasible. Moreover, resynchronization is possible by looking for any single key $K' \in \mathcal{K}'$ because K' will eventually appear in the key schedule in some future frame. However, the security of a periodic key schedule is not as strong as a time-independent key schedule. For example, the attacker can obtain the key sequence for the entire watermarked video by obtaining the key sequence over a single period. Estimating the period of the watermark key sequence may be possible by correlation, and vector quantization or clustering techniques may be used to estimate the embedded watermark from $\hat{\mathbf{Y}}$.

The time-invariant key, independent key, and periodic key sequences differ in the amount of *temporal redundancy* in the key schedule. Temporal redundancy in the key schedule refers to the degree of "randomness" of the keys in the key schedule. The sequence of keys appearing in a key schedule with low temporal redundancy (such as the time-independent key schedule) is highly random and recovering synchronization may require a search over $\mathcal{K}$. A key schedule with greater temporal redundancy allows the detector to reduce the search needed for synchronization. For example, the detector may be able to predict likely values for an unknown $\hat{K}(t)$, based on the past keys $\hat{K}(t-1), \hat{K}(t-2), \ldots$. However, there is a security tradeoff in that greater temporal redundancy in the key schedule can make deducing $\hat{K}(t)$ or estimating the embedded watermark easier for an attacker. Temporal redundancy shall be revisited in the design of key schedules that are resistant to frame dropping and transposition attacks in Section V. In Section IV, a model for watermarking is introduced which encompasses the time-invariant key, time-independent key, and periodic key watermarks as special cases.

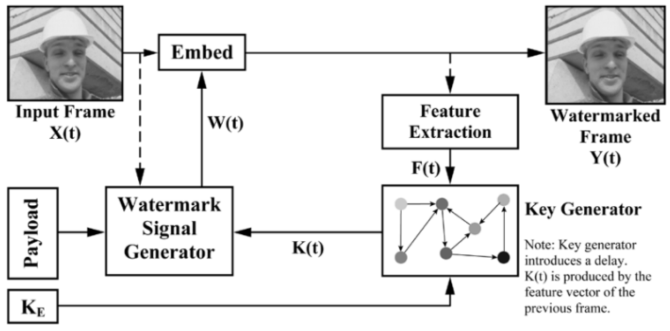

Embed
Input Frame X(t)
W(t)
Feature Extraction
Watermarked Frame Y(t)
F(t)
Payload
Watermark Signal Generator
K(t)
Key Generator
Note: Key generator introduces a delay. K(t) is produced by the feature vector of the previous frame.
K_E

Fig. 3. Watermark embedding model.

IV. Watermark Embedding and Detection Models

A. *Model for Watermark Embedding*

To model the construction of the key schedule, the classical watermark embedder is extended as shown in Fig. 3. (The attack and watermark detector are omitted.) The function of the watermark signal generator and watermark signal embedder are identical to that of the classical model. Any watermark construction and embedding techniques may be used, subject to the assumptions stated in the preceding discussion. The key generator and the feature extraction are new components, and will be described in detail below.

The overall steps for watermark embedding are as follows, for each frame of the video.

Watermark Embedding Procedure (WEP)

1) The key generator provides $K(t)$, the key used to watermark the current frame, to the watermark signal generator.
2) The watermark signal generator uses $K(t)$ to produce $W(t)$, the watermark signal to be embedded into the current frame. For example, $K(t)$ may be used to seed a pseudorandom number generator which produces $W(t)$. Like the classical model, the watermark signal may be dependent on the payload M, as well as the current video frame $X(t)$.
3) The watermark embedder embeds $W(t)$ into the original video frame $X(t)$ to produce the watermarked video frame $Y(t)$.
4) The feature extractor examines the watermarked video frame and produces a feature vector $F(t)$.[2]
5) The key generator generates the watermark key for the next frame, $K(t+1)$. To generate $K(t+1)$, the key generator uses K_E and $F(t)$. Return to step 1 for the next frame of the video.

The purpose of the key generator is to produce the watermarking key for each frame of the video, and thus, produce the key schedule for the watermark. The key generator is modeled as a state machine (SM) which accepts K_E and $F(t)$ as inputs. The SM is a general computational model for systems that possess memory (state) [37]. SMs have been used in a variety of applications, including character string recognition and regular-expression matching [38], [39], theoretical computing (such as Turing Machines) [40], [41], and simple "artificial intelligence" engines used in computer entertainment [42].

A SM is defined by the tuple $(\mathcal{S}, \mathcal{S}_0, \mathcal{I}, \mathcal{O}, \phi, \lambda)$, where $\mathcal{S}$ is the set of states, $\mathcal{S}_0$ is the set of initial states, $\mathcal{I}$ is the input domain, $\mathcal{O}$ is the output range, ϕ is the state transition function, and λ is the output function. $\mathcal{S} = \{s_0, s_1, \ldots, s_{|\mathcal{S}|-1}\}$ is the nonempty, countable set of states in the SM. A "state" is a representation of memory, and allows the output and behavior of the SM to depend on current and past inputs. At any given time $t \geq 0$ the state of the machine, known as the current state $s(t)$, is one of the members of $\mathcal{S}$.[3] The current state of the SM can change in response to the SM input in accordance to the state transition function. $\mathcal{S}_0$ is the set of initial states, and is a nonempty subset of $\mathcal{S}$. The current state of the SM is initialized to a member of $\mathcal{S}_0$ when the machine is started, such that $s(0) \in \mathcal{S}_0$. For the time being, it is assumed that $\mathcal{S}_0$ is a set with only one element, deferring the discussion when $|\mathcal{S}_0| > 1$. $\mathcal{I}$ is the input domain, which is the set of all possible inputs to the SM, and $\mathcal{O}$ is the output range, which is the set of all possible outputs of the SM. The state transition function $\phi : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{S}$ describes the state transitions of the SM. If the current state of the SM is $s(t) \in \mathcal{S}$, and the current input is $i(t) \in \mathcal{I}$, then the next state will be $s(t+1) = \phi(s(t), i(t))$. The output function $\lambda : \mathcal{S} \rightarrow \mathcal{O}$ is a mapping from each state to an output value. The output of the SM is $\lambda(s(t)) \in \mathcal{O}$, which is a function of only the current state $s(t)$. The output function of some SMs (known as Mealy Machines [37]) are more general, in that λ is also a function of the current input ($\lambda : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{O}$); however, it is sufficient for the output to depend only on $s(t)$ herein. A SM is often represented using a directed graph, where the vertices corresponds to the states and the edges correspond to the state transition function $\phi(\cdot)$.

When a SM is used as a key generator, it accepts fixed K_E and frame-dependent feature vector $F(t)$ as inputs and produces a key value as output. Thus, the input domain $\mathcal{I}$ will generally be the Cartesian product of the key space $\mathcal{K}$ and the range of all possible feature vectors from the feature extractor. The output range is the key space $\mathcal{O} = \mathcal{K}$. The starting state may depend on K_E. In step 1 of the WEP, the SM outputs $\lambda(s(t))$. State transitions occur in step 5, where $s(t+1) = \phi(s(t), K_E, F(t))$. After the state transition, the key $K(t+1)$ is obtained by $\lambda(s(t+1))$. The output mapping function $\lambda(\cdot)$ of a key generator must also be one-to-one.

The feature extractor examines each watermarked video frame $Y(t)$ and produces a feature vector $F(t)$. The feature vector is then provided to the key generator, which allows the key sequence to be video-dependent. A video-dependent key schedule can increase the difficulty of inverting the watermark and provide some benefit against ownership [43] and copy [44] attacks. However, the disadvantage of using features is that the dependence of the key schedule on the features imply that temporal synchronization can be destroyed by performing attacks that change the feature vectors. Feature extraction may be omitted if the key generator does not use $F(t)$. For robust video watermarking, the features should be chosen such that $F(t)$ changes in value

[2]Strictly speaking, $F(t)$ is a vector. However, it is a vector obtained from observing a single frame, $Y(t)$. $F(t)$ is not written as $\mathbf{F}(t)$ to avoid confusion with $\mathbf{X}$, $\mathbf{Y}$, $\hat{\mathbf{Y}}$, and other symbols whose elements form an ordered sequence across time (i.e., over multiple frames of video).

[3]Note the difference between the notation s_i and $s(t)$. s_i refers to a specific member of $\mathcal{S}$ (the ith state) and is independent of time. $s(t)$ is the current state of the SM at time t.

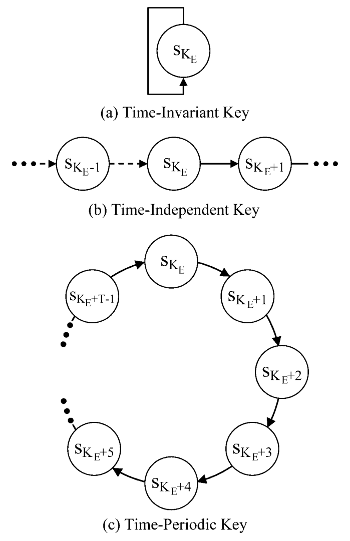


Fig. 4. Example key generators.

only when significant alterations are made to the watermarked frame $Y(t)$. Ideally, the features should be sufficiently robust so that such an attack is successful only when the attacked video no longer has any value in the application. For security, the features should be dependent on an auxiliary key. Lastly, features should also be computationally efficient to obtain.

Before describing the watermark detector, examples of the time-invariant key, time-independent key, and time-periodic key watermark embedders are shown to illustrate these key sequences as special cases of the model. These examples are not unique, and there are other key generator configurations which can produce these special key schedules. None of the watermark embedders for these three key schedules use feature extraction. Assume the key space is $\mathcal{K} = \{0, 1, 2, \ldots, |\mathcal{K}| - 1\}$ and the embedding key is $K_E \in \mathcal{K}$. With the exception of the state transition function, the SMs for these examples are common: $\mathcal{S} = \{s_0, s_1, \ldots, s_{|\mathcal{K}|-1}\}$, $\mathcal{S}_0 = \{s_{K_E}\}$, $\mathcal{I} = \mathcal{K}$, $\mathcal{O} = \mathcal{K}$, and $\lambda(s_i) = i$.

The key generator for a time-invariant key watermark has the state transition function

$$\phi(s_i, K_E) = s_i. \tag{1}$$

For a time-independent key watermark, the key generator has state transition function

$$\phi(s_i, K_E) = \begin{cases} s_{i+1}, & s_i \in \{s_0, s_1, \ldots, s_{|\mathcal{K}|-2}\} \\ s_0, & s_i \in \{s_{|\mathcal{K}|-1}\}. \end{cases} \tag{2}$$

For a periodic key watermark with period $T > 1$ frames, the key generator has state transition function

$$\phi(s_i, K_E) = \begin{cases} s_{i+1}, & s_i \in \{s_{K_E}, s_{K_E+1}, \ldots, s_{K_E+T-2}\} \\ s_{K_E}, & \text{otherwise} \end{cases} \tag{3}$$

for $K_E \in \{0, 1, \ldots, |\mathcal{K}| - T\}$.[4] The graph representation of these SMs is shown in Fig. 4.

[4]The state transition function for $K_E \in \{|\mathcal{K}| - T + 1, \ldots, |\mathcal{K}| - 1\}$ is conceptually similar to (3), but requires the indices of the states in the period to "wrap-around." There is no state $s_{|\mathcal{K}|}$ in $\mathcal{S}$, so the next state after $s_{|\mathcal{K}|-1}$ is s_0.

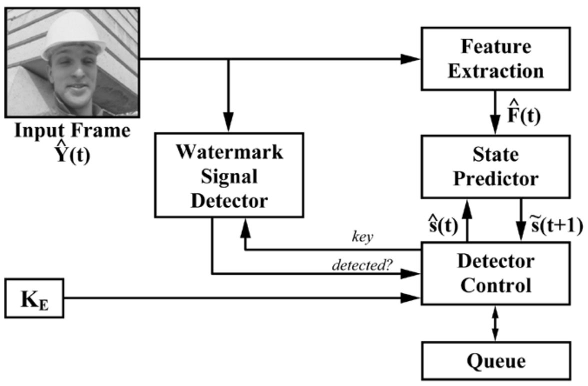


Fig. 5. Watermark detection model.

Nondeterministic Key Generators: The key generator may also be modeled using a nondeterministic SM. A nondeterministic SM may have one or both of the following: Multiple starting states, and nondeterministic state transitions. A SM with multiple starting states has $|\mathcal{S}_0| > 1$. When the SM is initialized at $t = 0$, a random member of $\mathcal{S}_0$ is chosen as the starting state. A SM with nondeterministic state transitions has a state transition function $\phi(s, i)$ which returns a nonempty set of possible next states. When a state transition occurs, the current state of the SM transitions to a randomly chosen member of that set.

B. Model for Watermark Detection

This section describes a model for symmetric $(K_E = K_D)$ blind video watermark detection. The detector model is shown in Fig. 5. The inputs to the detector are the video sequence to be examined $\hat{\mathbf{Y}}$ and the detection key. It was assumed that the watermarking technique is symmetric, so the detection key is K_E. The major components of the detector model are the watermark signal detector, feature extractor, state predictor, detector control, and the queue. The detector has knowledge of the embedder's key generator $(\mathcal{S}, \mathcal{S}_0, \mathcal{I}, \mathcal{O}, \phi, \lambda)$ and feature extraction.

The watermark signal detector is identical to that of the classical model. It is the watermark detector corresponding to the technique used to embed the watermark signal. The watermark signal detector is given $\hat{Y}(t)$ and a detection key, and determines whether or not the watermark was detected in the frame using the key. The detection key is obtained from the detector control. The watermark signal detector may be invoked multiple times for a single frame of video.

The feature extraction is identical to that of the embedder. The detector's feature extractor examines the input frame $\hat{Y}(t)$ and produces a feature vector $\hat{F}(t)$, which is provided to the state predictor.

The state predictor and the queue are the significant components for watermark detection and synchronization. At this time, the structure of the the state predictor and the queue are described. The underlying synchronization mechanism will be discussed later. The state predictor accepts a state input $\hat{s}(t)$, the embedding key K_E, and feature input $\hat{F}(t)$ and outputs $\tilde{s}(t+1)$, the predicted state for $\hat{s}(t+1)$. The prediction function is identical to the state transition function of the watermark embedder: $\tilde{s}(t+1) = \phi(\hat{s}(t), K_E, \hat{F}(t))$. The state predictor is not a SM.

The queue serves as the detector's memory. A queue is an array of elements $Q = \langle q(0), q(1), \ldots, q(|Q|-1) \rangle$, where each element may contain some data. The queue size $|Q|$ is the number of elements present in the queue and the queue capacity is the maximum size of the queue. The element $q(0)$ is known as the head, and $q(|Q|-1)$ is known as the tail. When new data is inserted into the queue, all of the data in the queue shift one element and the new data is placed at the head of the queue. If the queue was full prior to the insertion, then any data that resided at the tail of the queue is lost. This type of queuing behavior is known as a first-in-first-out (FIFO) queue. In addition to the insertion of new data, the queue supports moving or "promoting" any data present in the queue to the head of the queue. When data in a specific queue element is promoted, all the other data in the queue are moved—preserving their relative order—to make room at the head of the queue and the promoted data is placed into the head. For the watermark detector, the data stored in the queue will be members of $\mathcal{S}$. Prior to receiving the first frame of video, the watermark detector queue is empty.

The detector control component manages the operation of all the components of the watermark detector, performing the steps of the watermark detection procedure (WDP), for each frame $\hat{Y}(t)$.

Watermark Detection Procedure (WDP)

1) The input frame is provided to the watermark signal detector and the feature extractor. The feature extractor examines $\hat{Y}(t)$ and obtains $\hat{F}(t)$.
2) For each state $s_{\text{init}} \in \mathcal{S}_0$, the detector sends the key $\lambda(s_{\text{init}})$ to the watermark signal detector. If the detector finds the watermark using $\lambda(s_{\text{init}})$, let $\hat{s}(t) = s_{\text{init}}$ and continue to step 5. If the detector fails to find the watermark using the keys of all states in $\mathcal{S}_0$, continue to the next step.
3) For each state $s_q \in Q$, the detector sends the key $\lambda(s_q)$ to the watermark signal detector. If the detector finds the watermark using s_q, let $\hat{s}(t) = s_q$, promote s_q to the head of the queue, and continue to step 5. If the detector fails to find the watermark using the keys of all states in the queue, continue to the next step.
4) At this step, either $\hat{Y}(t)$ is not watermarked or temporal synchronization failed for the current frame. Return to step 1, and try synchronizing with the next frame, $\hat{Y}(t+1)$.
5) At this step, $\hat{K}(t)$ was found and temporal synchronization is achieved for the current frame. $\hat{s}(t)$ is the state which maps to $\hat{K}(t)$ under $\lambda(\cdot)$. Use the state predictor to predict the state $\tilde{s}(t+1) = \phi(\hat{s}(t), K_E, \hat{F}(t))$. If the predicted $\tilde{s}(t+1)$ is already in the queue, promote it to the head. Otherwise, insert the predicted $\tilde{s}(t+1)$ into the queue. Then return to step 1 for the next frame of video. If $\phi(\cdot)$ returns a set of states, then this step is repeated for every member of $\phi(\hat{s}(t), K_E, \hat{F}(t))$, before returning to step 1.

A glance at the WDP shows that the queue is used to perform a limited search to find the appropriate detection key for each frame of the video. The queue stores the states which produced recently detected keys, and their next states. The next states are obtained by using the state predictor. A more detailed analysis of the synchronization mechanism is presented in Section IV-C.

The watermark detection model generalizes the behavior of symmetric (private key) video watermark detectors, analogous to the watermark embedding model described in Section IV-A generalizing video watermark embedders. Many video watermark detectors perform the steps of the WDP in principle. It is known that the detector possesses side-information regarding the watermark embedding process. This side-information generally allows the detector to generate the watermark signal (step 3 of the WDP) and then detect for the presence of the watermark signal in the input video frame. If the detector discovers that $\hat{K}(t)$ was the correct key to watermark frame $\hat{Y}(t)$, it assumes (i.e., predicts) $\hat{K}(t+1)$ by using this side-information (step 5 of the WDP) and searches for this key in subsequent frames of the video. Watermark detection continues in such a manner until the detector becomes confused by a synchronization attack. When the detector loses synchronization, and for initial synchronization, the detector searches for a particular key to resynchronize (step 2 of the WDP).

C. Analysis

The function of the embedder and detector models are examined in this section. It is shown that the detector will detect the watermark embedded in every frame of the watermarked video if it has not been attacked and that the insertion of arbitrary (unwatermarked) frames will not desynchronize the detector. The vulnerability of the watermark detector in the presence of frame dropping or transposition attack is also shown.

The objective of the detector is to determine $\hat{K}(t)$ when frame $\hat{Y}(t)$ is examined. However, for a watermark produced by the watermark embedder discussed in Section IV-A, $\hat{K}(t) = \lambda(\hat{s}(t))$, where $\hat{s}(t)$ is the underlying state of the key generator which produced $\hat{K}(t)$. The function $\lambda(\cdot)$ is one-to-one, and is only a function of the state. Thus, the objective of the detector can be restated as to determine $\hat{s}(t)$ when frame $\hat{Y}(t)$ is examined. The analysis will focus on the states, specifically the states of the embedder's key generator to produce the watermark and the states in the detector's queue.

It is assumed that the watermark signal detector is always correct for any key and input video frame (no false positives and no misses). Also, the analysis shall assume that the key generator of the watermark embedder uses a deterministic SM. This is to simplify the presentation; a similar analysis can be performed for nondeterministic SMs, which would be complicated by $\phi(\cdot)$ returning a set of states, as opposed to a single state.

Some lemmas about the properties of the embedder and detector models are listed below. Their proofs are in the Appendix.

Lemma 1 (Fundamental Watermark Structure): Let $Y(t)$ and $Y(t+1)$ be any two successive frames of the watermarked video. Assume that $K(t) = \lambda(s(t))$ produced the watermark signal embedded in $Y(t)$ and $K(t+1) = \lambda(s(t+1))$ produced the watermark signal embedded in $Y(t+1)$. Assume that $F(t)$ is obtained from the feature extractor from $Y(t)$, and K_E is constant. Then, $s(t+1) = \phi(s(t), K_E, F(t))$.

Lemma 2: Suppose $\hat{Y}(t)$ and $\hat{Y}(t+1)$ are two successive watermarked frames of the detector's input video and neither frame is attacked or involved in an attack. Assume that $\hat{K}(t) = \lambda(\hat{s}(t))$ produced the watermark signal embedded in $\hat{Y}(t)$ and

$\hat{K}(t+1) = \lambda(\hat{s}(t+1))$ produced the watermark signal embedded in $\hat{Y}(t+1)$. Assume that $\hat{F}(t)$ is obtained from the detector's feature extractor at time t, and K_E is constant. Then, $\hat{s}(t+1) = \phi(\hat{s}(t), K_E, \hat{F}(t))$.

Lemma 3 (Detector Behavior): Let $\hat{Y}(t)$ be the frame of the video examined by the detector. Then, temporal synchronization is achieved if and only if $\hat{Y}(t)$ is watermarked, and $\hat{s}(t)$ is either a member of $\mathcal{S}_0$ or in the queue at time t, where $\hat{s}(t)$ is the state which produced $\hat{K}(t) = \lambda(\hat{s}(t))$.

Lemma 4 (Initial Synchronization): Let $\hat{Y}(t)$ be the frame of the video examined by the detector, watermarked by $\hat{K}(t) = \lambda(\hat{s}(t))$ and $\hat{s}(t) \in \mathcal{S}_0$. Temporal synchronization will be achieved.

Lemma 5 (Queue Lemma I): Let $\hat{Y}(t)$ be the watermarked frame examined by the detector, where $\hat{K}(t) = \lambda(\hat{s}(t))$ produced the watermark embedded in the frame. Suppose temporal synchronization is achieved. Then, for frame $\hat{Y}(t+1)$, the state at $q(0)$ (the head of the queue) is $\phi(\hat{s}(t), K_E, \hat{F}(t))$, and if $\hat{s}(t) \notin \mathcal{S}_0$, the state at $q(1)$ is $\hat{s}(t)$.

Lemma 6 (Queue Lemma II): Let $\hat{Y}(t)$ be the frame of the video examined by the detector. Suppose temporal synchronization is not achieved. Then the queue is unchanged.

Theorem 1 (Correctness of Watermark Detector With no Attacks): Let $\mathbf{Y}$ be the watermarked video provided to the detector. Suppose no attacks are performed on $\hat{\mathbf{Y}}$. Then the watermark detector will achieve synchronization and detect the watermark in every frame of the video.

Proof: The proof will be by induction on t. Base Case: $\hat{Y}(0)$ is the first frame of the video. There are no attacks, so $\hat{Y}(0) = Y(0)$, which was watermarked by the key generated by state $s(0) \in \mathcal{S}_0$. By Lemma 4, temporal synchronization will be achieved.

Inductive Case: Suppose temporal synchronization is achieved for frame $\hat{Y}(t)$. By Lemma 5, $\phi(\hat{s}(t), K_E, \hat{F}(t))$ will be at the head of the queue. By Lemma 2, the state for the next frame is $\hat{s}(t+1) = \phi(\hat{s}(t), K_E, \hat{F}(t))$. Thus, by Lemma 3, temporal synchronization will succeed for frame $\hat{Y}(t+1)$. ■

Theorem 2 (Ineffectiveness of Frame Insertion Attack): Suppose $Y(u)$ and $Y(u+1)$ are consecutive frames of the watermarked video, and $\hat{Y}(t) = Y(u), \hat{Y}(t+N+1) = Y(u+1), N > 0$, and all frames $\hat{Y}(t+i), i \in \{1, 2, \ldots, N\}$ are arbitrary, unwatermarked frames inserted as an attack, and temporal synchronization is achieved for frame $\hat{Y}(t)$. Then, temporal synchronization will be achieved for frame $\hat{Y}(t+N+1)$.

Proof: $\hat{Y}(t) = Y(u)$ implies $\hat{K}(t) = K(u)$, $\hat{s}(t) = s(u)$, and $\hat{F}(t) = F(u)$. $\hat{Y}(t+N+1) = Y(u+1)$ implies $\hat{K}(t+N+1) = K(u+1)$, $\hat{s}(t+N+1) = s(u+1)$. Lemma 1 applies to $s(u)$ and $s(u+1)$, thus $s(u+1) = \phi(s(u), K_E, F(u))$. And thus, $\hat{s}(t+N+1) = \phi(\hat{s}(t), K_E, \hat{F}(t))$.

Temporal synchronization on frame $\hat{Y}(t)$ is achieved, so by Lemma 5, the state $\phi(\hat{s}(t), K_E, \hat{F}(t))$ will be at the head of the queue. The next N (inserted) frames of $\hat{\mathbf{Y}}$ after $\hat{Y}(t)$ are all unwatermarked, which implies that temporal synchronization will not succeed for those frames (Lemma 3). By Lemma 6, however, the queue is not changed during the processing of any of the N frames and $\phi(\hat{s}(t), K_E, \hat{F}(t))$ will remain at the head of the queue. Thus, by Lemma 3, temporal synchronization will be achieved for frame $\hat{Y}(t+N+1)$. ■

Theorem 3 (Vulnerability to Frame Dropping): Suppose $Y(u)$, $Y(u+1)$, and $Y(u+2)$ are consecutive frames of the watermarked video, watermarked using keys $K(u) = \lambda(s(u))$, $K(u+1) = \lambda(s(u+1))$, and $K(u+2) = \lambda(s(u+2))$, respectively. Also assume $s(u) \neq s(u+1) \neq s(u+2)$, and $s(u+2) \notin \mathcal{S}_0$. Suppose $\hat{Y}(t) = Y(u)$, but the next frame is dropped so that $\hat{Y}(t+1) = Y(u+2)$. Suppose $\hat{s}(t+1)$ is not in the detector's queue at time t. Then, temporal synchronization will not succeed for frame $\hat{Y}(t+1)$, even if temporal synchronization succeeded for frame $\hat{Y}(t)$.

Proof: $\hat{Y}(t) = Y(u)$ implies $\hat{K}(t) = K(u)$ and $\hat{s}(t) = s(u)$. Likewise, $\hat{Y}(t+1) = Y(u+2)$ implies $\hat{K}(t+1) = K(u+2)$ and $\hat{s}(t+1) = s(u+2)$. If temporal synchronization is not achieved for frame $\hat{Y}(t)$, by Lemma 6, the queue will not change when the detector examines frame $\hat{Y}(t)$. Since $\hat{s}(t+1)$ was not in the queue at time t, then $\hat{s}(t+1)$ will not be in the queue when $\hat{Y}(t+1)$ is examined. Furthermore, since $\hat{s}(t+1) \notin \mathcal{S}_0$, by Lemma 3 temporal synchronization will not be achieved for frame $\hat{Y}(t+1)$.

Suppose temporal synchronization succeeded for frame $\hat{Y}(t)$. Then by Lemma 5, state $\tilde{s} = \phi(\hat{s}(t), K_E, \hat{F}(t))$ and possibly state $\hat{s}(t)$ will be in the queue. However neither $\hat{s}(t) = s(u)$ nor $\tilde{s} = s(u+1)$ is equal to $\hat{s}(t+1) = s(u+2)$ since $s(u) \neq s(u+1) \neq s(u+2)$. It was assumed that state $\hat{s}(t+1)$ was not in the queue at time t, thus when the detector examines frame $\hat{Y}(t+1)$, the state $\hat{s}(t+1)$ will not be in the queue. Since $\hat{s}(t+1) = s(u+2) \notin \mathcal{S}_0$, by Lemma 3 temporal synchronization will not be achieved for frame $\hat{Y}(t+1)$. ■

From the proof of Theorem 1, it can be seen that the $\phi(\cdot)$ function induces a "chain of states" that is created by the embedder (through the state transitions in the key generator) and traced by the detector (using the queue and state predictor). The frame-dropping attack severs the chain, causing the watermark detector to be unable to discover the state sequence used by the embedder. A frame transposition attack can affect the state sequence similarly. In Section V, watermark detection in the presence of frame dropping and transposition attack shall be addressed by adding temporal redundancy into the key schedule.

When the detector loses temporal synchronization, Lemma 3 shows why recovering synchronization is relatively simple for time-invariant key and time-periodic key watermarks, and difficult for time-independent key watermarks. For a time-invariant key watermark, the state of the key generator for all time is $s(t) = \hat{s}(t) = s_{K_E} \in \mathcal{S}_0$. Thus, synchronization will always succeed for any watermarked frame in a time-invariant key watermark, even without a queue. For a periodic key watermark, the state sequence includes $s_{K_E} \in \mathcal{S}_0$ in each period. If the detector loses synchronization, it will be able to recover synchronization at a frame which $\hat{s}(t) = s_{K_E}$, or a frame which $\hat{s}(t) \in Q$ in a future period. However, the situation is much different for a time-independent key watermark. If the detector loses synchronization, a frame that is watermarked by the state s_{K_E} or a state in the detector's queue may not appear until nearly $|\mathcal{K}|$ frames in the future. Resynchronization for a time-independent key watermark by search alone may not be practical, and synchronization may require embedding additional information into the watermarked signal (such as an explicit synchronization signal) to describe $\hat{K}(t)$.

V. Temporal Redundancy and Synchronization

It was shown in Section IV-C that the watermark detector may be vulnerable to frame dropping and transposition attacks. However, the resilience of the watermark detector against these attacks may be increased by adding temporal redundancy into the key schedule. Temporal redundancy adds robustness into the key schedule, permitting some frames to be dropped or re-ordered without adversely affecting temporal synchronization. A modified embedder is described which adds temporal redundancy by watermarking multiple frames of the video with an identical key.

One strategy for increasing the robustness of the watermark is to limit the effect of synchronization loss to as few frames as possible. If the detector can recover synchronization quickly, then the effect of losing synchronization is not severe. An example of this strategy is the time-periodic key watermark. If the detector loses synchronization, then a frame that allows the detector to recover synchronization should be examined by the detector in the near future.

The embedder's key generator can be modified to produce a key schedule that is similar to, but not necessarily identical to a time-periodic key watermark. The modification entails "resetting" the key generator after α consecutive frames of the video have been watermarked. When the key generator is reset, its current state is set to a member of $\mathcal{S}_0$, similar to the initialization which occurs before $t = 0$. The parameter α is the *period*, although the name is somewhat a misnomer because the watermark state sequence is not necessarily strictly periodic. The larger α, the less often the key generator is reset and the key schedule produced by the key generator has less temporal redundancy. It is also possible to reset the key generator at random intervals, where α is the expected number of frames between resets. This change does not require any modification to the watermark detector, nor is it necessary to provide α to the detector as side-information because temporal synchronization will succeed when a frame is watermarked with a state $\hat{s}(t) \in \mathcal{S}_0$. A frame watermarked when the key generator state is a member of $\mathcal{S}_0$ is known as a *resynchronization frame*, because these frames allow the detector to recover synchronization if it is desynchronized as well as initial synchronization.

Theorem 3 shows that the loss of a single state (frame) can cause the detector to lose synchronization, indicating the fragileness of the state sequence. Another strategy for increasing the robustness of the watermark is to find a means for protecting the state sequence so that the loss of individual states can be tolerated without synchronization loss. Consider a modified embedder in which the key generator does not change its state after every frame of the video is watermarked. In the modified embedder, the current state of the key generator changes to $\phi(s(t), K_E, F(t))$ only after β consecutive frames are watermarked, using the feature vector of the last frame to determine the next state. The current state of the SM remains unchanged and the feature vector is ignored for all other frames during step 5 of the WEP. The parameter β is known as the *repeat* parameter. The resulting state sequence resembles that of Fig. 6.

It will now be shown that having consecutive frames watermarked using the same key generator state will not destroy temporal synchronization at the detector. Because of this property,

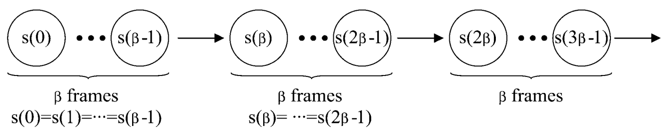


Fig. 6. State sequence of modified embedder. Arrows indicate frames where state transitions in the key generator occur.

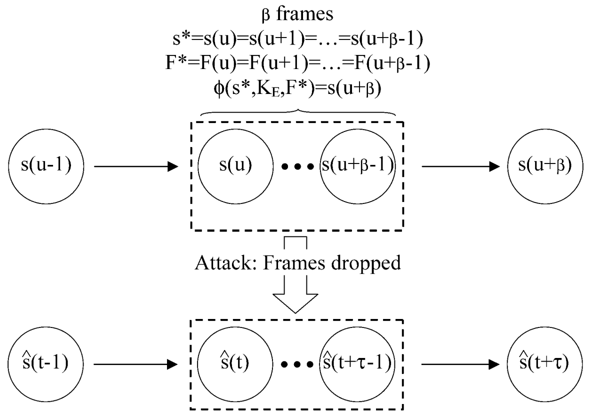


Fig. 7. State sequence in a frame-dropping attack.

it is not necessary to modify the watermark detector to accommodate the change made to the embedder, or to provide β to the detector as side-information. Also, if the feature vectors are identical over those frames, the detector's queue will not change over those frames (proofs are in the Appendix.)

Lemma 7 (Queue Lemma III): Suppose $\hat{Y}(t)$ and $\hat{Y}(t+1)$ are two consecutive frames watermarked using the same key $\lambda(\hat{s}(t)) = \lambda(\hat{s}(t+1))$ and thus, same state $\hat{s}(t) = \hat{s}(t+1)$. Suppose temporal synchronization is achieved for frame $\hat{Y}(t)$. Then, temporal synchronization will be achieved for frame $\hat{Y}(t+1)$. Furthermore, if $\hat{F}(t+1) = \hat{F}(t)$, then the detector queue after examining frame $\hat{Y}(t+1)$ is identical to that after examining $\hat{Y}(t)$.

Lemma 8 (Corollary): Suppose $\hat{Y}(t), \hat{Y}(t+1), \ldots, \hat{Y}(t+\beta-1)$ are consecutive frames watermarked using the same state $\hat{s}(t) = \hat{s}(t+1) = \cdots = \hat{s}(t+\beta-1)$ for $\beta > 1$, and that temporal synchronization is achieved for frame $\hat{Y}(t)$. Then temporal synchronization will be achieved for frames $\hat{Y}(t+1), \ldots, \hat{Y}(t+\beta-1)$. Furthermore, if $\hat{F}(t) = \hat{F}(t+1) = \cdots = \hat{F}(t+\beta-1)$, then the queue after the detector examines frame $\hat{Y}(t+\beta-1)$ is identical to the queue after the detector examines frame $\hat{Y}(t)$.

Theorem 4 (Temporal Redundancy and Resilience Against Frame Dropping): (See Fig. 7 for a diagram showing the assumptions of this theorem, as the hypothesis is lengthy.) Suppose $Y(u-1), Y(u), \ldots, Y(u+\beta)$ are consecutive frames of the watermarked video produced by the modified embedder such that

- frame $Y(u-1)$ is watermarked by key $K(u-1) = \lambda(s(u-1))$;
- frame $Y(u+\beta)$ is watermarked by key $K(u+\beta) = \lambda(s(u+\beta))$;
- all frames $Y(u), \ldots, Y(u+\beta-1)$, denoted set $\mathcal{Z}$, are watermarked by the key $K^* = K(u) = \cdots = K(u+\beta-1) = \lambda(s^*) = \lambda(s(u)) = \cdots = \lambda(s(u+\beta-1))$;

- feature value does not change for the frames in set $\mathcal{Z}$, or $F^* = F(u) = \cdots = F(u+\beta-1)$;
- the state transitions of the key generator occur after watermarking frames $\hat{Y}(u-1)$ and $\hat{Y}(u+\beta-1)$. Thus, $\phi(s(u-1), K_E, F(u-1)) = s^*$ and $\phi(s^*, K_E, F^*) = s(u+\beta)$.

The watermarked video is provided to the detector, where frames $\hat{Y}(t-1) = Y(u-1)$ and $\hat{Y}(t+\tau) = Y(u+\beta)$, fixed $\tau \leq \beta$, are not attacked. However, one or more frames in set $\mathcal{Z}$ may be dropped. Let the set $\hat{\mathcal{Z}} = \{\hat{Y}(t), \ldots, \hat{Y}(t+\tau-1)$ be the set of frames from $\mathcal{Z}$ that remain after the frame drop attack. Assume there are no other attacks on $\hat{\mathcal{Z}}$, and temporal synchronization is achieved for frame $\hat{Y}(t-1)$. Then, if there is at least one frame in $\hat{\mathcal{Z}}$, then temporal synchronization will be achieved for all the frames in $\hat{\mathcal{Z}}$, as well as frame $\hat{Y}(t+\tau)$.

Proof: Temporal synchronization is successful for frame $\hat{Y}(t-1)$. Thus, the queue shall contain the state $\phi(\hat{s}(t-1), K_E, \hat{F}(t-1)) = \phi(s(u-1), K_E, F(u-1)) = s^*$ (Lemma 5). Since $\hat{\mathcal{Z}}$ is nonempty and no other attacks have been performed in $\hat{\mathcal{Z}}$, there exists a frame, $\hat{Y}(t) \in \hat{\mathcal{Z}}$. For any frame in $\hat{\mathcal{Z}}$, the state of the key generator used to watermark the frame is s^* and the feature vector extracted from the frame is F^*. Since s^* is in the queue, temporal synchronization will be achieved for frame $\hat{Y}(t)$ (Lemma 3). After examining $\hat{Y}(t)$, the head of the queue will be the state $\phi(\hat{s}(t), K_E, \hat{F}(t)) = \phi(s^*, K_E, F^*) = s(u+\beta) = \hat{s}(t+\tau)$ (Lemma 5). If there is more than one frame in $\hat{\mathcal{Z}}$, temporal synchronization will be achieved for every frame in $\hat{\mathcal{Z}}$ and the queue will remain unchanged (Lemma 8). Thus, when the detector examines frame $\hat{Y}(t+\tau)$, the state $\hat{s}(t+\tau)$ will be in the queue and by Lemma 5, temporal synchronization will be achieved for $\hat{Y}(t+\tau)$. ■

Theorem 4 shows that the watermark produced by modified embedder, in which sets of β consecutive frames are watermarked by the same state of the key generator, will be resilient against frame-dropping attacks unless all β frames in a set are dropped (assuming feature vectors are constant over each set, or if feature extraction is not used). By performing state transitions every β frames instead of at every frame as described in Section IV-A, redundant copies of each state are created prior to a state transition. These redundant copies will not confuse the detector (Lemma 8), and the detector only needs to observe each state once to maintain temporal synchronization (Theorem 4). Any frame-dropping attack which drops less than β consecutive frames will fail to desynchronize the detector.

Both the strategies described above require the watermark embedder to be modified from Section IV-A, by the insertion of a *temporal redundancy control* [45], [46]. The new embedder is shown in Fig. 8. The temporal redundancy control interfaces with the WEP, as follows.

1) *Initialization*: Set counters $a = 0$ and $b = 0$. The key generator is reset, by setting the current state to a member of $\mathcal{S}_0$.
2) During step 1 of the WEP, the temporal redundancy control provides $\lambda(s(t))$ to the watermark signal generator, where $s(t)$ is the current state of the key generator.

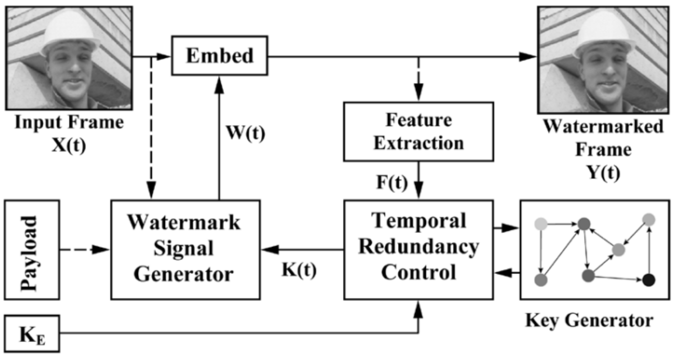


Fig. 8. Modified embedder with temporal redundancy control.

3) During step 5 of the WEP, the temporal redundancy control increments a and b. Then:
   a) if $a = \alpha$, then the key generator and the temporal redundancy control are reset for the next frame of the video. Return to step 1 for the next frame of the video.
   b) otherwise, if $b = \beta$, then the key generator performs a state transition. Set $b = 0$, and then return to step 2 for the next frame of the video.
   c) otherwise, return to step 2 for the next frame of the video, without changing the current state of the key generator.

Selecting $\alpha = \infty$ and $\beta = 1$ produces a watermark key schedule identical to that of Section IV-A.

A. Adaptive State Transitions

The temporal redundancy control described above uses two parameters to produce key sequences of varying degrees of temporal redundancy. The period (α) is the number of frames watermarked before the key generator is reset, which corresponds to the interval between resynchronization frames. The repeat (β) is the number of consecutive frames watermarked with the same key before the key generator is used to produce a new key. Generally, decreasing α and increasing β increases the amount of temporal redundancy in the key sequence and provides increased robustness against attack. However, using a fixed β for the temporal redundancy control can sometimes cause a dramatic loss in temporal redundancy when feature vectors change.

A key assumption in Theorem 4 is that feature vectors remain constant over each set of β frames watermarked with the same state. However, the actual state transition of the embedder's key generator uses only the feature vector of the last frame in a set. If feature vectors change within a set of β frames, then temporal redundancy is reduced. An example is illustrated in Fig. 9. Frames $Y(t), \ldots, Y(t+\beta-1)$ are watermarked with the same state s^*, and $Y(t), \ldots, Y(t+\beta-2)$ have the same feature vector F^*. Assume that $\phi(s^*, K_E, F^*) = \tilde{s}$, which is some state in the SM. However, suppose that the feature vector changes value for frame $F(t+\beta-1)$, and that $\phi(s^*, K_E, F(t+\beta-1)) = s(t+\beta) \neq \tilde{s}$. Then, the frames $Y(t), \ldots, Y(t+\beta-2)$ will not provide the detector queue with the state $s(t+\beta)$, even if temporal synchronization is successful for those frames. Because of the change in the feature value, the loss of frame $Y(t+\beta-1)$ would be catastrophic for temporal synchronization at the detector.

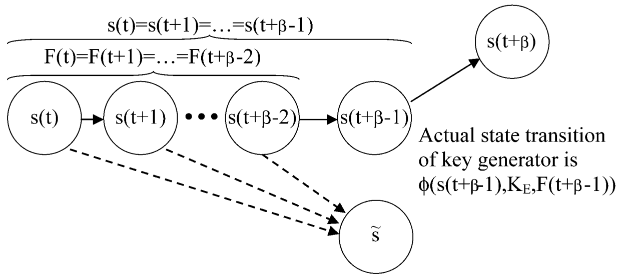


Fig. 9. Example of changing feature values reducing temporal redundancy.

An alternate strategy is to perform state transitions in the key generator based on when the feature values change, instead of a fixed interval of β frames. That is, the key generator changes its current state only after 1) feature values have been constant for β consecutive frames and 2) at least β frames have been watermarked with the same state. This strategy adapts the key generator state transitions to the characteristics of the video to increase the temporal redundancy of the key sequence and improve the robustness of the watermark.

B. Security

Temporal redundancy is an advantage for temporal synchronization of the watermark detector, however it is a disadvantage for security. A watermark whose key sequence has more redundancy is less random and is more vulnerable to estimation attacks. It is beneficial to make the state transitions appear as random as possible, which increases the difficulty for an attacker to predict the key schedule (or equivalently, the state sequence).

One method for increasing the degree of randomness in the key schedule is to use a nondeterministic SM for the key generator. The key schedule produced by a deterministic key generator is dependent only on the embedding key and the feature vectors obtained from the video. The state transitions and initial states in a nondeterministic SM introduce randomness into the key sequence. Even a small degree of randomness for each state transition can have a large effect on the state sequence because each state transition affects not just the next state of the SM, but potentially all future states of the SM.

Another method for increasing the randomness in the key schedule is to define the state transition function by using cryptographic hash functions. A cryptographic hash function, or a one-way hash function $H(M)$ is a function that accepts as input an arbitrary length binary string message M (in this context, M is not related to the watermark payload) and produces a fixed length bit string output of size L bits, known as a digest or hash value [35], [47]. The notation $H(x_1, x_2, \ldots)$ is used to indicate the L-bit digest produced by H when the concatenation of the binary representation of the values $x_1, x_2, \ldots$ is provided as the message input. Given a message M, it is very easy to obtain digest $H(M)$. However, given the digest, one cannot easily construct a message that hashes to that digest. Cryptographic hash functions are typically used for constructing message authentication codes.

An example state transition function is

$$s(t+1) = \phi(s(t), K_E, F(t)) = s_{H(s(t),K_E,F(t))} \quad (4)$$

where the set of states is $\mathcal{S} = \{s_0, s_1, s_2, \ldots, s_{2^L-1}\}$ with cardinality $|\mathcal{S}| = 2^L$. The properties of the hash function make it difficult to predict the state sequence without knowledge of K_E, and obtaining K_E from observation of the state sequence is difficult. An example state transition function for a nondeterministic SM is

$$s(t+1) = \phi(s(t), K_E, F(t)) = s_{H(s(t),K_E,F(t),\Gamma(t))} \quad (5)$$

where $\Gamma(t)$ is a randomly selected member of the set $\{0, 1, 2, \ldots, 2^R - 1\}$, R a fixed positive integer. Because the detector must insert into the queue and search all 2^R possible values of $\phi(\cdot)$, R is usually chosen to be a small number.

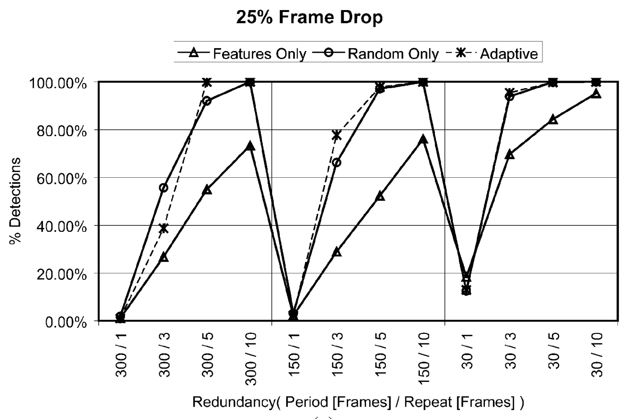
The major security concern lies in the resynchronization frames. These frames may occur frequently over the video, which presents an opportunity for estimation attacks similar in spirit to those attacking a time-periodic key watermark. The frequency of resynchronization frames (parameter α) is a tradeoff between the ease of synchronization and security. To improve security, resynchronization frames must be more difficult to identify and estimate. One method to do this is to use image-dependent watermarking [32], [48]. Another method is to use the feature vector of the previous frame to determine the key for the resynchronization frame. That is, if the key generator is reset for watermarking frame $X(t)$, $s(t)$ is not set to a member of $\mathcal{S}_0$, but to a state which is dependent on $F(t-1)$, K_E, and possibly $\Gamma(t)$. The WDP (step 2) would be modified accordingly. These methods generate different watermark patterns for the resynchronization frames and make them more difficult to identify and estimate. Neither of these methods are used in our experiments.

VI. Experimental Results

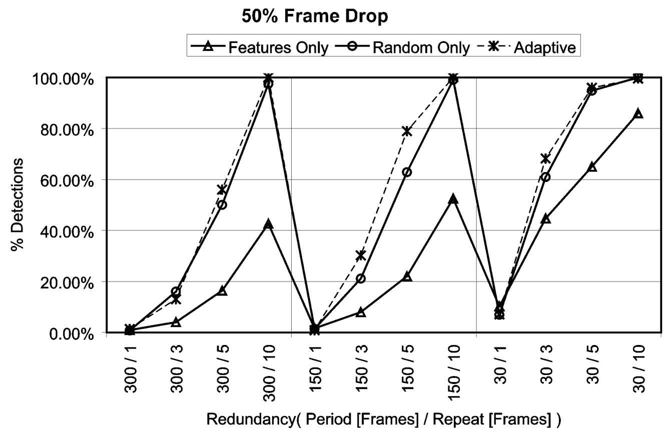
To evaluate the effectiveness of adding temporal redundancy on temporal synchronization, the watermark embedding and detection models have been implemented for uncompressed video sequences. The watermark signal generator is a pseudorandom number generator which is seeded by $K(t)$ for each frame, producing a zero-mean Gaussian watermark signal. The watermark embedder adds the watermark to the original video in the spatial domain (luminance only). The watermark detector applies a spatial de-correlating filter to reduce host-signal interference, followed by a correlation detector and comparison with a threshold value. The detector queue capacity for all experiments is fixed to ten entries.

Watermark embedders using three different key generators are examined. All three key generators include the temporal redundancy control shown in Fig. 8 and the state transition functions are cryptographic hash functions as described in Section V-B. SHA-1 [49] is used as the hash function, which produces a message digest of $L = 160$ bits in size. The key generators also share in common $\mathcal{K} = \{0, 1, 2, \ldots, 2^{160} - 1\}$, $\mathcal{S} = \{s_0, s_1, \ldots, s_{2^{160}-1}\}$, $\mathcal{S}_0 = s_{K_E}$, and $\lambda(s_i) = i$.

The first key generator ("Features Only") uses feature extraction and a deterministic SM (4). The feature extraction produces a feature vector by partitioning the watermarked frame into nonoverlapping regions, calculating the average luminance value of each region, and then quantizing these values (by using



(a)



(b)

Fig. 10. Detection rate under frame-dropping attack.

a uniform scalar quantizer). These features are selected because they are straightforward to implement and computationally efficient to obtain, and not because they are optimal in any sense (including robustness and security). The second key generator ("Random Only") uses the nondeterministic key generator with the state transition function of (5), with $R = 2$. The "Random Only" embedder does not perform feature extraction. The third key generator ("Adaptive") uses a nondeterministic key generator with the state transition function of (5), with $R = 2$, the same features as the "Features Only" key generator, and the adaptive state transitions described in Section V-A.

The results show the mean performance of the detector after ten trials of each of the *Akiyo*, *Foreman*, and *Bus* sequences (352×288 CIF, 3 frames/s). A randomly-generated K_E was used for each trial. Performance is measured by the percentage of the watermarked frames that are detected in the attacked video as the temporal redundancy is varied. A detection rate of 100% indicates that the watermark detector detects the watermarks embedded in every frame of the attacked video and the attack is ineffective. A detection rate of 0% indicates that the detector is unable to detect the watermark in the attacked video. Temporal redundancy is expressed in terms of α (period) and β (repeat) parameters.

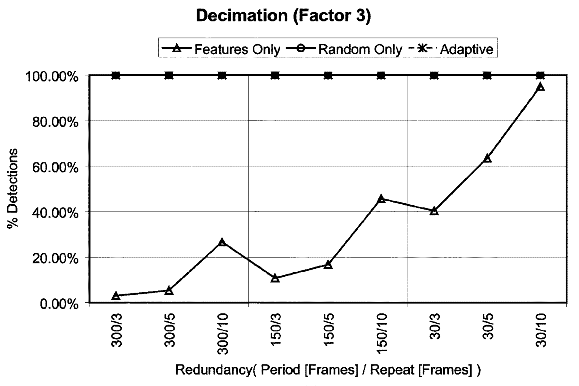
Fig. 10 shows the watermark detection performance after frame-dropping attack, where each frame of the watermarked video is dropped with probability $P = 0.25$ and 0.5. For fixed α, the detections improve for all the embedders as β is increased, showing that robustness improves with increasing temporal redundancy (Theorem 4). When there is little temporal redundancy ($\beta = 1$), the performance of all the embedders is poor, in agreement with Theorem 3. However, the performance of the embedders improve significantly with added temporal redundancy ($\beta = 5$, 10), particularly for the "Random Only" and "Adaptive" embedders. For fixed β, the performance improves with decreasing α. When the detector loses temporal synchronization, it will not recover synchronization until it discovers a frame watermarked with a state $\hat{s}(t) \in \mathcal{S}_0$ or in the queue. When α is decreased, frames are watermarked from a state in $\mathcal{S}_0$ more frequently, allowing the detector to recover synchronization.

To show that the loss of temporal synchronization occurs when β consecutive frames are dropped, the detection rate was examined after temporal decimation attack. The results are shown in Fig. 11. Decimation by a factor of N retains the first frame out of every N consecutive frames, and filtering was not performed prior to decimation. Choosing to retain the first frame of every N frames provides the most advantageous situation with respect to the watermark detector (and not the attacker). This shows Theorem 4 most vividly, as it is our intention to show that the detection rates drop significantly when the decimation factor exceeds β, even in the best case. If the decimation attack retained the last frame of every N frames, initial synchronization will fail when N exceeds β.

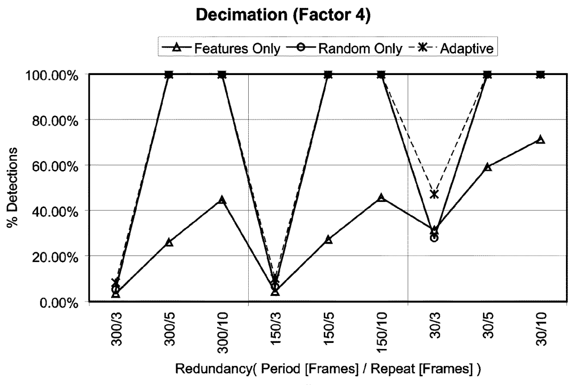
The effect of changing feature values on temporal redundancy (described in Section V-A) is dramatic, resulting in the poor performance of the "Features Only" embedder. For the "Random Only" and "Adaptive" embedders, the watermark detection rate is 100% when the decimation factor is equal to or below β, demonstrating that the loss of $\beta - 1$ frames will not destroy temporal synchronization. The detection rate decreases significantly when the decimation factor exceeds β. There is one exception, in which the detection rate of these embedders when $\alpha = 30$ and $\beta = 5$ remains good even after the watermarked video is decimated by factor of six. This arises because the key generator is reset so often that the watermark detector discovers a frame watermarked by a state in $\mathcal{S}_0$ before the decimation attack can affect detection.

The insertion of unwatermarked frames into the video does not affect watermark detection, in agreement with Theorem 2. The observed detection rate is 100% for all embedders and temporal redundancy parameters. In this attack, an arbitrary number of frames from the original video are inserted between consecutive frames of the watermarked video. Failure to detect the watermark in the inserted frames does not penalize the performance of the detector because the inserted frames are not watermarked.

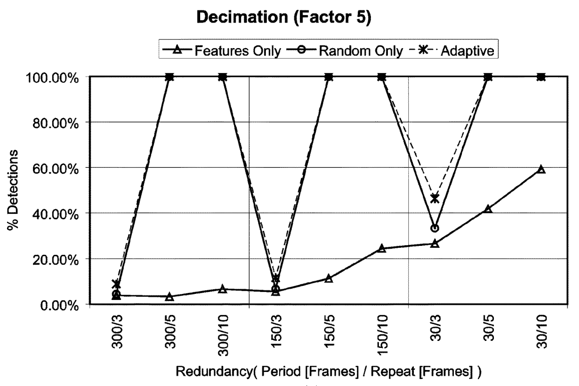
Frame transposition has a similar effect to frame dropping on disrupting the "chain of states" induced by $\phi(\cdot)$, with the difference being that the attack displaces the video frames in time instead of removing frames from the video. The implementation of this attack is as follows: The watermarked video is scanned from beginning to end, with each frame having a fixed probability ($P = 0.25$ and 0.5) of being interchanged with another (target) frame in the local neighborhood of the candidate frame.



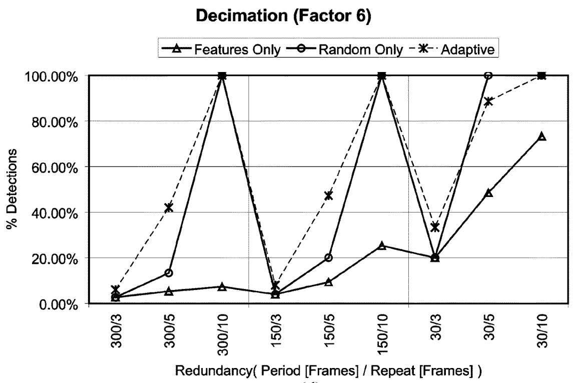
(a)



(b)

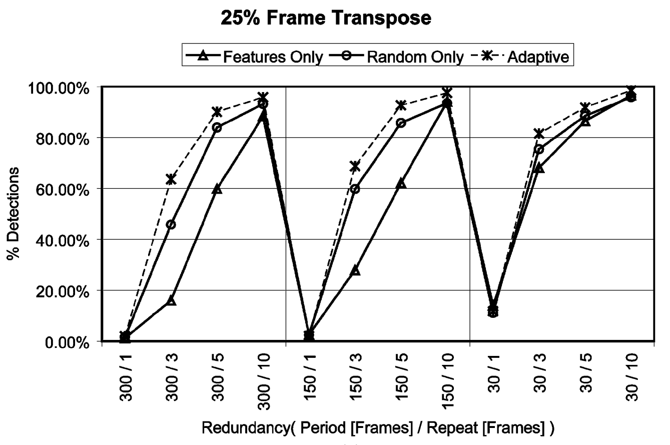


(c)

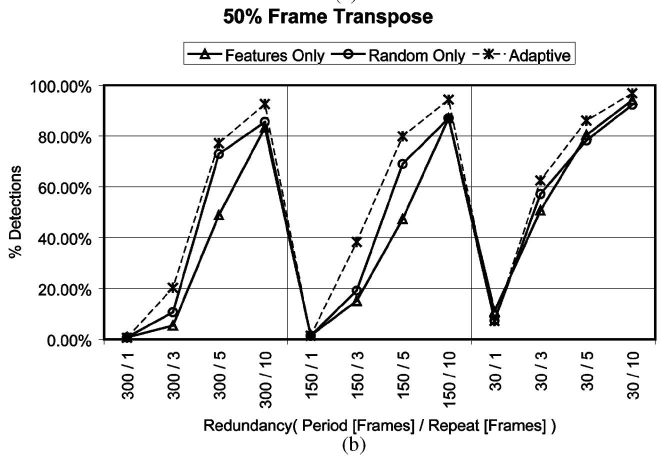


(d)

Fig. 11. Detection rate under frame decimation attack.



(a)

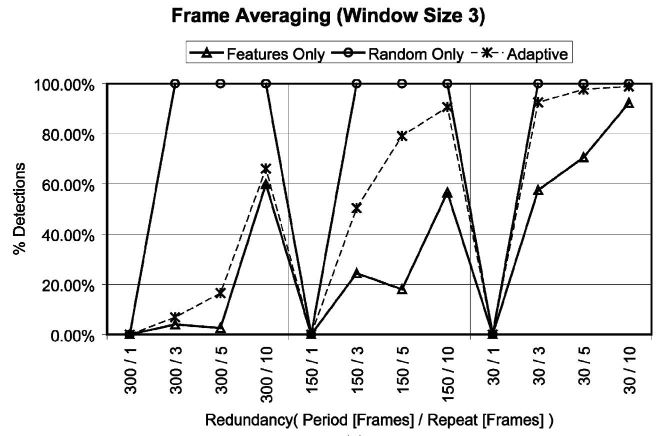


(b)

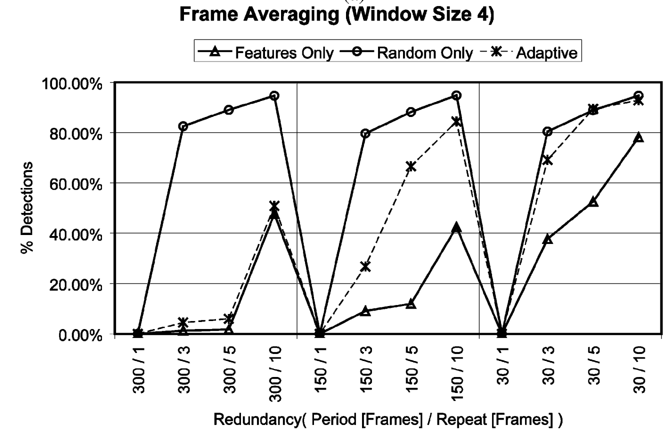
Fig. 12. Detection rate under frame transposition attack.

The target frame is selected by generating a Gaussian random number ($\sigma^2 = 5.0$), dropping fractions, and treating the number as a relative time index where 0 is the current frame, -1 is one frame in the past, $+1$ is one frame in the future, and so on. Because transposing a frame with itself does not accomplish anything, another random number is chosen if the relative time index is zero or out of range of the video. The performance of the watermark detection under frame transposition attack, shown in Fig. 12, shows similar trends to frame dropping.

Temporal frame averaging was also investigated, using several moving averaging window sizes. Fig. 13 shows the detection rate of the watermark after frame averaging using window sizes of 3 and 4. The detection rates using a window size of 2 are superior to those shown in Fig. 13 and are not shown. Frame averaging does not change the sequence of frames appearing in the video, but the averaging may affect the feature values and cause the detector to obtain different feature values than those used by the embedder to generate the key schedule. When this occurs, the detector's state prediction will fail and the detector will lose synchronization. The "Random Only" embedder, which does not use feature extraction, is not affected by frame averaging attack until degradation of the embedded watermark signal itself causes the watermark detector to miss. The performance of the "Features Only" and "Adaptive" embedders decreases under frame averaging attack. The "Adaptive" embedder often shows much better detection rate than "Features Only." For fixed α and β, state transitions do not occur as frequently for the "Adaptive" embedder compared with the "Features Only" embedder. Because less state transitions occur for the "Adaptive" embedder, there is less opportunity for the detector to lose synchronization.



(a)



(b)

Fig. 13. Detection rate under frame averaging attack.

Comparing the three embedders, the "Features Only" embedder often shows worse performance than the "Random Only" and the "Adaptive" embedders, whose performance is often similar. This vividly shows the effect of changing feature values on the temporal redundancy of the watermark described in Section V-A, as the "Features Only" and "Adaptive" embedders use the same feature extractor but the performance of "Adaptive" is better than "Features Only." Adaptive state transitions significantly improve the performance of the watermark when feature extraction is used.

The inserted watermark is generally not noticeable (the embedding in these experiments is such that the peak signal-to-noise ratio between the watermarked and original videos is ≥ 40 dB); however, the attacks performed in these experiments reduce the quality of the attacked video. The frame-dropping attack is generally unnoticeable when approximately 5%–10% of the frames are dropped; however, higher drop rates make the video appear "jerky." Decimation reduces the frame rate, which is noticeable even when the decimation factor is 2. The frame transposition attack produces "jerky" video like the frame-dropping attack, and frame averaging noticeably blurs the video.

VII. Conclusions

Temporal synchronization was examined using new models for video watermark embedding and detection. These models can be extended in further investigation. The graph structure of the key generator affects temporal redundancy. For example, a key generator using a SM with a strongly connected graph representation may be more robust or secure. Instead of watermarking multiple frames with the same key, temporal redundancy may be inserted into the key schedule by altering or exploiting the structure of the graph. Another extension is to extend the state prediction from predicting the next state $(\tilde{s}(t+1))$ to predicting multiple states in time $(\tilde{s}(t+1), \tilde{s}(t+2), \ldots)$. Feature extraction, while offering a means for creating a video dependent key schedule, is relatively unexplored. Also, the embedder model uses feature extraction for each frame to affect the watermark structure of future frames. This can be compared, in terms of robustness, security, and invisibility, to frame-dependent techniques (where features of the video frame are used to affect the watermark structure of that frame).

The adaptive state transition model described in this paper is simple, but has the shortcoming that state transitions occur when the video is relatively static. This can introduce flicker when the watermarked video is viewed. Flicker was not observed in the experiments because the power of the embedded watermark was sufficiently small, but flicker may be an issue when the embedding power is increased. A more sophisticated adaptive model would would also consider the video before deciding a state transition. For example, the state transition may be deferred until the *next* frame when feature values change, after the feature values have been constant for at least β consecutive frames.

The examination of synchronization in watermark detection has often focused on designing templates, either by embedding an explicit synchronization signal, organizing the watermark to produce such a signal, or by using salient features of the video for synchronization. However, there has been relatively little work in modeling the watermark embedding and detection processes for synchronization. Such models can be used to show why some watermarks (such as time-invariant key watermarks for temporal synchronization and tiled watermarks for spatial synchronization) are "easier" to synchronize than others, even when the watermark embedding does not take advantage of any special transform or invariance properties.

Appendix

The proofs for the lemmas are shown here.

Lemma 1:

Proof: This follows from the WEP. In particular, exactly one state transition occurs for every pair of successive frames, and for any state transition $s(t+1) = \phi(s(t), K_E, F(t))$. ■

Lemma 2:

Proof: Because neither $\hat{Y}(t)$ nor $\hat{Y}(t+1)$ is attacked, there must be a pair of frames $Y(u)$ and $Y(u+1)$, such that $\hat{Y}(t) = Y(u)$ and $\hat{Y}(t+1) = Y(u+1)$. (The time indices t and u are not necessarily identical because it is possible that frames were dropped or inserted into $\hat{\mathbf{Y}}$ outside the time interval t and $t+1$.) From $\hat{Y}(t) = Y(u)$, it follows that $\hat{K}(t) = K(u)$ and $\hat{s}(t) = s(u)$. The feature extraction will also produce identical feature vectors $\hat{F}(t) = F(u)$. From $\hat{Y}(t+1) = Y(u+1)$, it follows that $\hat{K}(t+1) = K(u+1)$ and $\hat{s}(t+1) = s(u+1)$. Lemma 1 holds for $Y(u)$ and $Y(u+1)$, so

$s(u+1) = \phi(s(u), K_E, F(u))$. Then, $\hat{s}(t+1) = s(u+1) = \phi(s(u), K_E, F(u)) = \phi(\hat{s}(t), K_E, \hat{F}(t))$. ■

Lemma 3:

Proof: Suppose temporal synchronization is achieved. This implies that the detector found some key $\hat{K}(t)$ and state $\hat{s}(t)$, such that $\hat{K}(t) = \lambda(\hat{s}(t))$ produced the watermark signal embedded in frame $\hat{Y}(t)$. To establish synchronization, the WDP searches the keys corresponding to the states in $\mathcal{S}_0$ (during step 2) and the queue (during step 3), and no other states. Thus, it must be the case that $\hat{s}(t)$ is either in $\mathcal{S}_0$ or the queue. Suppose that $\hat{Y}(t)$ is watermarked with key $\hat{K}(t) = \lambda(\hat{s}(t))$, and $\hat{s}(t)$ is either a member of $\mathcal{S}_0$ or is in the queue at time t. Because the WDP attempts to detect the watermark using all of the states in $\mathcal{S}_0$ and in the queue, the watermark detector will try $\hat{s}(t)$ and temporal synchronization is achieved. ■

Lemma 4:

Proof: This follows immediately from Lemma 3. The significant statement of this Lemma is that synchronization will be achieved regardless of the contents of the queue. ■

Lemma 5:

Proof: By Lemma 3, if temporal synchronization is achieved then $\hat{s}(t)$ must either be in $\mathcal{S}_0$ or the queue. The remainder of the proof follows by inspecting the queue after step 5 of the WDP. ■

Lemma 6:

Proof: If temporal synchronization fails, then either $\hat{Y}(t)$ is not watermarked or the state that produces the key which generated the watermark signal embedded in $\hat{Y}(t)$ is not in $\mathcal{S}_0$ or the queue. In this case, none of the steps in the WDP affect the queue. ■

Lemma 7:

Proof: Suppose $\hat{s}(t) \in \mathcal{S}_0$. Then $\hat{s}(t+1) = \hat{s}(t) \in \mathcal{S}_0$, so by Lemma 4, temporal synchronization will succeed for frame $\hat{Y}(t+1)$. By Lemma 5, the head of the queue will be $q(0) = \phi(\hat{s}(t), K_E, \hat{F}(t))$ after examining frame $\hat{Y}(t)$. If $\hat{F}(t+1) = \hat{F}(t)$, then $\phi(\hat{s}(t+1), K_E, \hat{F}(t+1)) = \phi(\hat{s}(t), K_E, \hat{F}(t))$, which is already at the head of the queue and no new state will be added to the queue during step 5 of the WDP for frame $\hat{Y}(t+1)$. Thus, the queue after examining frame $\hat{Y}(t+1)$ is identical to that after examining $\hat{Y}(t)$.

Now suppose $\hat{s}(t) \notin \mathcal{S}_0$. By Lemma 5, successful temporal synchronization for frame $\hat{Y}(t)$ implies that after examining $\hat{Y}(t)$, the queue has $q(0) = \phi(\hat{s}(t), K_E, \hat{F}(t))$ and $q(1) = \hat{s}(t)$. Consider the WDP when frame $\hat{Y}(t+1)$ is examined by the detector. State $\hat{s}(t+1) = \hat{s}(t)$ is in the queue, so temporal synchronization will succeed in step 3 of the WDP. During this step, the state $\hat{s}(t)$ will be promoted and thus $q(0) = \hat{s}(t)$ and $q(1) = \phi(\hat{s}(t), K_E, \hat{F}(t))$. But if $\hat{F}(t+1) = \hat{F}(t)$ then $\phi(\hat{s}(t+1), K_E, \hat{F}(t+1)) = \phi(\hat{s}(t), K_E, \hat{F}(t))$, which is already in the queue, so during step 5 of the WDP, that state shall be promoted to the head and $q(0) = \phi(\hat{s}(t), K_E, \hat{F}(t))$ and $q(1) = \hat{s}(t)$. No other states in the queue are affected by the WDP. Thus, if $\hat{F}(t+1) = \hat{F}(t)$, then the queue is not changed when $\hat{Y}(t+1)$ is examined. ■

Lemma 8:

Proof: Apply Lemma 7 to every pair of frames $\hat{Y}(t)$ and $\hat{Y}(t+1)$, $\hat{Y}(t+1)$ and $\hat{Y}(t+2), \ldots$, and $\hat{Y}(t+\beta-2)$ and $\hat{Y}(t+\beta-1)$. ■

References

[1] R. Wolfgang, C. Podilchuk, and E. Delp, "Perceptual watermarks for digital images and video," *Proc. IEEE*, vol. 87, pp. 1108–1126, July 1999.

[2] A. M. Eskicioglu and E. J. Delp, "An overview of multimedia content protection in consumer electronics devices," *Signal Processing: Image Commun.*, vol. 16, no. 7, pp. 681–699, Apr. 2001.

[3] F. Hartung and F. Ramme, "Digital rights management and watermarking of multimedia content for m-commerce applications," *IEEE Commun. Mag.*, vol. 38, pp. 78–84, Nov. 2000.

[4] J. A. Bloom, I. J. Cox, T. Kalker, J.-P. M. G. Linnartz, M. L. Miller, and C. B. S. Traw, "Copy protection for DVD video," *Proc. IEEE*, vol. 87, pp. 1267–1276, July 1999.

[5] G. Doërr and J.-L. Dugelay, "A guide tour of video watermarking," *Signal Processing: Image Commun.*, vol. 18, no. 4, pp. 263–282, Apr. 2003.

[6] I. Cox, M. Miller, and J. Bloom, *Digital Watermarking*. San Francisco, CA: Morgan Kaufmann, 2002.

[7] G. Langelaar, I. Setyawan, and R. Lagendijk, "Watermarking digital image and video data: A state-of-the-art overview," *IEEE Signal Processing Mag.*, vol. 17, pp. 20–46, Sept. 2000.

[8] F. Hartung and M. Kutter, "Multimedia watermarking techniques," *Proc. IEEE*, vol. 87, pp. 1079–1107, July 1999.

[9] M. Swanson, M. Kobayashi, and A. Tewfik, "Multimedia data-embedding and watermarking technologies," *Proc. IEEE*, vol. 86, pp. 1064–1087, June 1998.

[10] S. Voloshynovskiy, S. Pereira, T. Pun, J. Eggers, and J. Su, "Attacks on digital watermarks: Classification, estimation-based attacks, and benchmarks," *IEEE Commun. Mag.*, vol. 39, pp. 118–126, Aug. 2001.

[11] J.-L. Dugelay and F. A. P. Petitcolas, "Possible counter-attacks against random geometric distortions," in *Proc. SPIE Security and Watermarking of Multimedia Contents II*, vol. 3971, San Jose, CA, Jan. 24–26, 2000, pp. 338–345.

[12] E. Lin, C. Podilchuk, T. Kalker, and E. Delp, "Streaming video and rate scalable compression: What are the challenges for watermarking?," *J. Electron. Imaging*, vol. 13, no. 1, pp. 198–205, Jan. 2004.

[13] D. Wu, Y. T. Hou, and Y.-Q. Zhang, "Transporting real-time video over the internet: Challenges and approaches," *Proc. IEEE*, vol. 88, pp. 1855–1877, Dec. 2000.

[14] F. Hartung and B. Girod, "Watermarking of uncompressed and compressed video," *Signal Processing*, vol. 66, no. 3, pp. 283–301, May 1998.

[15] F. Hartung, "Digital watermarking and fingerprinting of uncompressed and compressed video," Ph.D. dissertation, Univ. Erlangen-Nürnberg, Erlangen-Nürnberg, Germany, 2000.

[16] J. Lichtenauer, I. Setyawan, T. Kalker, and R. Lagendijk, "Exhaustive geometrical search and the false positive watermark detection probability," in *Proc. SPIE Security and Watermarking of Multimedia Contents V*, vol. 5020, Santa Clara, CA, Jan. 2003.

[17] R. Lancini, F. Mapelli, and S. Tubaro, "A robust video watermarking technique in the spatial domain," in *IEEE Region-8 Int. Symp. Video/Image Processing and Multimedia Communications*, Zadar, Croatia, June 16–19, 2002, pp. 251–256.

[18] N. V. Boulgouris, F. D. Koravos, and M. G. Strintzis, "Self-synchronizing watermark detection for MPEG-4 objects," in *Proc. 8th IEEE Int. Conf. Electronics, Circuits, and Systems 2001*, vol. 3, Oct. 2001, pp. 1371–1374.

[19] F. Deguillaume, G. Csurka, J. O'Ruanaidh, and T. Pun, "Robust 3D DFT video watermarking," in *Proc. SPIE Security and Watermarking of Multimedia Contents I*, vol. 3657, San Jose, CA, Jan. 25–27, 1999, pp. 113–124.

[20] X. Niu, M. Schmucker, and C. Busch, "Video watermarking resisting to rotation, scale, and translation," in *Proc. SPIE Security and Watermarking of Multimedia Contents IV*, vol. 4675, San Jose, CA, Jan. 21–24, 2002, pp. 512–519.

[21] A. Herrigel, S. Voloshynovskiy, and Y. Rytsar, "The watermark template attack," in *Proc. SPIE Security and Watermarking of Multimedia Contents III*, vol. 4314, San Jose, CA, Jan. 22–25, 2001, pp. 394–405.

[22] T. Kalker, G. Depovere, J. Haitsma, and M. Maes, "A video watermarking system for broadcast monitoring," in *Proc. SPIE Security and Watermarking of Multimedia Contents I*, vol. 3657, San Jose, CA, Jan. 25–27, 1999, pp. 103–112.

[23] D. Delannay and B. Macq, "Generalized 2-D cyclic patterns for secret watermark generation," in *Proc. IEEE Int. Conf. Image Processing 2000*, vol. 2, Vancouver, BC, Canada, Oct. 10–13, 2000, pp. 77–79.

[24] P. Bas, J.-M. Chassery, and B. Macq, "Geometrically invariant watermarking using feature points," *IEEE Trans. Image Processing*, vol. 11, pp. 1014–1028, Sept. 2002.

[25] K. Su, D. Kundur, and D. Hatzinakos, "A novel approach to collusion-resistant video watermarking," in *Proc. SPIE Security and Watermarking of Multimedia Contents IV*, vol. 4675, San Jose, CA, Jan. 21–24, 2002, pp. 491–502.

[26] C.-P. Wu, P.-C. Su, and C.-C. J. Kuo, "Robust and efficient digital audio watermarking using audio content analysis," in *Proc. SPIE Security and Watermarking of Multimedia Contents II*, vol. 3971, San Jose, CA, Jan. 24–26, 2000, pp. 382–392.

[27] J. Dittmann, T. Fiebig, and R. Steinmetz, "A new approach for transformation invariant image and video watermarking in the spatial domain: Ssp—Self spanning patterns," in *Proc. SPIE Security and Watermarking of Multimedia Contents II*, vol. 3971, San Jose, CA, Jan. 24–26, 2000, pp. 176–185.

[28] C.-Y. Lin, M. Wu, J. A. Bloom, I. J. Cox, M. L. Miller, and Y. M. Lui, "Rotation, scale, and translation resilient watermarking for images," *IEEE Trans. Image Processing*, vol. 10, pp. 767–782, May 2001.

[29] J. J. O'Ruanaidh and T. Pun, "Rotation, scale and translation invariant spread spectrum digital image watermarking," *Signal Processing*, vol. 66, no. 3, pp. 303–317, May 1998.

[30] I. Setyawan, G. Kakes, and R. L. Lagendijk, "Synchronization-insensitive video watermarking using structured noise pattern," in *Proc. SPIE Security and Watermarking of Multimedia Contents IV*, vol. 4675, San Jose, CA, Jan. 21–24, 2002, pp. 520–530.

[31] S. Pereira and T. Pun, "Robust template matching for affine resistant image watermarks," *IEEE Trans. Image Processing*, vol. 9, pp. 1123–1129, June 2000.

[32] M. Holliman, W. Macy, and M. M. Yeung, "Robust frame-dependent video watermarking," in *Proc. SPIE Security and Watermarking of Multimedia Contents II*, vol. 3971, San Jose, CA, Jan. 2000, pp. 186–197.

[33] J. Eggers and B. Girod, *Informed Watermarking*. Boston, MA: Kluwer, 2002.

[34] F. Hartung and B. Girod, "Fast public-key watermarking of compressed video," in *Proc. IEEE Int. Conf. Image Processing 1997*, vol. 1, Santa Barbara, CA, Oct. 1997, pp. 528–531.

[35] B. Schneier, *Applied Cryptography*, 2nd ed. New York: Wiley, 1996.

[36] G. Langelaar and R. Lagendijk, "Optimal differential energy watermarking of DCT encoded images and video," *IEEE Trans. Image Processing*, vol. 10, pp. 148–158, Jan. 2001.

[37] J. F. Wakerly, *Digital Design Principles and Practices*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

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

[39] C. Fischer and R. J. LeBlanc Jr, "Crafting a Compiler With C," Benjamin/Cummings, Redwood City, CA, 1991.

[40] M. D. Davis, R. Sigal, and E. J. Weyuker, *Computability, Complexity, and Languages*, 2nd ed. Boston, MA: Academic, 1994.

[41] M. R. Garey and D. S. Johnson, *Computers and Intractability a Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[42] S. Cass, "Mind games," *IEEE Spectrum*, pp. 40–44, Dec. 2002.

[43] S. Craver, N. Memon, B.-L. Yeo, and M. M. Yeung, "Resolving rightful ownerships with invisible watermarking techniques: Limitations, attacks, and implications," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 573–586, May 1998.

[44] M. Kutter, S. Voloshynovskiy, and A. Herrigel, "The watermark copy attack," in *Proc. SPIE Security and Watermarking of Multimedia Contents II*, vol. 3971, San Jose, CA, Jan. 24–26, 2000, pp. 371–380.

[45] E. T. Lin and E. J. Delp, "Temporal synchronization in video watermarking-further studies," in *Proc. SPIE Security and Watermarking of Multimedia Contents V*, vol. 5020, Santa Clara, CA, Jan. 21–24, 2003, pp. 493–504.

[46] ——, "Temporal synchronization in video watermarking," in *Proc. SPIE Security and Watermarking of Multimedia Contents IV*, vol. 4675, San Jose, CA, Jan. 21–24, 2002, pp. 478–490.

[47] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk, "Cryptographic hash functions: A survey," Department of Computer Science, Univ. Wollongong, Tech. Rep. 95-09, 1995.

[48] D. Delannay and B. Macq, "A method for hiding synchronization marks in scale and rotation resilient watermarking schemes," in *Proc. SPIE Security and Watermarking of Multimedia Contents IV*, vol. 4675, San Jose, CA, Jan. 21–24, 2002, pp. 548–554.

[49] *Secure Hash Standard*, Standard 180-1, Apr. 17, 1995.

Eugene T. Lin (S'99) was born in Stillwater, OK, in 1973. He received the B.S. degree in computer and electrical engineering in 1994 and the M.S. degree in electrical engineering in 1996, both from Purdue University, West Lafayette, IN, where he is currently pursuing the Ph.D. degree in video watermarking techniques.

He was an intern at Lucent Technologies in the summer of 2000 and then at Digimarc Corporation in 2001 and 2002. His research interests include video watermarking and steganography, as well as video coding and image processing.

Mr. Lin is a member of Eta Kappa Nu.

Edward J. Delp (S'70–M'79–SM'86–F'97) was born in Cincinnati, OH. He received the B.S.E.E. (*cum laude*) and M.S. degrees from the University of Cincinnati, Cincinnati, OH, and the Ph.D. degree from Purdue University, West Lafayette, IN. In May 2002, he received an Honorary Doctor of Technology degree from the Tampere University of Technology, Tampere, Finland.

From 1980 to 1984, he was with the Department of Electrical and Computer Engineering, The University of Michigan, Ann Arbor. Since August 1984, he has been with the School of Electrical and Computer Engineering and the Department of Biomedical Engineering, Purdue University. In 2002, he received a Chaired Professorship and currently is The Silicon Valley Professor of Electrical and Computer Engineering and Professor of Biomedical Engineering. His research interests include image and video compression, multimedia security, medical imaging, multimedia systems, and communication and information theory.

Dr. Delp was Co-Chair of the SPIE/Imaging Science and Technology (IS&T) Conference on Security and Watermarking of Multimedia Contents from 1999 to 2004. He was the General Co-Chair of the 1997 Visual Communications and Image Processing Conference (VCIP). He was Program Chair of the IEEE Signal Processing (SP) Society's Ninth IMDSP Workshop in 1996 and Program Co-Chair of the IEEE International Conference on Image Processing in 2003. He is a Fellow of the SPIE, the Society for Imaging Science and Technology (IS&T), and the American Institute of Medical and Biological Engineering. In 2000, he was selected as a Distinguished Lecturer of the IEEE SP Society.