Movee: Video Liveness Verification for Mobile Devices Using Built-In Motion Sensors

Mahmudur Rahman, Umut Topkara, and Bogdan Carbunar

Abstract—The ubiquitous and connected nature of camera-equipped mobile devices has greatly increased the value and importance of visual information they capture. Today, broadcasting videos from camera phones uploaded by unknown users is admissible on news networks, and banking customers expect to be able to deposit checks using mobile devices. In this paper, we introduce Movee, a system that addresses the fundamental question of whether the visual stream uploaded by a user has been captured live on a mobile device, and has not been tampered with by an adversary. Movee leverages the mobile device motion sensors and the intrinsic user movements during the shooting of the video. Movee exploits the observation that the movement of the scene recorded on the video stream should be related to the movement of the device simultaneously captured by the accelerometer. Contrary to existing algorithms, Movee has the unique strength of not depending on the audio track. We introduce novel attacks that focus on Movee's defenses, to fabricate acceleration data that mimics the motion observed in targeted videos. We use smartphones and wearable smart glasses to collect both genuine and attack data from 13 users. Our experiments show that Movee is able to efficiently detect human and automatically generated plagiarized videos: Movee's accuracy ranges between 68-93 percent on a smartphone, and between 76-91 percent on a Google Glass device.

Index Terms—Mobile video liveness, sensor based verifications

1 Introduction

In response to the ubiquitous and connected nature of mobile and wearable devices, industries such as utilities, insurance, banking, retail, and broadcast news have started to trust visual information gleaned from or created using mobile devices. Mobile apps utilize mobile and wearable device cameras for purposes varying from authentication to location verification, tracking, witnessing, and remote assistance. Today, one can deposit a check using a mobile phone, and videos from mobile phones uploaded by unknown users are shown on broadcast news to a national audience.

We address the fundamental question of whether the visual stream uploaded by a user has been captured live on a mobile device, and has not been tampered with by a malicious user attempting to game the system. We refer to this problem as video "liveness" verification. The practical attacks we consider are feeding a previously recorded video through man in the middle software ("Copy-Paste" attack) and pointing the camera to a projection of a video ("Projection" attack).

This problem is a cornerstone in a variety of practical applications that use the mobile device camera as a trusted witness. Examples applications include citizen journalism, where people record witnessed events (e.g., public protests, natural or man-made disasters) and share their records with the community at large. Other applications include video based proofs of physical possession of products and prototypes (e.g., for sites like Kickstarter [5], Amazon [1] and eBay [3]), and of deposited checks [17], [33].

In this paper we introduce Movee, a motion sensor based video liveness verification system. Movee leverages the ubiquitous mobile device accelerometers and the intrinsic movements of the user's hand and body during the shooting of the video. Movee exploits the intuition that video frames and accelerometer data captured simultaneously will bear certain relations. Specifically, the movement of the scene recorded in the video stream should be related to the movement of the device registered by the accelerometer. We conjecture that such relations are difficult to fabricate and emulate.

If the data from the accelerometer corroborates the data from the camera, Movee concludes that the video stream was genuine, and has been taken by the user pointing the camera to a real scene.

In essence, Movee provides CAPTCHA [61] like verifications, by including the user, through her mobile device, into the visual verification process. However, instead of using the cognitive strength of humans to interpret visual information, we rely on their innately flawed ability to hold a camera still. Movee can also be viewed as a visual public notary that stamps an untrusted video stream, with data simultaneously captured from a trusted sensor. This data can later be used to verify the liveness of the video.

Previous work [20], [22], [26], [27], [34], [46] has proposed to use audio streams in captured videos to protect against spoofing attacks in biometric authentication. The proposed verifications use static and dynamic relations between the recorded voice and the motion of the user's face. In this paper we use the previously unexplored combination of

- *M. Rahman is with the School of Computing and Information Sciences, Florida International University, Miami, FL 33199. E-mail: mrahm004@cs.fiu.edu.*
- *U. Topkara is with the JW Player in New York, NY 10001. E-mail: umut@jwplayer.com.*
- *B. Carbunar is with the School of Computing and Information Sciences, Florida International University, Miami, FL 33199. E-mail: carbunar@cs.fiu.edu.*

Manuscript received 17 July 2014; revised 21 May 2015; accepted 2 July 2015. Date of publication 20 July 2015; date of current version 31 Mar. 2016. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TMC.2015.2456904

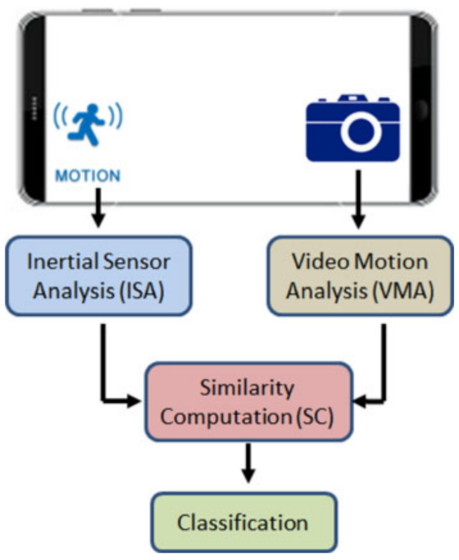

MOTION
Inertial Sensor Analysis (ISA)
Video Motion Analysis (VMA)
Similarity Computation (SC)
Classification

Fig. 1. Movee uses four modules to verify a video stream: the i) Video motion analysis, and the ii) Inertial sensor motion analysis, produce movement estimations during capture, iii) Similarity computation extracts features, which iv) classification uses to make the final decision.

video and accelerometer data to solve a different problem: verify the liveness of the video capture process.

Movee consists of four modules, illustrated in Fig. 1. The video motion analysis (VMA) module processes the video stream captured by the camera. It uses video processing techniques to infer the motion of the camera, producing a time-dependent motion vector. VMA is inspired by the process used in image stabilization capable cameras. The Inertial sensor motion analysis (IMA) module converts the data signal captured from the inertial sensor into another time-dependent motion vector. The motion vectors produced by the VMA and IMA modules are compared in the similarity computation (SC) module. SC relies on a flavor of the dynamic time warping (DTW) algorithm [19], [53] to compute the "similarity" of the two motion vectors. The SC module also produces a set of features which summarize the nature of the similarity. The features are then used by the classification module, which runs trained classifiers to decide whether the two motion sequences corroborate each other. If they do, Movee concludes the video is genuine.

We note that Movee detects plagiarized mobile videos without having to precisely quantify the camera displacement as inferred from the accelerometer or the video frames. This would have required the use of state of the art computer vision and robotics analysis. Instead, Movee leverages a classifier to verify whether the motion-related features extracted from the video stream are consistent with the motion-related features inferred from the accelerometer sensor stream.

In a second contribution, we extend the previously mentioned copy-paste and projection attacks, with manual and automated attacks that target Movee's defenses. The novel attacks we introduce, seek to produce acceleration data that mimics the motion observed in the targeted video.

We have implemented Movee on both Android and Google Glass devices and used it to collect more than 1,000, 6 second long, genuine video and corresponding inertial sensor samples from 13 users. We have used these samples to create test datasets, each containing video and inertial sensor samples fabricated according to one of the attacks we introduced.

Our cross-validation tests show that Movee achieves an accuracy that ranges between 68 and 93 percent on our attack datasets created on a Samsung Admire smartphone. On a Google Glass device, Movee's accuracy ranges between 76-91 percent for the attacks tested. We conjecture that Movee's improved accuracy on Google Glass on the more complex attacks is due to the ability of head mounted cameras to capture more complex motions, that are harder to plagiarize.

The paper is organized as follows. Section 2 presents the system and adversarial model, including the novel attacks we introduce. Section 3 introduces Movee and Section 4 describes its implementation. Section 5 describes our data collection process and Section 6 evaluates the performance of Movee. Section 7 describes the limitations of this work. Section 8 details related work and Section 9 concludes.

2 The Model: System and Adversary

We now describe the system and adversary models that we assume in this work.

2.1 System Model

We consider a system that consists of a service provider, (e.g., video sharing services such as Vine [13], YouTube [16] or check deposit services [17], [33]). The provider offers an interface for subscribers to upload or stream videos they shot on their mobile devices.

We assume subscribers own mobile devices equipped with a camera and inertial sensors (i.e., accelerometers). Devices have Internet connectivity, which, for the purpose of this work may be intermittent. Subscribers need to install an application on their mobile devices, which we henceforth denote as the "client".

A subscriber needs to use this client to capture videos. In addition to video, the client simultaneously captures the inertial sensor (accelerometer) stream from the device. The client uploads both the video and the accelerometer streams to the provider. The provider verifies the authenticity of the video by checking the consistency of the two streams. The verification is performed using limited information: the two streams are from independent sources, but have been captured at the same time on the same device.

We assume a system where the problems of establishing trust in the mobile device, operating system and associated drivers, and the mobile client are already addressed. This includes for instance a system where a a chain of trust has been established [23], [45], [60]. The chain of trust ensures that the operating system, including the camera and sensor device drivers, and the installed apps, are trusted and have not been tampered with by an attacker, see e.g., [4], [6]. A discussion of limitations is included in Section 7.

In the remainder of the paper we use the terms accelerometer and inertial sensor interchangeably.

2.2 Adversary Model

We assume that the service provider is honest. Users however can be malicious. An adversarial user can tamper with or copy video streams and inertial sensor data. The goal is to fraudulently claim ownership of videos they upload to the provider. Let V be such a video. The adversary can use a trusted device to launch the following attacks, that produce fraudulent videos or fraudulent video and acceleration data:

Copy-paste attack. Copy V and output it.

Projection attack. Point the camera of the device over a projection of the target video. Output the result.

Random movement attack. Move the device in a random direction, and capture the resulting acceleration data. Output the video V and the captured acceleration stream.

Direction sync attack. Use the video to infer the dominant motion direction of V. Use the device to capture an acceleration sample that encodes the same motion direction. Output V and the acceleration sample.

Cluster attack. Capture a dataset of videos and associated sensor streams. Use a clustering algorithm (e.g., K-means [36]) to group the videos based on their movement (i.e., the values of video features extracted by the VMA module, see Sections 3.1 and 3.4). Assign V to the cluster containing videos whose movement is closest to V. Randomly choose one of the videos and associated sensor streams in the cluster, (V', A'). Output (V, A').

Replay attack. Study the target video V. Then, holding a mobile device that captures acceleration data, emulate the movements observed in V. Let A' be the acceleration data captured by the device during this process. Output (V, A').

3 Movee: Solution Overview

We introduce Movee, a system to verify the live capture of videos uploaded from mobile devices. Movee performs an analysis based on the consistency between the motion inferred from the simultaneously and independently captured camera and inertial sensor streams. If the data from the inertial sensor corroborates the data from the camera, Movee concludes that the video was genuine: it has been taken by the user pointing the camera to a real scene.

The Movee client is intended to be installed in mobile devices as part of special purpose video capture apps. When the user wants to capture a video or a photo, the client performs two actions simultaneously: First, it turns the camera on and starts to capture video frames; second, it starts to collect a stream of accelerometer readings.

The data from the camera sensor is stored in periodically captured image frames. The data from the inertial sensor in most mobile devices comes in the form of periodically captured acceleration magnitudes on three main axes as measured by the accelerometer. Movee infers the direction and the magnitude of motion from these two different types of sensor data.

Fig. 1 shows a diagram of Movee. The VMA module uses an efficient image processing method to infer a motion vector over the timeline of the video from frame-by-frame progress. The IMA module, converts the raw inertial sensor readings into a motion vector over the same timeline. Subsequently, the SC module extracts features which represent agreements and differences between the two motion data, from the VMA and IMA modules. The final decision of whether the captured video is genuine is made in the *classification* module. The classification module uses trained classifiers on the data produced by the SC module to find out whether the inertial sensor data corroborates the video sensor data.

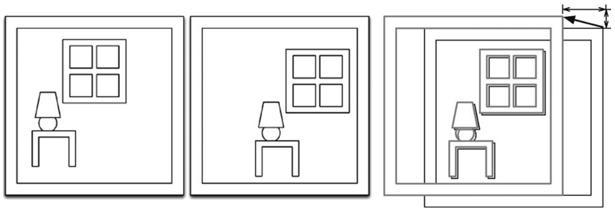


Fig. 2. The video motion analysis module processes each consecutive video frame and finds the motion vector by computing the amount of displacement that common image components have shifted between two frames.

In the rest of this section, we detail each of these modules.

3.1 Video Motion Analysis

The video motion analysis module takes as input the captured video stream and outputs an estimate for the direction and magnitude of the camera movement. The output of VMA is then used by the similarity computation module of Movee (see Fig. 1).

VMA first retrieves the frame per second (fps) rate of the stream and each available frame. In a pre-processing step, it applies a Hamming window [57] filter to eliminate noise from each frame.

Algorithm 1. Pseudocode of the Video Motion Analysis Module. The VideoShift Operation Computes and Returns the Total Displacement on One Axis, as Computed from Video Frames.

```
1.  Object implementation VMA;
2.    Operation double videoShift(Video V)
3.      int N := V.getFrameCount();
4.      double totalShift := 0;
5.      h := createHammingWindow();
6.      for i := 1 to N-1 do
7.        Frame $f_1$ := V.getFrame(i);
8.        Frame $f_2$ := V.getFrame(i+1);
9.        totalShift += phaseCorrelate($f_1$, $f_2$, h); od
10.     return totalShift;
11. end
```

For each pair of consecutive frames, VMA needs to find the movement of the camera. It is possible to perform this operation manually: First, print the photos on transparency films. Then *shift* one sheet placed on top of the other and keep *comparing* the two prints until they line up with minimal difference. The amount that the edges of one sheet overhang the other represents the offset between the photos (see Fig. 2). The common optical mice [8] use this principle to determine pointer movement from a stream of images taken with a low resolution optical sensor mounted to their bottom side. The movement inferred from this analysis will be limited to two axes, i) horizontal along the X axis, and ii) vertical along the Y axis.

Phase Correlation. VMA uses the *shift* and *compare* principle as well, by applying it on all consecutive frames of the video (see Algorithm 1). The result is a frame-by-frame displacement vector. However, it would have been prohibitively expensive to compute the differences between two frames for all possible pixel shifts, especially considering how large each frame is.

Instead, we use *Phase Correlation* [31] to find the shift that minimizes the difference, by carrying the computation into the frequency domain. Phase correlation is an image processing technique that computes the spatial shift between two similar images (or sub-images). It is based on the Fourier shift property: a shift in the spatial domain of two images results in a linear phase difference in the frequency domain of the Fourier transform (FT) [32]. It performs an element-wise multiplication of the transform images. It then computes the inverse Fourier transform (IFT) of the result, and finds the shift that corresponds to the maximum amplitude. This yields the resultant displacement. The maximum amplitude can be defined in the two-dimensional surface with delta functions (colloquially referred to as *peaks*) at the positions corresponding to spatial shifts between the two images.

Then, for each pair of consecutive frames, VMA applies the phase correlation method to obtain linear shifts between images in both *X* and *Y* directions (see Algorithm 1). It then computes the *cumulative shift* along the *X* and *Y* axes by adding up the linear shifts for all consecutive frames retrieved from that video. Let $VS_{x,i}$ and $VS_{y,i}$ denote the cumulative video shifts of the ith frame on the X and Y axes. We use $VS_{x,i}$ and $VS_{y,i}$ as feature descriptors (see Section 3.4).

Extract motion direction from video. Based on the computed cumulative shift along the X and Y axes, VMA infers the camera direction of movement. For instance, if the camera is in landscape orientation, and the cumulative shift over the *X* axis is negative, $VS_{x,i} < 0$, and dominates the one over the *Y* axis, (i.e., $|VS_{x,i}| \gg |VS_{y,i}|$), the video motion direction is to the right. Table 1 (first and second columns) summarizes the direction inference process. We use the notation X-- and X++ to denote negative and positive cumulative shifts along the *X* axis that dominate shifts along other axes.

3.2 Inertial Sensor Motion Analysis

The Inertial sensor motion analysis module (see Fig. 1) relies on the accelerometer sensor widely available in mobile devices. The IMA processes the data from the accelerometer in order to produce a motion direction and magnitude which is then compared in the similarity computation module with the output from the VMA module.

The inertial sensor coordinate system is defined relative to the screen of the phone in portrait orientation. The *X* axis is horizontal and points to the right, the *Y* axis is vertical and points up and the *Z* axis points towards the outside of the front face of the screen (coordinates behind the screen have negative Z values). Let $\{(A_{x,i}, A_{y,i}, A_{z,i}) | i = 1..m]\}$ denote the accelerometer trace, recorded every T seconds. $(A_{x,i}, A_{y,i}, A_{z,i})$ is the ith sample, containing accelerometer readings on the three axes. Instead of faster accelerometer sampling modes (e.g., "fastest" or "game" modes) we chose the slower but less noisy 16 Hz mode.

Filtering step. In a pre-processing step, at each sampling time T, IMA removes duplicate (noise) acceleration values that occur (i) at time T, phenomenon that frequently occurs at the beginning of the capture interval, and (ii) within

TABLE 1
Camera Motion Inference Based on Cumulative Shifts Along *X* and *Y* Axes Inferred from the Video and Inertial Sensor Streams

Direction	Video Shift	Sensor Shift
Up	Y++	X++
Down	Y--	X--
Left	X++	Y++
Right	X--	Y--

The device is considered to be in landscape orientation. X++ (and X−−) denote positive (and negative) X *axis shifts that dominate shifts along other axes.*

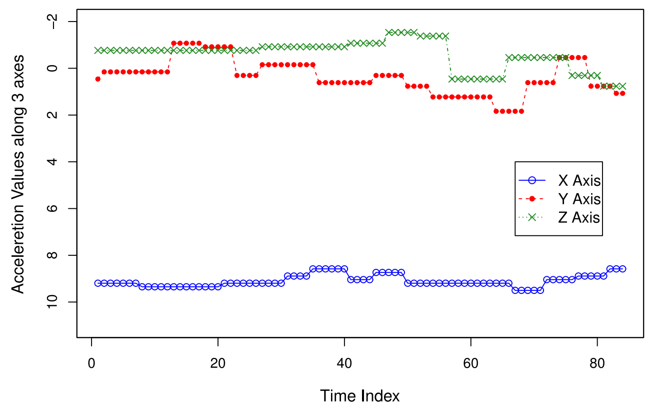
interval $[T - 10\text{ ms}, T + 10\text{ ms}]$. Furthermore, IMA uses a combination of low-pass and high-pass filters to remove the effects of gravity from the recorded acceleration stream. In a first, low-pass filter, let $G_{a,i}$ be the filtered gravity value on the a axis ($a \in \{X, Y, Z\}$) in the ith sample and let $G_{a,i+1}$ be the gravity value to be filtered in the current, $(i+1)$th sample. $A_{a,i+1}$ is the acceleration reading on the a axis for the $i+1$th sample. Then, $G_{a,i+1} = \alpha G_{a,i} + (1-\alpha) A_{a,i+1}$, $\forall a \in \{x, y, z\}$. We have experimented with values of α ranging between 0.6 and 0.95. We have found the value $\alpha = 0.8$ to perform best.

Subsequently, IMA passes the result through a high-pass filter, $FA_{a,i+1} = A_{a,i+1} - G_{a,i+1}$, where $FA_{a,i+1}$ denotes the filtered acceleration value on the a axis, $\forall a \in \{x, y, z\}$, for the $(i+1)$th sample.

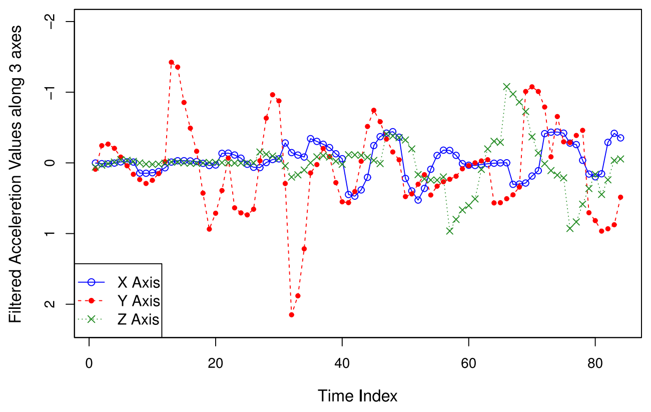
Fig. 3b shows the effects of filtering for the sample raw acceleration of Fig. 3a), where the phone was held in landscape orientation. Fig. 3a shows that the gravity primarily influences the *X* axis, with acceleration values being changed around 9.19 (g value for the phone when in fixed position). However the gravity value also affects the acceleration readings on other axes. The filtering method eliminates the gravity affect.

Extract motion direction from acceleration data. Fig. 3b shows that the *Y* axis movement is dominant (the highest translation/shift variations), thus the direction of movement is to the right in landscape orientation of the phone. Table 1 (first and third columns) summarizes the direction inference process also for acceleration data. For instance, given the cumulative shifts, *AS*, on all acceleration axes, if the device is in landscape orientation, $AS_{y,i} < 0$ and $|AS_{y,i}| > thr \times AS_{x,i}$ (for a given threshold thr), the accelerometer motion direction is to the right. We denote this situation through the notation Y--. We chose the thr value experimentally to be larger than 1.5, after analyzing sample data for different directions.

Extract motion distance from acceleration data. The $dIntegral$ function used in Algorithm 2 uses the acceleration data to infer the displacement, as follows. Given acceleration data on each axis, $A_{a,1}, ..A_{a,m}$, where $a \in \{X, Y, Z\}$, captured every T seconds, IMA computes the position (relative to the starting point) using a double integral. We adopt the trapezoidal rule [37] to approximate the definite integral $\int_c^d f(x)dx$, representing the area below the curve. The integration step is first applied to obtain velocity ($vel_{a,i} = vel_{a,i-1} + \frac{A_{a,i}+A_{a,i-1}}{2} * T$). In a second application, the integration retrieves the position ($pos_{a,i} = pos_{a,i-1} + \frac{vel_{a,i}+vel_{a,i-1}}{2} * T$). $vel_{a,i}$ and $pos_{a,i}$, $i = 1..m$,



(a)



(b)

Fig. 3. (a) Raw accelerometer data. (b) Filtered accelerometer data. The Y axis is the dominant axis for the direction and orientation of the device.

denote the velocity and position at the i-th sample on the axis a. The resulting position shifts are combined to obtain the cumulative shift, $AS_{x,i}, AS_{y,i}, AS_{z,i}$, along each axis. $AS_{x,i}, AS_{y,i}, AS_{z,i}$ are then used as feature descriptors (see Section 3.4).

Algorithm 2. Pseudocode of the IMA module. The Sensorshift Operation Computes and Returns the Total Displacement on One Axis, as Computed from Instantaneous Accelerometer Readings.

```
1.  Object implementation IMA;
2.    double Th;          # Threshold
3.    Direction dr;        # movement direction
4.    Operation double sensorShift (sensorData S)
5.      double totalShift := 0;
6.      int N := S.getSensorLogCount();
7.      for i := 1 to N do
8.        if (i=1) then
9.          totalShift += dIntegral(S[i],0);
10.       else
11.         totalShift += dIntegral(S[i],S[i-1]);
13.     return totalShift;
14. end
```

3.3 Similarity Computation

The similarity computation module compares the two motion sequences computed by the VMA and the IMA modules. It returns a set of features that summarize the nature of the similarity between the two sequences. The features are then used by the classification module (see Section 3.4) to decide whether the two motion sequences corroborate each other, thereby concluding whether the video is genuine or not. The video motion and inertial sensor streams encode the same user hand movement, which are processed by the VMA and IMA modules respectively (see Fig. 1) to each yield a motion stream.

To compute their similarity, we use a well-known sequence similarity measurement method from speech and pattern recognition, called dynamic time warping [19], [53]. Similar to the well-known string edit distance, DTW is a dynamic programming solution to find the minimum cost set of operations that converts one sequence to the other.

In this section, we describe how we adapted the DTW algorithm to the practical issues in comparing the two motion sequences from the VMA and IMA modules. The two sequences differ in their number of samples, and have different magnitudes due to the nature of their source sensors. The VMA sequence length is proportional to the number of video frames, whereas the IMA sequence length is proportional to the product of the sample rate of the inertial sensor and the length of the recording interval. Thus, we first perform a *stretching* step to make sure that the sequences output by VMA and IMA are of same length.

Furthermore, the motion sequence that the VMA infers from the video stream does not take into account the distance of objects into the camera. This may result in the same motion being registered as faster when the objects are close to the camera, and slower when the objects are far. To address this problem, in a second step we perform a *calibration* process: compute a coefficient to match the average speed of the motion the video stream to that of the inertial sensor stream.

In the rest of this section, we first briefly detail the DTW algorithm, then present the stretching and calibration processes. We provide justification to the use of these methods with observed improvements in the resulting accuracy that the system gains after processing the features in the classification module.

3.3.1 *Dynamic Time Warping*

Let $\mathcal{F}$ be a feature space. Let $X = (x_1, x_2, .., x_n)$ and $Y = (y_1, y_2, \dots, y_m)$, $n, m \in \mathbb{N}$, be time-dependent vectors, $x_i, y_j \in \mathcal{F}$, $i = 1..n, j = 1..m$. DTW computes the (n, m)-warping path of X and Y, that is a sequence $P(X, Y) = (p_1, \dots, p_L)$, where $p_l = (i, j) \in [1 : n] \times [1 : m]$, $\forall l \in [1 : L]$. The warping path satisfies boundary, monotonicity and step size conditions. At each step, DTW has the option to perform one of the following three *moves*, illustrated in Fig. 4: a diagonal (or match) move, an expansion move, or a contraction move. The cost of a warping path $P(X, Y)$ is defined as the sum over the costs of all the moves in the path. The goal of DTW is to find a warping path of minimal cost among all possible warping paths.

Movee uses a variation of the DTW algorithm: the variable penalty dynamic time warping (VPdtw) [28]. This is because the process of expanding and contracting the time

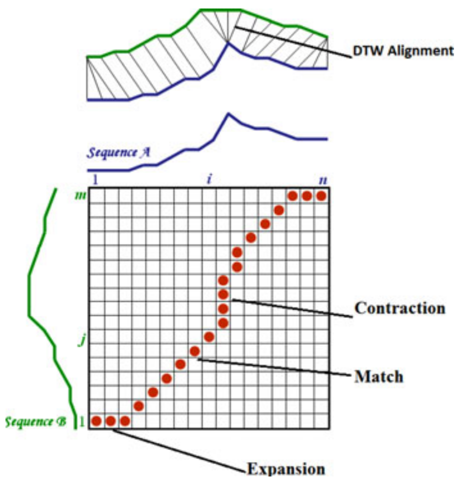


Fig. 4. Illustration of DTW alignment for two time-dependent sequences. The red dots show the optimal warping path. A *diagonal (match) move* is a match between the two sequences. An *expansion* duplicates one point of one sequence and a *contraction* eliminates one of the points.

axis of a sensor stream can produce a very high quality alignment to a video stream. However, excessive numbers of expansions and/or contractions can often result in matches at random parts of the streams and appear artificial rather than catching the genuine common movement patterns. VPdtw uses a penalty to constrain the use of expansions and/or contractions. This penalty is incurred whenever a non-diagonal (i.e., expansion or contraction) move is taken (see Fig. 4).

Let L denote the length of the longer sequence between the video and inertial sensor sequences for each sample. We extract several characteristics of the computed DTW alignment as feature descriptors, to be used by the classification module (see Section 3.4). First, the *normalized penalty cost*, defined as the penalty cost divided by L. Second, the *ratio of overlap points*, which is the number of overlap points between the two streams, divided by L. Third, the *ratio of diagonal moves*, which is the number of diagonal moves divided by L. Fourth, the *ratio of expansion moves*, which is the number of expansion moves divided by L. Finally, the *ratio of contractions moves*, which is the number of contraction moves divided by L. The normalization to L ensures that the values are independent of the sample length.

3.3.2 Stretching

The sensor and video streams are sampled at different rates, thus the two vectors are of different length. The stretching step extends the shorter sequence (length s) to the length of the longer sequence (l). We use linear interpolation to compute $l - s$ new points for the shorter sequence. In Sections 6.3 and 6.4 we show that depending on the attack type, the use of stretching improves the accuracy of Movee in differentiating fraudulent from genuine videos by a rate of 5-12 percent. This result is illustrated in Fig. 5b, where the use of stretching significantly improves the ability of the DTW procedure to align the video and inertial sensor movement streams when compared to Fig. 5a.

3.3.3 Calibration

An artifact of the method used in the video motion analysis module is that the same motion pattern can be registered as faster when the objects in the view are close to the camera, and slower when the objects are far. In order to compensate for this artifact, we calibrate the speed of the video motion vector with a coefficient to match that of the speed of the inertial sensor motion vector.

The goal is to compute a calibration factor CF, that is used to multiply all the points in the video stream. We have explored several calibration methods, including mean based and linear curve fitting. We provide details however only on the two methods that performed the best in our experiments, *truncated mean* and *polynomial curve fitting*.

Truncated mean. The truncated mean computes the mean after discarding the high and low ends of the probability distribution (see Fig. 6b). We apply this concept as follows: For each pair of points in the sensor and video vectors, compute their ratio and add it to a *ratio vector*. Compute the truncated mean of the ratio vector, discarding 12.5 percent from both the low and the high ends of the distribution.

Polynomial curve fitting. Polynomial curve fitting [29] constructs the polynomial that has the best fit to a series of data points (see Fig. 6c). To compute the coefficients that best fit the curve to the given data, we use the least squares method [29] to minimize the error between the data and the fitted polynomial [29]. Let SP_s denote the average value over the points on the fitted curve for the sensor stream and let SP_v denote the average value of the points on the curve of the video stream. Compute the calibration factor as $CF = \frac{SP_s}{SP_v}$.

Figs. 6b and 6c show sample calibration outputs for these two methods, when compared to the uncalibrated version shown in Fig. 6a.

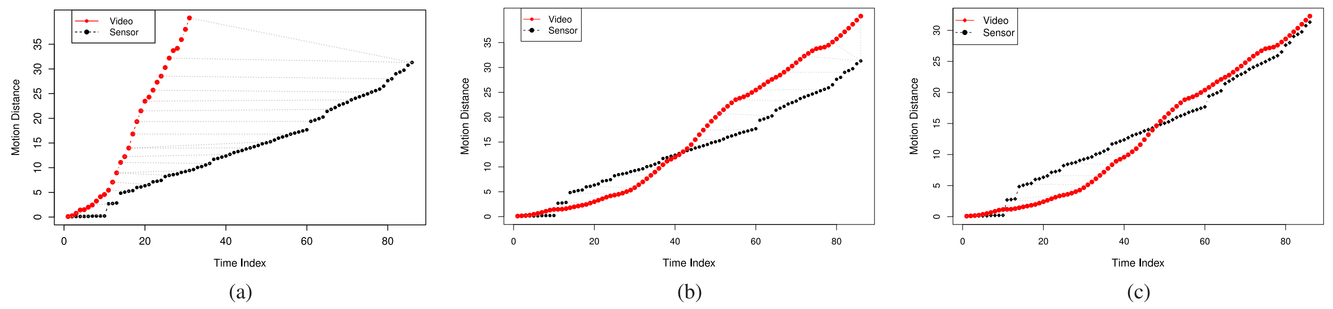


Fig. 5. Example alignment of video and inertial motion streams extracted from the same experiment: (a) When using only DTW. (b) When stretching the shorter vector and applying DTW. (c) After stretching *and* calibration and applying DTW. Stretching helps achieve a significant alignment improvement.

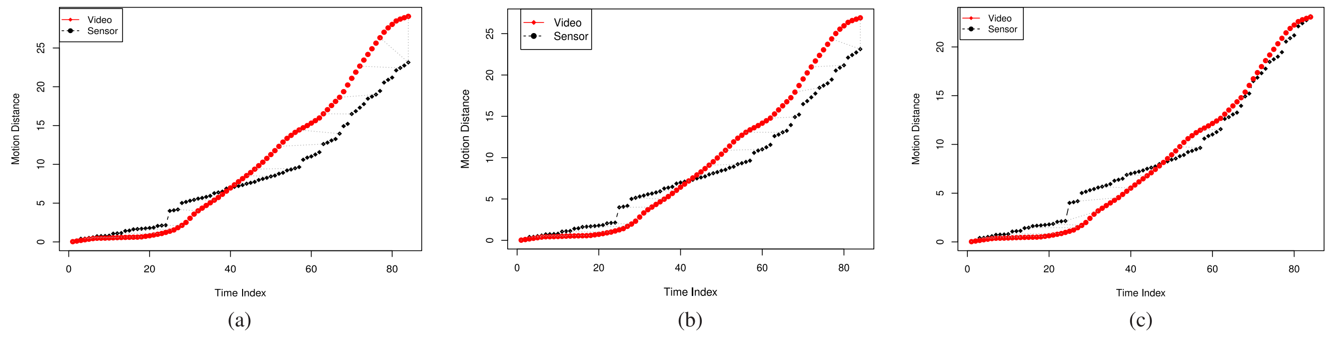


Fig. 6. The video motion analysis module has a limited ability to determine the average speed of the device motion. For this reason, we use the average speed calculated from the inertial sensor motion vector to calibrate that of the video motion vector. This figure shows the effect of calibration in the similarity computation. (a) No calibration. (b) Truncated mean calibration. (c) Polynomial curve fitting calibration.

3.3.4 *Example Alignment*

To illustrate the need for the DTW, stretch and calibration steps previously described, we provide here experimental results of their use on a genuine sample of video and inertial sensor streams, captured using Movee (see Section 4 for implementation details). Fig. 5a shows the alignment between the video and inertial sensor streams when only DTW is used. Fig. 5b shows the resulting alignment when DTW and stretching are applied. Finally, Fig. 5c shows the alignment achieved when DTW is applied along with stretching and calibration. The experiment shows that stretching is vital to achieve a good alignment, while calibration further improves the quality of the alignment.

3.4 Classification

The similarity computation module produces 14 features that represent the nature of the similarity between the motion information inferred from the video stream and the one observed from the inertial sensor data. The features are: (1) the movement direction of the target from the center of the screen (see Section 4), (2-5) the cumulative shift of the video and accelerometer on the x and y axes (four descriptors), (6) the video motion direction, (7) the sensor motion direction, (8) the DTW distance after stretching and calibration steps, (9) the calibration factor, CF, (10) the normalized penalty cost, (11) the ratio of overlap points, (12) the ratio of diagonal moves, (13) the ratio of expansion moves and (14) the ratio of contractions moves.

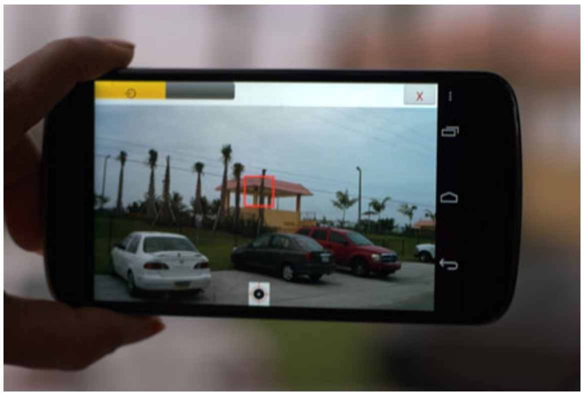


Fig. 7. Movee in action on smartphone: Target icon (bullseye) at the bottom of the screen shows the direction in which the user needs to move the camera.

The classification module runs trained classifiers over these features to determine whether there is sufficient evidence that the video stream is genuine. Section 6.1 describes the classifiers used in our experiments.

4 Movee Implementation

We have implemented a Movee client using Android and a server component using C++, R and PHP. We used the Open Source Computer Vision (OpenCV) library [7] for the video motion analysis. The client allows users to capture movies and simultaneously provide proofs of liveness. Fig. 7 shows a snapshot of Movee in action, on a smartphone. When starting the client, the user is presented with an initial screen that instruct her to hold the device firmly before pressing the start button. This is done to prevent initial accelerometer reading errors. Once the user presses the start button, a target appears (bullseye). The user is instructed to move the camera in the direction of the target. Once the user starts to move the camera toward the target, the target begins to move toward the center of the screen. The target moves at a speed that ensures that the process takes at least 6 s.

We call this 6 s long process, the *verification* interval. During the verification interval, the Movee client captures the video stream and logs the accelerometer data. After the verification interval, the user can continue capturing the intended scenes. We were inspired by Vine[1] to choose the verification interval to be 6 s. This choice presents the additional advantage that it keeps the size of the video file small (around 150 KB in the Samsung Admire Phone), and reduces the communication overhead.

We have also implemented MoveeG, a Movee app variant for the Google Glass, see Fig. 8. We have used the glass development kit (GDK) to build MoveeG as a glassware that runs directly on Glass (around 700 lines of code). MoveeG starts and stops by voice command or through a tap based menu. Since the built-in camera activity has limited functionality, we have built our own logic with the Android Camera API [2], to capture videos. Once the video capture is completed, MoveeG sends the captured video and accelerometer streams to a server over the Glass Wi-Fi connectivity, using HTTP POST requests.

1. Vine [13] is an application that allows users to create and post (on Twitter, Facebook) video clips.

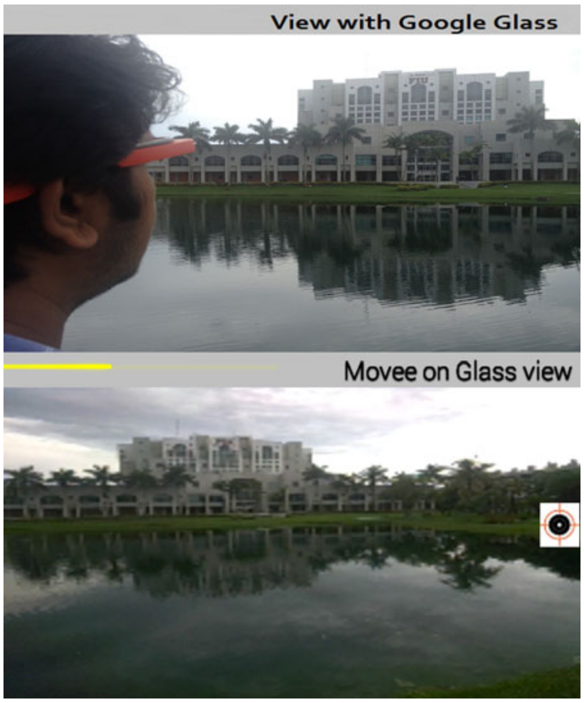


Fig. 8. Movee on Google Glass: Top snapshot shows view with Google Glass. Bottom snapshot shows view from Movee on Glass perspective. Target icon (bullseye) at the right of the screen shows the direction in which the user needs to move the head mounted glass.

5 Data Collection

We have used the implemented Movee and MoveeG applications to collect video and acceleration samples from real life users. We have worked with the Institutional Review Board (protocol number IRB-13-0582) at FIU to ensure an ethical interaction with the users during the collection process.

We used the 3D accelerometers, available in most recent smartphones, tablets and Google Glass, to acquire motion acceleration data. We have collected smartphone data using a Samsung Admire with a frames per second (fps) rate of 14, that samples accelerometer readings at 16.67 Hz [11] mode. We have collected Google Glass data using a Glass device that samples accelerometer readings at 50 Hz [11].

We have collected data from a total of 13 users, in multiple rounds. We have selected participants from FIU's student body and campus visitors. Nine of the users are males and four are females. Their age ranges between 23 and 32 yo., and their occupations include biologist, fashion designer, housewife, software, civil and electrical engineers.

5.1 Smartphone Data Collection

Random and direction sync attack datasets. We have first collected "genuine" video and accelerometer data from a subset of 10 participants (seven male, three female). Each participant was asked to use Movee, following the instructions shown on the screen (see Section 4). We have collected 10 well defined (6 s long) samples from each user; the total of 100 samples are stored in a "genuine" dataset.

We have used the genuine dataset to generate random and direction sync attack datasets as follows. Each of these datasets contains an equal number of genuine and fraudulent video and acceleration samples. The "random attack" dataset consists of 50 video and corresponding acceleration samples from the genuine dataset, and 50 fraudulent samples created according to the Random attack. Specifically, each fraudulent "random" video and acceleration sample is created from one genuine sample, by coupling its video with the acceleration data of another, randomly chosen sample.

Similarly, the "direction sync attack" dataset contains the other 50 genuine samples from the genuine set, and 50 fraudulent samples created according to the Direction Sync attack. That is, each fraudulent sample couples the video of one genuine sample with the acceleration data of another genuine sample, with the same direction of movement.

Cluster attack dataset. We have generated the "cluster attack" dataset from data we collected from a subset of 12 participants (eight male, four female). The participants were given the freedom to move (themselves and the smartphone) in any direction for at least 15 s but no more than 30 s, while using Movee. We have collected a total of 141 samples of 15-30 s long video and acceleration streams.

Similar to the random and direction sync attack datasets, we have built the cluster attack dataset to consist of an equal number of genuine and fraudulent video and corresponding acceleration samples. For this, we introduced the following variation of the Cluster attack introduced in Section 2. First, we divided each of the 141 samples into 6 s long chunks. Second, we ran K-means clustering [36] at the chunk level, to cluster the chunks according to the motion determined by the VMA module (see the cluster attack in Section 2). This resulted in 423 genuine chunks of video and corresponding acceleration stream. We applied the v-fold cross-validation [58] algorithm to automatically determine the number of motion clusters in the data. The v-fold cross-validation step produced $K = 6$ for our cluster dataset. Then, for each of the 423 genuine chunks, we randomly chose another chunk from the same motion cluster. Third, we coupled the video from the first chunk with the acceleration stream of the randomly selected chunk. We added each such fraudulent sample to the "cluster attack" dataset. Thus, this dataset contains 423 genuine and 423 fraudulent (video, acceleration) chunks.

Replay attack dataset. We have also collected a "replay attack" dataset built according to the attack described in Section 2. Similar to the cluster attack dataset, the replay attack dataset also consists of 423 genuine and 423 fraudulent chunks.

5.2 Google Glass Data Collection

We performed a similar data collection process for a Google Glass device. First, we collected data from a subset of five users (three male, two female): 20 well defined 6 s long samples from each user. The total of 100 samples form a "genuine" dataset. Similar to Section 5.1, the "random attack" dataset for Glass, contains 50 genuine samples and 50 fraudulent samples created according to the Random attack. The "direction sync attack" dataset for Glass, contains the other 50 genuine samples and 50 fraudulent samples created according to the direction sync attack.

For the cluster and replay attack datasets we have collected 84 genuine samples (30 s long each), resulting in a

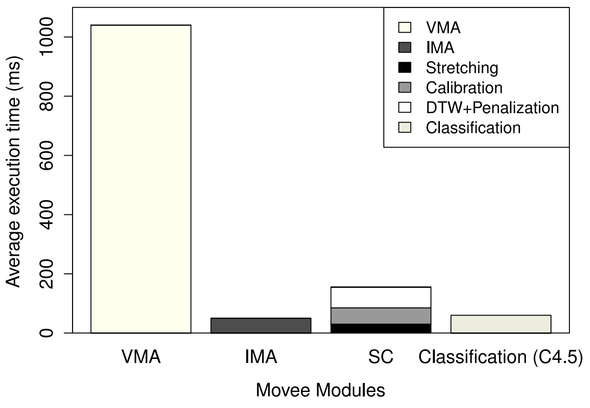


Fig. 9. Movee (per-module) server side overhead: video processing is the most expensive. The total cost is however under 1.3 s.

total of 420 chunks of 6 s each. Then, each fraudulent cluster and replay chunk is created as specified in Section 2. Thus, each of the Glass cluster and replay attack datasets contains 420 genuine and 420 fraudulent chunks.

6 Evaluation

In this section we present experimental results for Movee and MoveeG. We first describe the experimental setup. Second, we evaluate the overhead of the liveness analysis on the server. Furthermore, we evaluate the performance of Movee and MoveeG on the attack datasets introduced in Sections 5.1 and 5.2. Finally, we evaluate the impact of MoveeG on the battery lifetime of a Google Glass device.

6.1 Experiment Setup

The classification module (see Section 3.4) runs trained classifiers to determine whether there is sufficient evidence that a video stream is genuine. We have used several classifiers, including multilayer perceptron (MLP) [35], decision tree (C4.5), random forest (RF) [25], Bagging and Random Tree [21].

We have applied 10-fold cross-validation tests [40] to assess how the results of the statistical analysis will generalize to an independent data set. Specifically, the ground truth data set is randomly partitioned into k equal sized subsets. k-1 subsets are used for training the model and the last subset is used for testing the model. This process is repeated k times (the folds), with each of the k subsets used exactly once for validation. The k results from the folds are averaged to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once. We have used the Weka version 3.7.9 data mining suite [14] to perform the experiments, with default settings: For the backpropagation algorithm of the MLP classifier, we set the learning rate to 0.3 and the momentum rate to 0.2.

Metrics. We briefly define the metrics we use to evaluate the accuracy of Movee. The true positive rate (TPR) metric denotes the fraction of videos correctly identified as genuine, the false positive rate (FPR) denotes the fraction of videos incorrectly identified as genuine and the false negative rate (FNR) denotes the fraction of videos incorrectly identified as fraudulent. The receiver operating characteristic (ROC) curve [9] is a visual characterization of the trade-off between the FPR and the FNR. The equal error rate (EER) [59] is the rate at which both accept and reject errors are equal. A lower EER denotes a more accurate solution. The area under the ROC curve (AUC) is equal to the probability that a classifier will rank a randomly chosen genuine sample higher than a randomly chosen fraudulent one. An area of 1 represents a perfect test; an area of 0.5 represents a worthless test.

TABLE 2
Detailed Accuracy Results of Movee on the Smartphone Random and Direction Sync Attacks

Attack	Classifier	Acc(%)	TPR(%)	FPR(%)	FNR(%)
Random	MLP	92	93.33	9.09	6.67
	RF	91	88.89	7.84	11.11
	C4.5	90	91.11	10.9	8.89
	Bagging	81	82.0	20.0	18.0
Dir sync	MLP	78	85.71	31.8	14.29
	RF	78	82.14	27.27	17.86
	C4.5	84	82.14	13.63	17.86
	Bagging	76	82.14	31.81	17.86

For the random attack, Movee with MLP achieves an accuracy of 92 percent. For the more effective direction sync attack, Movee using Decision Tree (C4.5) achieves an 84 percent accuracy.

During the experiments, we have tested Movee on a Samsung Admire smartphone running Android OS Gingerbread 2.3 with an 800 MHz CPU. We have tested MoveeG on a a Google Glass running Android OS KitKat 4.4.2 with OMAP 4430 dual-core ARM Cortex-A9 CPU and 682 MB of RAM. We have used a Dell laptop equipped with a 2.4 GHz Intel Core i5 processor and 4 GB of RAM for the server.

6.2 Server Overhead

Fig. 9 shows the overhead (divided into modules) of the Movee liveness analysis on the server, running on the Dell laptop, for 6 s videos. The values shown are an average over 10 experiment runs. It shows that the VMA is the most time consuming module, slightly exceeding 1 s. The IMA and classification components (running the C4.5 classifier) impose the smallest overheads, of 110 ms taken together. MLP takes an average of 940 ms and random forest an average of 140 ms. The overhead of the SC module is around 150 ms, with the smallest cost imposed by the stretching step and the highest cost by the penalty based DTW.

6.3 Movee Attack Detection

Movee prevents the Copy-Paste attack of Section 2: no sensor stream exists. Movee also detects the Projection attack: the motion registered in the projected video is likely inconsistent with the acceleration data of the device capturing the video. We now evaluate the ability of Movee to detect random, direction sync, cluster and replay attacks, on a smartphone. The next section studies the performance of MoveeG on the same attacks, but executed on a Google Glass.

We focus first on the random and direction sync attacks on the smartphone, using the corresponding attack datasets described in Section 5.1. Details of the three best performing classifiers, including TPR, FPR and FNR values are shown in Table 2. For the random attack dataset, the three classifiers perform similarly, with MLP being the top performer

TABLE 3
Detailed Accuracy Results of Movee on the Smartphone Cluster and Replay Attack Datasets

Attack	Classifier	Acc(%)	TPR(%)	FPR(%)	FNR(%)
Cluster	MLP	70.26	74.46	34.08	25.54
	RF	73.46	78.14	31.39	21.86
	Bagging	72.25	77.27	32.95	22.72
	C4.5	70.04	74.24	34.30	25.76
Replay	MLP	65.81	70.93	39.9	29.07
	RF	67.44	72.03	37.68	27.97
	Bagging	67.91	71.81	38.42	28.19
	C4.5	63.84	69.61	42.61	30.39

For both the cluster and replay attacks, Bagging achieves the best accuracy of 73 and 68 percent respectively.

(92 percent accuracy). For the direction sync attack dataset, Decision Tree (C4.5) outperforms MLP and RF with 84 percent accuracy.

Fig. 11a shows the ROC curve and the computed EER and AUC values for the MLP classifier on the random dataset. The EER value of MLP is as small as 0.08 (maximum is 0.5) and the area under the curve exceeds 0.95, denoting an accurate classifier.

We have also evaluated the impact of each step of the SC module on the accuracy of Movee, for the random and direction sync attack datasets. For each dataset, we measured the accuracy of the three classifiers when (i) no alignment phase was applied, (ii) when stretching and DTW were applied, (iii) for stretching, calibration and DTW, and (iv) for stretching, calibration and penalty based DTW. Fig. 11b shows the results for the random attack dataset and Fig. 11c shows the results for the direction sync attack datasets. The stretching step contributes the most to the accuracy of Movee for the random attack while the penalization step contributes the most for the direction sync attack (around 12 percent).

Table 3 shows the results of our experiments on the cluster and replay attack datasets. The replay attack is more effective. The Bagging algorithm achieves the best performance on both attacks, of 73 percent for the cluster and 68 percent for the replay attack. Fig. 10 summarizes the performance of the best four classifiers on the four attacks considered, on smartphone captured data. The more complex cluster and replay attacks are more efficient.

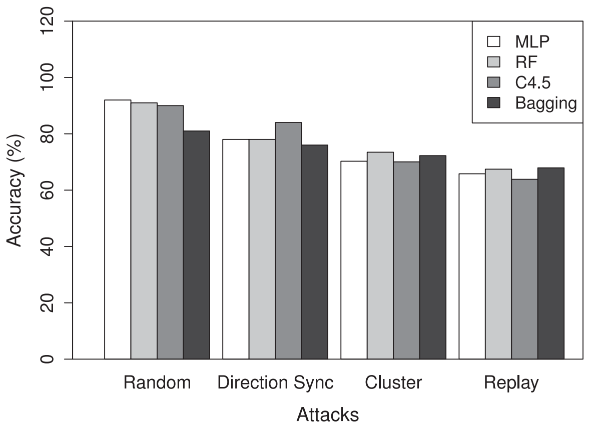


Fig. 10. Summary of Movee accuracy on smartphone random, direction sync, cluster and replay attack datasets. The accuracy (y axis) labels exceed 100 percent to fit the legend.

6.4 MoveeG Attack Detection

We first investigate the accuracy of MoveeG running on Google Glass to classify genuine and fraudulent video samples, on the random and direction sync attack. Table 4 shows the detailed results of the three best performing classifiers (RF, RT and MLP). RT performs best on the random and direction sync attack datasets. Fig. 12a shows the ROC curve and the computed EER value for the RF classifier and the random attack dataset for MoveeG. The EER value of RF is 0.07 and the area under the curve exceeds 0.97.

Fig. 12b shows the impact of the steps in the SC module for the random attack dataset and Fig. 12c shows their impact for the direction sync attack dataset. The stretching step contributes the most to the accuracy of MoveeG for the random attack (5 percent). For the direction sync attack, the penalization step contributes the most (around 8 percent).

We now evaluate Movee on the cluster and replay attacks performed on the data captured on the Google Glass device (see Section 5.2). Table 5 shows the results of the experiments. Similar to the smartphone investigation, the replay attack is slightly more effective than the cluster attack. Both attacks are more effective than the random and direction sync attacks. The Random Forest classifier achieves best accuracy both for the cluster (78 percent) and for the replay attack (77 percent). We observe a surprising result: Movee has higher accuracy on the cluster and replay attacks performed on Glass videos when compared to the

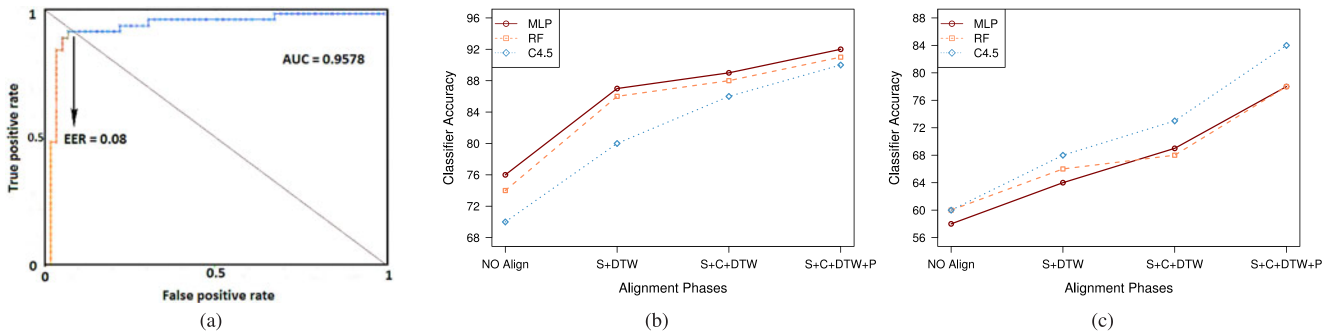


Fig. 11. Smartphone data evaluation. (a) ROC curve on random dataset for Movee when the MLP classifier is used. The EER of MLP is 0.08 (the maximum values is 1.0). (b) The impact of the SC steps on the Movee accuracy, for the three classifiers, for the random attack. Stretching provides the highest improvement (12 percent). (c) The impact of the SC steps on the Movee accuracy for the direction sync attack. The penalization brings a 12 percent improvement for the direction sync attack.

TABLE 4
Detailed Accuracy Parameters of Movee Glassware (Using Different Classifiers) for All Three Attack Datasets

Attack	Classifier	Acc(%)	TPR(%)	FPR(%)	FNR(%)
Random	MLP	90	87.5	7.7	12.5
	RF	90	89.6	9.6	10.41
	RT	91	88.89	6.52	11.11
Dir sync	MLP	72	74.5	31.1	25.4
	RF	74	61.4	16.1	38.6
	RT	79	68.2	12.5	31.8

"Acc" denotes the accuracy of the classifier.

TABLE 5
Detailed Accuracy Results of MoveeG on the Glass Cluster and Replay Attack Datasets

Attack	Classifier	Acc(%)	TPR(%)	FPR(%)	FNR(%)
Cluster	RF	78.36	79.3	22.5	20.7
	MLP	72.45	75.1	30.0	24.9
	RT	76.34	72.0	19.6	28.0
Replay	RF	77.86	81.8	26.0	18.2
	MLP	62.76	67.7	42.2	32.3
	RT	74.74	78.1	28.6	21.9

Random Forest achieves best accuracy for both cluster and replay attacks. Movee is more accurate on Glass than on smartphone.

smartphone videos. We conjecture that for the replay attack, the reason for this stems from the fact that head mounted cameras capture more complex motions, that are harder to emulate.

Fig. 13 summarizes the accuracy of Movee on all four attacks performed on Google Glass data. Different from the smartphone data investigation, where the cluster and replay attacks are more efficient, we note that on Google Glass data, MoveeG with Random Forest achieves higher accuracy on the replay attack than on the direction sync attack.

6.5 MoveeG Battery Impact

MoveeG impacts the battery lifetime of the Google Glass device. To properly evaluate and profile the components responsible, we first turned off the wireless interfaces and stopped all applications running on the Google Glass. We then performed the following experiments, each starting with a full battery charge. First, we ran MoveeG until the device ran out of battery. Second, we separately ran only the video capture, and third, we ran only the acceleration stream capture components of MoveeG. We have logged the battery level of the Google Glass device during these experiments.

Fig. 14 shows our results: when recording only acceleration data, the battery lasts 209 minutes, when recording only video the battery lasts 102 minutes, and when recording both (MoveeG), it lasts 86 minutes. As a baseline, we also ran the experiment with no application running on the Glass: on average, the battery lasted 3,034 minutes. We note that the device turns itself to sleep when not in use. This experiment shows that keeping the device continuously active even when only capturing acceleration data, significantly reduces the battery lifetime to 6 percent. The video capture activity further reduces the battery lifetime by 50 percent when compared to the acceleration capture activity.

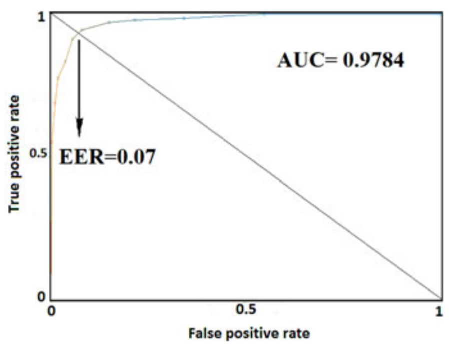
7 Limitations

Movee is designed to work with video streams captured by directly pointing the camera to the scene. Movee's liveness verifications will likely not work for videos captured in indirect ways (e.g., through reflection or refraction), or videos encoding fast motion. Movee will have difficulty extracting accurate motion information from video frames containing occlusions, illumination changes and blur.

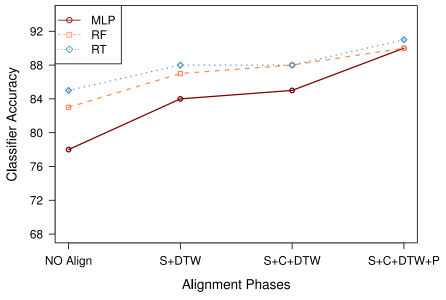
Movee's 6 s long verification step is a limitation of this work: it may impact usability. In future work, we will investigate seamless video verification techniques, that do not involve the user in this process.

We have also not experimented with very short videos (less than 6 s) or with videos shot in unusual circumstances: involving very high accelerometer activity, e.g., running, or when the user is in a moving vehicle. Due to the lack of gyroscope sensors in the Samsung Admire device, we have not integrated gyroscope readings to verify camera rotation movements.

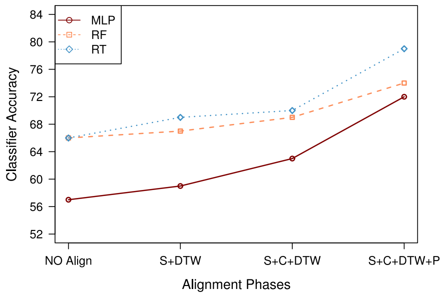
Furthermore, we have not experimented with doctored video and accelerometer streams. For instance, given an input video, the attacker can use the work of Davison et al. [30] to recover the 3D trajectory of the camera. Then,



(a)



(b)



(c)

Fig. 12. (a) ROC curve on random attack dataset for MoveeG when the MLP classifier is used. (b) The impact of the SC steps on the MoveeG accuracy, for the three classifiers, for the random attack. (c) The impact of the SC steps on the MoveeG accuracy for the direction sync attack/ Stretching provides the highest improvement (5 percent) for the random attack. The penalization step brings an 8 percent improvement for the direction sync attack.

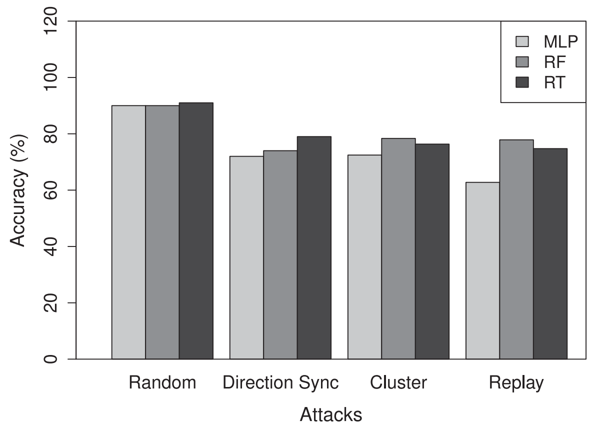


Fig. 13. Summary of Movee accuracy on data collected from Google Glass, when different classifiers are used.

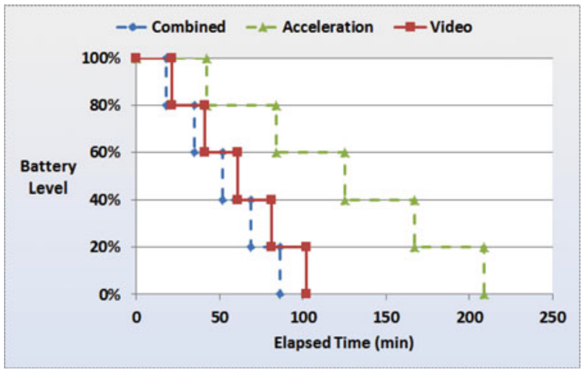


Fig. 14. Impact of video and acceleration recording on Google Glass battery lifetime. The video recording activity halves the battery lifetime when compared to the acceleration recording activity.

given root access (e.g., using [10], [12], create a corresponding accelerometer sample and feed it to Movee, e.g., using a solution similar to [15]. We defer the task of providing trust for the integrity of the mobile app, as well as the trust for the integrity of the device's connection to its camera and accelerometer sensors to the providers of Movee. Establishing the integrity of a mobile platform and mobile apps is currently an active area of research and is also being addressed by products readily available in the market [6], [54], [56].

As mentioned in Section 2, we assume a trusted device, operating system and device drivers (e.g., accelerometer), achieved e.g., using chain of trust solutions. This assumes an attacker with reasonable limitations. That is, circumventing the defenses would require the attackers to incur overwhelming time and effort costs. A successful attacker to a trusted app inside a trusted system satisfying our assumptions would have to successfully inject unsigned code into an operating system which uses chain of trust (e.g., iOS [4]) and modify an app which is protected by cryptographic mobile app verification (e.g., Arxan [6]) at the same time. Alternatively, a successful attacker who can produce both the fraudulent video and the corresponding acceleration streams would have to use a computer generated (CGI) video stream. Movee would not be effective in either scenario.

We have not experimented with "green screen" attacks, where the attacker captures a video with a portion of the scene being a green screen. Following the video capture, the attacker overlays additional video footage or static images on the green section. We note that Movee raises the bar here: an attacker needs to invest in additional equipment to thwart the defenses of Movee. The quality of the equipment determines the (in)ability of a human observer to detect the attack.

8 Related Work

This paper extends our earlier work [48] with (i) the cluster attack, (ii) an implementation of Movee on Google Glass (MoveeG), (iii) the collection of genuine and attack data from 13 users on a smartphone and a Google Glass device, and (iv) experimental results on the new data, and MoveeG's impact on the battery life of the Google Glass.

Video and acceleration. The combination of video and accelerometer data has been studied by Hong et. al. [38] in order to improve the compute-intense motion estimation in video encoding. They have shown that the use of accelerometer data improves the speed of the encoding process by a factor of 2-3. Moiz et. al. [42] introduced and developed a wearable, multi-modality, motion capture platform, and used its inertial and ultrasonic sensors to estimate position. The focus of our work is different, on verifying liveness of a video through the consistency of its video and accelerometer data.

Biometric liveness. Kollreider et al. [41] study the problem of verifying the actual presence of a live face in contrast to a photograph (playback attack) for face recognition based biometrics. They introduce a lightweight optical flow approach that estimates face motion estimation on the structure tensor and a few input frames. Park et. al. [47] introduce a liveness detection method for distinguishing a two-dimensional object from a three-dimensional object. The approach proposed uses video sequence images and does not require additional hardware or user interaction. Their work has direct application to face recognition biometrics: it can identify the use of a flat picture. Further work is needed to understand the vulnerability of this approach to photo movement and photo bending/3D printing attacks.

Multi-modal approaches relying on different sensor sets [26], [27], [34] have been proposed, to exploit the static and dynamic relationship between voice and face information from speaking faces for biometric authentication. Chetty [26] proposed liveness checking techniques for multimodal biometric authentication systems. Their techniques fuse acoustic and visual speech features and measure the degree of synchronization between the lips and the voice extracted from speaking face video sequences.

Pose estimation. Full-body human pose recognition from images is a fundamental problem in computer vision, that has been extensively studied, see e.g., [18], [24], [43], [44], [51], [55]. Rogez et al. [52] propose an efficient method to jointly localize humans and recognize their pose in images, using an exemplar based approach and fast search techniques. Murphy-Chutorian and Trivedi [44] survey work done in estimating head pose from images. Rodgers et al. [50] propose a probabilistic framework to detect articulated objects and their pose in 3D range scan data, without knowledge of the object orientation, in the presence of occlusion and clutter. Huang and Trivedi [39] introduce a framework to detect and track pose estimation of faces in video streams. Wang et al. [62] propose real-time multi-view face detection and pose estimation in video streams. Rehbinger and Ghosh [49] perform rigid body pose estimation using inertial sensors and a monocular camera. The detection and tracking of human and object pose in the captured videos can benefit Movee as long as we can identify consistencies between the changes in pose and the simultaneously captured acceleration information. One difficulty may arise from the presence of multiple humans and objects in captured videos. Movee achieves its goal through a simpler yet effective approach instead of trying to accurately perform pose estimation: it extracts and verifies the consistency of motion features from both the video frames and the acceleration stream.

In summary, this article introduces novel techniques for combining video and inertial sensor data to verify the liveness of a video stream. Movee verifies that the video has been shot live on a mobile device and does not plagiarize material from other sources. Movee does not require additional equipment, but requires the user to install and capture the video using a client application.

9 Conclusions

In this paper we have introduced the concept of “liveness” analysis, of verifying that a video has been shot live on a mobile device. We have proposed Movee, a system that relies on the accelerometer sensors ubiquitously deployed on most recent mobile devices to verify the liveness of a simultaneously captured video stream. We have implemented Movee, and, through extensive experiments, we have shown that (i) it is efficient in differentiating fraudulent and genuine videos and (ii) imposes reasonable overheads on the server. In future work we intend to integrate more sensors (e.g., gyroscope), as well as the use of Mono-SLAM [30] as an alternative VMA implementation to improve accuracy.

Acknowledgments

This research was supported in part by DoD grant W911NF-13-1-0142 and US NSF grant EAGER-1450619. A preliminary version of this article has appeared in ACSAC 2013. This research was supported in part by DoD grant W911NF-13-1-0142 and US NSF grant EAGER-1450619.

References

[1] Amazon [Online]. Available: http://www.amazon.com, Jul. 2015.

[2] Android Camera API [Online]. Available: http://developer.android.com/reference/android/hardware/Camera.html, Jul. 2015.

[3] eBay [Online]. Available: http://www.eBay.com, Jul. 2015.

[4] iOS Security [Online]. Available: https://www.apple.com/ipad/business/docs/iOS_Security_Feb14.pdf, 2014.

[5] Kickstarter [Online]. Available: http://www.kickstarter.com/, Jul. 2015.

[6] Mobile application protection [Online]. Available: https://www.arxan.com/products/application-protection/mobile/, Jul. 2015.

[7] Open source computer vision [Online]. Available: http://opencv.org/, Jul. 2015.

[8] Optical mouse [Online]. Available: https://en.wikipedia.org/wiki/Optical_mouse, Jul. 2015.

[9] Receiver operating characteristic (ROC). Wikipedia [Online]. Available: http://en.wikipedia.org/wiki/Receiver_operating_characteristic, Jul. 2015.

[10] Root and Me [Online]. Available: https://play.google.com/store/apps/details?id=com.iamjake.root&hl=en, Jul. 2015.

[11] Sensor delay [Online]. Available: http://developer.android.com/reference/android/hardware/SensorManager.h tml, Jul. 2015.

[12] Unlock root [Online]. Available: http://www.unlockroot.com/, Jul. 2015.

[13] Vine [Online]. Available: http://vine.co/, Jul. 2015.

[14] Weka [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/, Jul. 2015.

[15] XPrivacy 1.9.5: The ultimate privacy manager [Online]. Available: http://forum.xda-developers.com/showthread.php?t=2320783, Jul. 2015.

[16] YouTube [Online]. Available: http://www.youtube.com, Jul. 2015.

[17] Deposit checks easily and securely with mobile check deposit [Online]. Available: http://promo.bankofamerica.com/mobile-check-deposit/, Jul. 2015.

[18] A. Agarwal and B. Triggs, “Recovering 3D human pose from monocular images,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 1, pp. 44–58, Jan. 2006.

[19] G. Al-Naymat, S. Chawla, and J. Taheri, “Sparsedtw: A novel approach to speed up dynamic time warping,” in *Proc. 8th Australasian Data Mining Conf.*, 2009, vol. 101, pp. 117–127.

[20] A. Ali, F. Deravi, and S. Hoque, “Liveness detection using gaze collinearity,” in *Proc. Emerging Security Technol.*, 2012, pp. 62–65.

[21] Y. Amit and D. Geman, “Shape quantization and recognition with randomized trees,” *Neural Comput.*, vol. 9, no. 7, pp. 1545–1588, Oct. 1997.

[22] A. Anjos and S. Marcel, “Counter-measures to photo attacks in face recognition: A public database and a baseline,” in *Proc. Biometrics*, 2011, pp. 1–7.

[23] I. Bente, G. Dreo, B. Hellmann, S. Heuser, J. Vieweg, J. von Helden, and J. Westhuis, “Towards permission-based attestation for the android platform,” in *Proc. Trust Trustworthy Comput.*, 2011, pp. 108–115.

[24] A. Bissacco, M.-H. Yang, and S. Soatto, “Fast human pose estimation using appearance and motion via multi-dimensional boosting regression,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog*, 2007, pp. 1–8.

[25] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, pp. 5–32, 2001.

[26] G. Chetty, “Biometric liveness detection based on cross modal fusion,” in *Proc. 12th Int. Conf. Inf. Fusion*, Jul. 2009, pp. 2255–2262.

[27] G. Chetty and M. Wagner, “Multi-level liveness verification for face-voice biometric authentication,” in *Proc. Biometric Symp.*, 2006, pp. 1–6.

[28] D. Clifford and G. Stone, “Variable penalty dynamic time warping code for aligning mass spectrometry chromatograms in r,” *J. Statist. Softw.*, vol. 47, no. 8, pp. 1–17, 2012.

[29] I. Coope, “Circle fitting by linear and nonlinear least squares,” *J. Optim. Theory Appl.*, vol. 76, pp. 381–388, 1993.

[30] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, “Monoslam: Real-time single camera slam,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[31] E. De Castro and C. Morandi, “Registration of translated and rotated images using finite fourier transforms,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 9, no. 5, pp. 700–703, May 1987.

[32] J. B. J. Fourier and A. Freeman, *The Analytical Theory of Heat*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[33] G. Fowler. (2010, Sep.). App Watch: Paypal lets you cash checks on your phone [Online]. Available: http://blogs.wsj.com/digits/2010/09/30/app-watch-paypal-lets-you-cash-checks-on-your-phone/

[34] R. Frischholz and U. Dieckmann, “Bioid: A multimodal biometric identification system,” *IEEE Comput.*, vol. 33, no. 2, pp. 64–68, Feb. 2000.

[35] S. I. Gallant, “Perceptron-based learning algorithms,” *IEEE Trans. Neural Netw.*, vol. 1, no. 2, pp. 179–191, Jun. 1990.

[36] J. Hartigan, *Clustering Algorithms*. New York, NY, USA: Wiley, 1975.

[37] B. F. Hildebrand, *Introduction to Numerical Analysis*, 2nd ed. New York, NY, USA: Dover, 1987.

[38] G. Hong, A. Rahmati, Y. Wang, and L. Zhong, "Sensecoding: Accelerometer-assisted motion estimation for efficient video encoding," in *Proc. 16th ACM Int. Conf. Multimedia*, 2008, pp. 749–752.

[39] K. S. Huang and M. M. Trivedi, "Robust real-time detection, tracking, and pose estimation of faces in video streams," in *Proc. 17th Int. Conf. Pattern Recog.*, 2004, vol. 3, pp. 965–968.

[40] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, 1995, pp. 1137–1143.

[41] K. Kollreider, H. Fronthaler, and J. Bigün, "Non-intrusive liveness detection by face images," *Image Vis. Comput.*, vol. 27, no. 3, pp. 233–244, 2009.

[42] F. Moiz, D. Leon-Salas, and Y. Lee, "A wearable motion tracker," in *Proc. 10th Int. Conf. Body Area Netw.*, 2010, pp. 214–219.

[43] G. Mori and J. Malik, "Recovering 3d human body configurations using shape contexts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 7, pp. 1052–1062, Jul. 2006.

[44] E. Murphy-Chutorian and M. M. Trivedi, "Head pose estimation in computer vision: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 4, pp. 607–626, Apr. 2009.

[45] M. Nauman, S. Khan, X. Zhang, and J.-P. Seifert, "Beyond kernel-level integrity measurement: Enabling remote attestation for the android platform," in *Proc. Trust Trustworthy Comput.*, 2010, pp. 1–15.

[46] G. Pan, L. Sun, Z. Wu, and S. Lao, "Eyeblink-based anti-spoofing in face recognition from a generic webcamera," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2007, pp. 1–8.

[47] G.-T. Park, H. Wang, and Y.-S. Moon, "Liveness detection method and apparatus of video image," US Patent 20080192980 A1, Aug. 2007.

[48] M. Rahman, U. Topkara, and B. Carbunar, "Seeing is not believing: Visual verifications through liveness analysis using mobile devices," in *Proc. 29th Annu. Comput. Security Appl. Conf.*, 2013, pp. 239–248.

[49] H. Rehbinder and B. K. Ghosh, "Pose estimation using line-based dynamic vision and inertial sensors," *IEEE Trans. Automatic Control*, vol. 48, no. 2, pp. 186–199, Feb. 2003.

[50] J. Rodgers, D. Anguelov, H.-C. Pang, and D. Koller, "Object pose detection in range scan data," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recog.*, 2006, vol. 2, pp. 2445–2452.

[51] G. Rogez, C. Orrite-Uruñuela, and J. Martínez-del Rincón, "A spatio-temporal 2d-models framework for human pose recovery in monocular sequences," *Pattern Recognit.*, vol. 41, no. 9, pp. 2926–2944, 2008.

[52] G. Rogez, J. Rihan, S. Ramalingam, C. Orrite, and P. H. Torr, "Randomized trees for human pose detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2008, pp. 1–8.

[53] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, no. 5, pp. 561–580, 2007.

[54] A. Shabtai, Y. Fledel, and Y. Elovici, "Securing android-powered mobile devices using selinux," *IEEE Security Privacy*, vol. 8, no. 3, pp. 36–44, May/Jun. 2010.

[55] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, 2003, pp. 750–757.

[56] J. Six, *Application Security for the Android Platform: Processes, Permissions, and Other Safeguards*. Sebastopol, CA, USA: O'Reilly, 2011.

[57] J. O. Smith, *Spectral Audio Signal Processing*. Washington, CA, USA: W3K Publishing, 2011..

[58] ST. Hill and P. Lewicki, "Finding the right number of clusters in k-means and em clustering: V-fold cross-validation," in *Statistics: Methods and Applications: A Comprehensive Reference for Science, Industry, and Data Mining*. Tulsa, OK, USA: StatSoft, 2010.

[59] N. P. H. Thian and S. Bengio, "Evidences of equal error rate reduction in biometric authentication fusion," Idiap Res. Inst., Martigny, Switzerland, Tech. Rep. EPFL-REPORT-83089, 2004.

[60] O. Ugus, M. Landsmann, D. Gessner, and D. Westhoff, "A smartphone security architecture for app verification and process authentication," in *Proc. 21st Int. Conf. Comput. Commun. Netw.*, 2012, pp. 1–9.

[61] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2003, pp. 294–311.

[62] Y. Wang, Y. Liu, L. Tao, and G. Xu, "Real-time multi-view face detection and pose estimation in video stream," in *Proc. 18th Int. Conf. Pattern Recog.*, 2006, vol. 4, pp. 354–357.

Mahmudur Rahman received the bachelor's degree in CSE from BUET, Bangladesh. He is currently working toward the PhD degree in the School of Computing and Information Sciences, Florida International University. His research interests include security and privacy with applications in online and geosocial networks, wireless networks, distributed computing systems, and mobile applications.

Umut Topkara received the PhD degree in computer science from Purdue University. He is a research engineer at JW Player in New York. Previously, he was a research scientist at the IBM T.J. Watson Research Lab in Yorktown Heights, NY. His research interests include mobile collaboration, security, and machine learning.

Bogdan Carbunar received the PhD degree in computer science from Purdue University. He is an assistant professor in the School of Computing and Information Sciences, Florida International University. Previously, he held various researcher positions within the Applied Research Center at Motorola. His research interests include distributed systems, security, and applied cryptography.