iSpy: Automatic Reconstruction of Typed Input from Compromising Reflections

Rahul Raguram, Andrew M. White, Dibenyendu Goswami,
Fabian Monrose and Jan-Michael Frahm
Department of Computer Science, University of North Carolina at Chapel Hill
Chapel Hill, North Carolina
{rraguram,amw,dgoswami,fabian,jmf}@cs.unc.edu

ABSTRACT

We investigate the implications of the ubiquity of personal mobile devices and reveal new techniques for compromising the privacy of users typing on virtual keyboards. Specifically, we show that so-called compromising reflections (in, for example, a victim's sunglasses) of a device's screen are sufficient to enable *automated* reconstruction, from video, of text typed on a virtual keyboard. Despite our deliberate use of low cost commodity video cameras, we are able to compensate for variables such as arbitrary camera and device positioning and motion through the application of advanced computer vision and machine learning techniques. Using footage captured in realistic environments (e.g., on a bus), we show that we are able to reconstruct fluent translations of recorded data in almost all of the test cases, correcting users' typing mistakes at the same time. We believe these results highlight the importance of adjusting privacy expectations in response to emerging technologies.

Categories and Subject Descriptors: K.4.1 [Computers and Society]: Privacy

General Terms: Human Factors, Security

Keywords: Compromising Emanations; Mobile Devices

Figure 1: Some example threat scenarios that we investigated. Video was recorded in both indoor and outdoor environments, using various consumer video cameras. top: shoulder surfing, bottom: reflection surfing, bottom right: key *pop-out* event.

1. INTRODUCTION

The ability to obtain information without the owner's knowledge or consent is one which has been sought after throughout human history, and which has been used to great effect in arenas as diverse as war, politics, business and personal relationships. Accordingly, methods and techniques for compromising – and protecting – communications and data storage have been studied extensively. However, the ubiquity of powerful personal computing devices has changed how we communicate and store information, providing new possibilities for the surreptitious observation of private messages and data. In particular, mobile phones have become omnipresent in today's society, and are used on a daily basis by millions of us to send text messages and emails, check bank balances, search the internet, and even make purchases. And while some of us may be concerned with – and take steps to prevent – shoulder-surfing and direct observation of the text we input into these devices (see Figure 1), how many of us spare a second glance for the person facing us across the aisle on our morning bus ride?

In this work, we show that automated reconstruction of text typed on a mobile device's virtual keyboard is possible via *compromising reflections*, e.g., those of the phone in the user's sunglasses. Such *compromising reflections* have been exploited in the past for reconstructing text displayed on a screen [2, 3], but these approaches require expensive, high-powered telescopic lenses. Our approach operates on video recorded by inexpensive commodity cameras, such as those found in modern smartphones. The low resolution of these cameras makes visual analysis difficult, even for humans, and severly limits the possibility of directly identifying on-screen text. What makes this threat practical, however, is that most modern touchscreen smartphones make use of a *virtual keyboard*, where users tap keys on-screen. In the absence of tactile feedback, visual confirmation is typically provided to the user via a key *pop-out* effect, as illustrated on the bottom right of Figure 1. Note that although the on-screen text is essentially unreadable, the pop-out event provides a strong

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

visual cue to help identify the letter that was tapped. The approach we take in this paper exploits this effect in order to recover the text typed by the user.

Recent work has also explored the feasibility of reconstructing text typed on a full-sized physical keyboard based on video from compromised web-cams [4]. In that scenario, both the camera and keyboard are stable and in known positions relative to each other. In our case, both the phone and the camera are free to move and can be positioned arbitrarily with respect to each other, adding considerable difficulty to any automated visual analysis. To further complicate matters, the user's fingers often occlude portions of the visual keyboard. We apply a variety of state-of-the-art techniques to overcome these and other challenges.

As we show later, the approach we take offers much promise in two real-world threat models, namely: (1) *direct surveillance*, wherein we assume the adversary is able to direct the camera towards the screen of the mobile device (e.g., over the victim's shoulder) and capture visual cues from the virtual keyboard, and (2) *indirect surveillance*, wherein the adversary takes advantage of indirect views of the virtual keyboard obtained, for example, via "compromising reflections" of the phone in the victim's sunglasses.

In both cases, we assume only inexpensive, commodity video cameras, without any telescopic lenses or additional high-end equipment. In addition, we make no limiting assumptions about the capture setup, the motion of the phone, or the typing style of the user. Thus, the only input we assume is a video sequence, either direct or indirect, of a user typing on a virtual keyboard. We then use a number of computer vision techniques to process the recorded video, identifying, for each frame, potential keys that were pressed. This visual detection, coupled with a language model, enables us to achieve surprisingly accurate sentence retrieval results, even under challenging real-world scenarios.

Our ability to reconstruct text typed on virtual keyboards from compromising reflections underscores the need to continually reevaluate our preconceptions of privacy – or the lack thereof – in modern society. Even cryptography and secure devices are of little use when, across the aisle, someone who appears to be reading email on their phone is in fact surreptitiously recording every character we type.

2. RELATED WORK

By now, it is well understood that electronic, electro-optical and electromechanical devices give off some form of unintentional electromagnetic signals that can inadvertently leak sensitive information. The risks from these so-called "*compromising emanations*" were noted over half a century ago, and led to the introduction of emission-security tests standards to control leakage from digital electronics, particularly when pertaining to classified information [23]. Although the nature of these emissions has changed with evolving technology, side-channel attacks continue to surface [11, 16, 17, 21, 28, 31].

More recently, both visual emanations (e.g., from reflections on curved surfaces of close-by objects such as tea pots) and acoustic emanations (e.g., from key presses on a keyboard or from the sounds made by dot-matrix printers) [1, 32] have been used to undermine the confidentiality of information displayed or entered into commodity devices. More closely related is the work of Backes et al. [2, 3] on "*compromising reflections*" that presents eavesdropping techniques for exploiting optical emanations using telescopic equipment. There, the authors show that an adversary is able to successfully spy from distances as far as 30 meters away, and in certain cases, can even read large text reflected in the eyeball of the victim. In this work, we focus on a related, but different, problem: namely, exploring the feasibility of automatic generation of transcripts from low-resolution, indirect footage captured using inexpensive, and ubiquitous, consumer electronics.

Also germane to this paper is the work of Balzarotti et al. [4] that explores the idea of automatically reproducing text from surveillance video – albeit from a camera mounted directly above a terminal – that captures a user's typing as she inputs data at the keyboard. Similar to Balzarotti et al., we apply the noisy channel model to help recover sequences of words from streams of frames (with guessed labels); both works also employ an n-gram language model. However, the error model employed by Balzarotti et al. only accounts for the deletion of identified characters and the substitution of one character for another. In contrast, our model allows for insertions, deletions, and substitutions, with substitutions weighted according to the distance between the two characters on the keyboard. Moreover, unlike Balzarotti et al., our frame parsing model handles spacing, allowing for the insertion and removal of spaces. An additional challenge in our setting is the need to overcome significant instability in the captured footage, as well as operate at a far lower resolution.

A more distantly related problem is that of extracting captions in broadcast news in order to provide search metadata for digital archives. In these works, the low resolution of characters within the video makes the problem of segmenting characters quite challenging, so much so that video OCR typically does not perform well without significant text enhancement [13]. As we show later, the problem is exacerbated in our context, where we operate on even lower resolutions and must simultaneously deal with instability in the video (e.g., due to both camera and phone motion).

3. OUR APPROACH

Our approach consists of a number of stages, each a difficult problem requiring the application of advanced computer vision and machine learning techniques. At a high level, the approach we take is to first detect and track the phone across the frames of a video sequence. As we do so, stable and distinctive feature points are extracted from the phone area in each frame, and subsequently used to compute stabilizing image transformations. These transformations help eliminate the effects of camera and phone motion. The stabilized video frames are also aligned to a reference image of the phone (obtained, for instance, from the user manual of the device of interest). In this aligned state, pop-out events for each key are then localized to specific known regions of the keyboard, and we train classifiers to recognize when a particular key has been pressed. The result is that for each frame, we output a set of possible key pop-out events. To account for missed and spurious detections, we subsequently use a language model to impose additional constraints, thus refining the output of the computer vision modules. Finally, we evaluate the effectiveness of our approach using established metrics, designed to correlate well with human judgements, for scoring machine translations. Next, we discuss each component of the overall design (given in Figure 2) in turn.

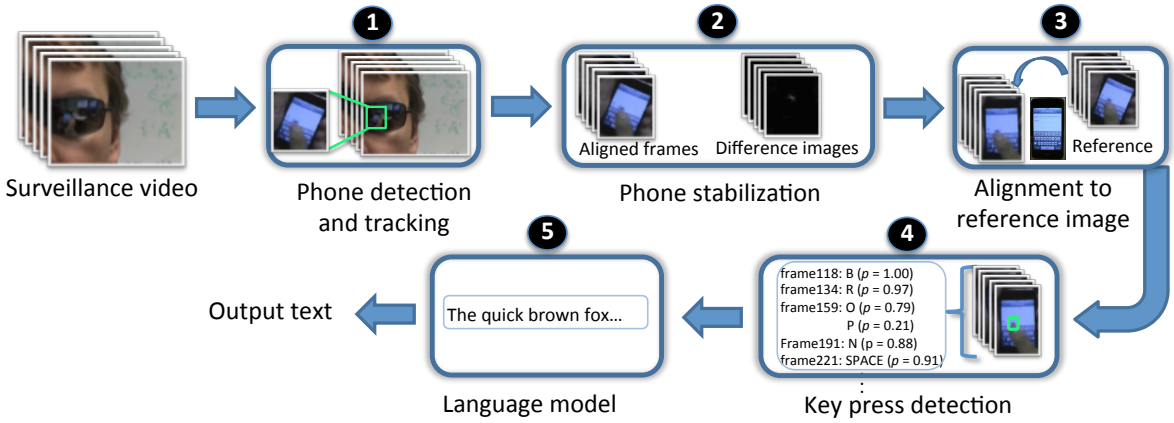


Figure 2: Overview of our approach for typed input reconstruction from video.

3.1 Phone detection and tracking (Stage ❶)

Given a surveillance video, one of the most basic challenges we face is in determining the location of the phone in the video. It is often the case that the phone occupies only a small spatial region of the image, with the remaining portion being unrelated background clutter (e.g., the phone in Figure 3 only occupies 1.8% of the total image area). Indeed, the visual features on the background are invariably "distracting" for subsequent stages of our approach, since they vastly outnumber the visual features on the phone itself. Determining the location of the phone in the video thus enables us to focus specifically on the object of interest, eliminating all irrelevant background information. Hence, our goal in this stage is to compute (for every frame) an approximate spatial region, or bounding box, that contains the phone.

The domains of *object detection* and *object tracking* have received widespread attention in computer vision, due to the ubiquitous nature of these problems [8, 15, 27, 29, 30]. In certain applications, such as frontal-view face detection, modern techniques are capable of providing very accurate results; indeed, most consumer point-and-shoot cameras today come equipped with built-in face detection technology. The approach we take in this paper involves the formulation of the tracking problem as one of binary classification [9, 15, 27]. The intuition is to train binary classifiers to distinguish the appearance of the object being tracked from that of the background. This training is either performed *offline* (using a dedicated training phase prior to running the tracking algorithm), or *online* (where the appearance of the object is learned *during* the tracking process). In the former case, tracking is typically very fast, since the classifiers have been pre-trained beforehand. The latter, while slower, is capable of adapting on-the-fly to changes in appearance of the object. In our setting, the appearance of the object being tracked can vary considerably, based on the model of the phone, the relative orientation between the surveillance cameras and the user, the ambient lighting conditions, occlusions, etc. Hence, to handle this appearance variability, we elect to perform online training, learning the appearance of the phone during the tracking process.

We base our phone tracker on the techniques proposed in [8, 27], which describe an online AdaBoost [7] feature selection algorithm for tracking. At a high level, *boosting* is a classification scheme that works by combining a number of so-called *weak learners* into a more accurate *ensemble* classifier. A weak learner may be thought of as a "rule of thumb" that only has to perform slightly better than chance – for example, in a binary classification problem, the error rate must be less than 50%. These can be as simple as a threshold on a feature value. The intuition is that a combination of these "weak" rules will often be much more accurate than any individual rule. Given a set of training images, where each image is labeled as positive (containing the phone) or negative (not containing the phone), we obtain a training set $\chi^L = \{\langle \mathbf{x}_1, y_1 \rangle, ..., \langle \mathbf{x}_{|\chi^L|}, y_{|\chi^L|} \rangle\}$, where $\mathbf{x}_i$ is an m-dimensional feature representing the i^{th} training image, and $y_i \in \{+1, -1\}$ is the corresponding label. As suggested by Grabner et al. [8], we use three different types of image features, concatenated to form the vectors $\mathbf{x}_i$: Haar-like features [30], orientation histograms [20] and local binary patterns [24].

Initially, each training example is given a uniform weight $p(\mathbf{x}_i) = 1/|\chi^L|$. During the first round of training, we select a weak learner that has the lowest weighted classification error, given χ^L and $p(\mathbf{x})$, and add that learner to our ensemble. Following this, the weights $p(\mathbf{x}_i)$ of the misclassified training examples are increased, while the weights of the correctly classified samples are decreased. At each subsequent training iteration, we select a weak learner h_i that does well on the training examples that were hard for the previous weak learners. The final ensemble classifier is then computed as a linear combination of the selected weak learners, with the weight of each learner being proportional to its accuracy.

While the above discussion describes an offline version of boosting, the online variant assumes that one training example is available (for instance, by drawing an approximate bounding box in the first video frame). This image region becomes the positive training sample, and negative examples are extracted from the surrounding background regions. Given this data, multiple training iterations of the online boosting algorithm are performed as above. In the next frame, the ensemble classifier is evaluated at a number of possible image locations (e.g., in a search window surrounding the object's position in the first frame), with each location being assigned a confidence value. The target window is then shifted to the new location of the maxima,

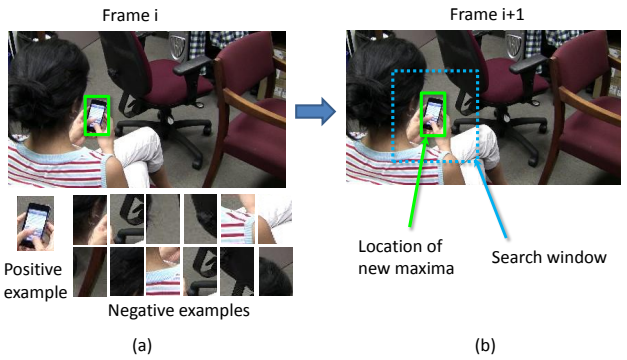


Figure 3: Phone tracking: (a) User-selected bounding box highlighting our positive example for training. (b) In the next frame, the classifier is evaluated within a search window, and the position of the maximum response defines the phone's new location.

and the classifier is updated using new positive and negative training examples so as to become discriminative to variable object/background appearance. The operation of the tracker is illustrated in Figure 3. Note that this is the *only* stage of our overall approach that requires user input – the user simply needs to draw a single rectangular bounding box around the phone area in the first video frame. All subsequent stages of the algorithm are fully automatic. The output of this module is the phone's location in each frame, allowing all further processing steps to focus on the phone.

3.2 Phone stabilization (Stage ❷)

The phone tracking stage provides, for every frame of video, the coordinates of a bounding box that contains the phone. Given these subimages, the next step is to compensate for the effects of phone and camera motion. As discussed earlier, we do not impose any constraints on the motion of either the camera or the user. While this enables us to operate in a wide range of real-world threat scenarios, it also results in a tremendous degree of variability in the appearance of the phone within each detected subwindow. Explicitly compensating for this motion would allow us to effectively reduce one dimension of variability, resulting in a more "stable" set of image frames to work with.

Before presenting the details of our stabilization algorithm, we introduce the notion of a *homography* [10]. In computer vision parlance, a homography is a 2D projective transformation that relates two images of the same planar surface. In our setting, the phone represents a (mostly) rigid, planar object, and we capture multiple images of this plane from potentially different viewpoints. The images that we capture – specifically, the portions of the image that contain the phone – are related to each other via a 2D homography transformation. Note that in the case of reflections, the image of the phone can be distorted due to the curvature of the sunglasses. We do not explicitly model this in the current system, but rather assume that the sunglasses are approximately locally planar. Since the phone occupies only a small area of the sunglasses, this approximation proves to be sufficient in practice.

The 2D homography transformation has a number of important practical applications, one of which is image stabilization. Given two views of a rigid planar object, along with the homography $\mathbf{H}$ between them, the homography can be used to *warp* one of the images in order to align it with the other. In other words, if we were given access to $\mathbf{H}$ with respect to the phone, then we could align the phone in neighboring frames of a video sequence, thereby removing the effects of phone and camera motion. In short, the basic idea is to compute pairwise homographies between phones in video frames, chaining them together to align all phones in the frames to a common reference frame. The problem now reduces to that of automatically determining the transformation $\mathbf{H}$, given two neighboring video frames I_t and I_{t+1}. This is a well studied problem in computer vision, and several algorithms have been proposed to automatically estimate this transformation.

The approach we take involves two key steps. In our **feature extraction and matching** step, we extract stable, repeatable and distinctive *feature points* in the two images, with the intuition being that we would like to identify matching points in the captured images that correspond to the same 3D point on the phone. For this we use the Scale Invariant Feature Transform, or SIFT [22]. Each SIFT feature consists of a 2D image location, scale, orientation vector, and a 128-dimensional *feature descriptor* which represents a histogram of gradient orientations centered around the extracted feature. The main point is that a pair of features in two images that correspond to the same point in 3D space will have similar SIFT descriptors. The SIFT representation is powerful in practice, and its popularity stems from their ability to tolerate a wide range of scale and illumination changes, as well some degree of viewpoint variation. For this task, we use a fast in-house GPU implementation of SIFT extraction and matching[1], capable of processing more than 12 frames per second on a standard graphics card.

For our **robust homography estimation** step, we compute the homography $\mathbf{H}$ from N "true" feature matches between the two images using the normalized Direct Linear Transformation (DLT) algorithm [10]. One limitation here, however, is that doing so requires a minimum of $N = 4$ feature matches between the two images. Additionally, the DLT algorithm is sensitive to outliers, or mismatched features. To combat this problem, we turn to the field of *robust statistics* [12], where the problem of estimating quantities from data that has been corrupted by noise and outliers has been well studied. Perhaps the most popular of these *robust estimators* is Random Sample Consensus (RANSAC). The popularity of RANSAC, particularly in computer vision – where noisy and outlying image measurements are practically unavoidable – is primarily due to its ability to tolerate a high level of data contamination. In our approach, we apply a fast, real-time variant of RANSAC, called ARRSAC (Adaptive Real-Time Random Sample Consensus) [26], which simultaneously estimates the 2D homography, as well as returning the set of "true" feature correspondences. The resulting homography can then be used to align the phone images together, thus nullifying the effects of scene and camera motion.

The two steps outlined above are illustrated in Figure 4. Figures 4(a)-(c) denote the process of SIFT feature extraction, matching, and robust homography estimation, respectively. Notice that the incorrect feature matches present

[1]Available online: `http://cs.unc.edu/~ccwu/siftgpu`

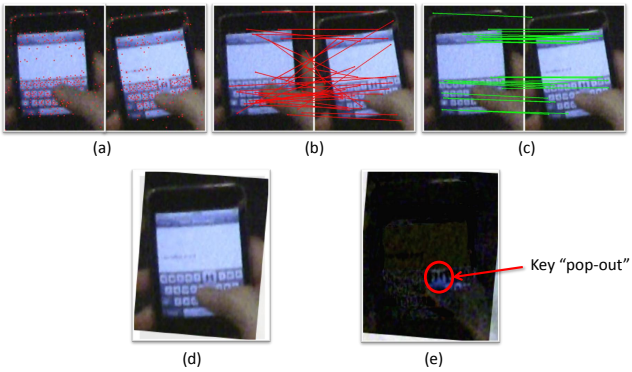


Figure 4: Automatic phone stabilization of two video frames.

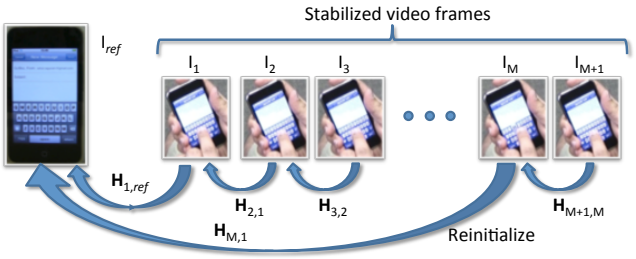


Figure 5: Iterative procedure to align video frames to the reference image (I_{ref}). To prevent long-term drift, the frames are periodically reinitialized with respect to the reference image.

in Figure 4(b) are "cleaned up" by ARRSAC, which selects the set of true correspondences (shown in Figure 4(c)) out of the potential correspondences (Figure 4(b)). These true correspondences are then used to estimate the homography between the two frames. Figure 4(d) shows the two frames aligned with respect to each other, and Figure 4(e) represents the pixel-wise difference between the images after alignment. In the difference image, dark pixels represent areas of low image difference, and lighter pixels represent areas of high difference. Two points are pertinent: first, since the difference image consists mainly of dark pixels, this is an indication that the homography-based alignment has accurately aligned the images to each other; and second, observe that the key pop-out event becomes apparent in the difference image. We will leverage these observations later.

3.3 Alignment to reference image (Stage ❷)

In the previous section, we showed how one could compensate for the effects of scene and camera motion by aligning the video frames using a robust homography estimation procedure. However, while this results in a stabilized video, one other aspect of appearance variation that remains unaccounted for is the relative positioning between the surveillance camera and the user. Note that we do not assume that the camera has a clean, frontal view of the screen of the device; rather, the surveillance camera can be oriented arbitrarily with respect to the user. Given that our ultimate goal is the detection of pressed keys, we now reduce the difficulty of our problem further by aligning the stabilized video to a *reference phone image*. This image can be obtained easily, in a number of ways: for example, by taking a single frontal-view photograph of the phone or using a photo from a reference manual. Alternatively, it would also be possible to automatically infer the keyboard layout by using character frequency analysis.

Given a reference image, the process of aligning the stabilized video to this reference image can be carried out in much the same way as before; that is, by detecting features in the reference image and the video frames, matching them, and computing a robust homography estimate which can then be used to warp all the video frames to the reference. In principle, we actually need to align only a *single* video frame to the reference image, since the frames of the video sequence have already been aligned to each other by pairwise homographies. More specifically, let $\mathbf{H}_{j+1,j}$ be the homography that transforms video frame I_{j+1} to frame I_j. Assuming that I_1 denotes the first video frame, the transformation between frame I_{j+1} and frame I_1 can be computed by chaining together all previous pairwise transformations:

$$\mathbf{H}_{j+1,1} = \prod_{k=1}^{j} \mathbf{H}_{k+1,k}. \quad (1)$$

In theory, given a single transformation $\mathbf{H}_{1,ref}$, that aligns frame I_1 to the reference image I_{ref}, then by extension, one can align the entire video to the reference image. However, one important issue arises in practice: error propagation. Since the transformations are chained together, even a small error in the estimation of homography $\mathbf{H}_{j+1,j}$ propagates to all subsequent transformations. For a reasonably long video sequence, this invariably leads to "drift", where the alignment progressively deteriorates as more frames are aligned to the reference image.

To combat this effect, we instead perform a more careful alignment process, depicted in Figure 5. We begin by aligning frame I_1 to the reference image I_{ref}, via a robust homography $\mathbf{H}_{1,ref}$, estimated using the techniques introduced in Section 3.2. We then align subsequent frames of video by chaining together pairwise homography transformations – the difference being that every M frames ($M = 50$ in our experiments), we re-initialize our transformation with respect to the reference image by recomputing the video-to-reference image homography. We use the newly estimated homography as the base transformation for the next window of M frames.[2] This process of inter-frame homography estimation is much more accurate, as well as far more efficient, than performing alignment to the reference image. The reason is that the change in appearance of the phone between two successive video frames is often very small, while the change in appearance between the phone image in a random video frame (captured under arbitrary pose and lighting conditions) and the reference image is much larger.

3.4 Key press detection (Stage ❸)

Thus far, we have focused primarily on accounting for sources of appearance variability: phone and camera motion and the spatial relationships between the user and the

[2]In particular, we perform a non-linear minimization of the estimation error [10] with respect to the reference image. This is a common trick often used in practice to prevent drift, and also helps prevent catastrophic failure in the event of failed alignments.

surveillance system. The net effect of the operations performed thus far is to convert an arbitrary, free-form video sequence into one that has been aligned to a stable, *known* reference frame. There is a significant advantage to doing so: by aligning the video to a known coordinate frame, we know precisely which regions to inspect in order to find key pop-out events. More specifically, once the video frames have been aligned to the reference image, we can isolate the key pop-out event of each key on the virtual keypad to a *specific spatial location* (derived, for example, by overlaying 2D boxes on the reference image).

Although we have greatly simplified the problem, we are still faced with challenges. For one, because we are operating at a fairly low resolution, coupled with the fact that the appearance of the keys is often "blurred out," one can not readily apply OCR techniques to recover the characters in the isolated frames. Moreover, in several cases the pop-out events are occluded. As an experiment, we applied a popular open source OCR engine (`tesseract-ocr`) [3] to stabilized frames from Stage ❸. The results were an abysmal failure – no characters were correctly recognized.

Another complication is that the 2D boxes constituting the keypad grid are overlapping – in other words, the key pop-out events for neighbouring keys have a non-negligible area of overlap. To address this, we do not make any final decisions at this stage; rather, for each frame, we inspect each key location independently and assign a score to each key, which may be interpreted as the probability of the key having been pressed in that frame. These scores, along with their key labels, are then used in the final stage.

Training a key press classifier. The basic idea we use to identify key press events is to exploit the fact that we have a known, regular grid, and train a binary classifier for each key on the keypad. The classifier for each key focuses on a specific bounding box on the reference keypad, and aims to distinguish between a key pop-out event and the "background". We again make use of AdaBoost classifiers, introduced in Section 3.1, to perform this classification. In addition, since we have explicitly compensated for multiple sources of appearance variation, we can, at this stage, use an offline training procedure in order to have a classifier that is capable of rapid classification when processing each frame of video. Since we are operating on small image patches, and some illumination variation remains, we use dense SIFT descriptors as the features for each patch (i.e., a SIFT descriptor is extracted for each pixel in the patch, and concatenated to form a feature vector).

For each key on the keypad, we train a binary classifier, by providing positive and negative examples of key pop-out events. This data is obtained by running a representative collection of training videos through Stages ❶-❸, subsequently labeling each aligned frame with the key pressed for the frame. For example, to detect the tapping of the letter 'C', we extract a positive training patch from the 2D box corresponding to that letter, and negative patches for all other letters, at their respective locations. Each classifier is then trained offline using the acquired samples.

On detecting keyboard layout mode: While the above discussion focuses primarily on the alphabet keys, the same principles apply for special characters and numbers. On a smartphone, there is usually a special key that allows one to toggle between alphabet and numeric/special character mode. There are a couple of strategies one could adopt to detect keyboard toggle: (a) train a classifier that inspects the entire keyboard area to detect when the keyboard layout has been toggled, and then use the classifiers for the appropriate keys in each layout or (b) at each key pop-out location, run the classifiers for all keys that could potentially pop out at that location, and select the classifier that yields the highest score. In this work, we chose to pursue the latter option, and have used that approach to successfully detect numbers interspersed with alphabet characters.

Testing the classifier. Given a test video, and a pool of trained key press classifiers, we run the test video through Stages ❶-❸. Then, for every frame of video, each classifier inspects its respective image patch and outputs a classification score (the probability of that key having been pressed in the frame). We reject detections that score less than 0.5. Note that each classifier is run independently, and so there could potentially be multiple keys that pass this threshold. For each frame, we store all potential key labels and scores. Once a key pops-out on the keypad, it typically stays in this state for a fixed amount of time (e.g., about 0.25s on the `iPhone`); this fact can be used to parse the detected sequence of keys in order to identify tentative character breaks.

The observant reader would have noticed by now that we have yet to discuss the issue of the "space" bar. On many popular smartphones we examined (e.g., the `iPhone` and `NexusOne`), there is no pop-out event for the space bar. However, it is still possible to obtain a reasonable estimate of the locations of the spaces in typed text, by performing some straightforward post-hoc analysis. Given a sequence of identified key press events, we determine the median time interval t between successive key presses. If we now reinspect our key press detections, we can label the frames lying between two widely separated[4] key presses as potential space events. Additionally, the visual classifier we use to determine the space key event inspects a larger region of the keyboard, with the intuition being that when users press the space, a large portion of the keyboard is visible. This is by no means foolproof: a few spaces may be missed, and false spaces may be inserted when the user pauses between keystrokes. However, coupled with an image-based classifier, this analysis can still provide useful information.

3.5 Parsing and Language Modeling (Stage ❺)

Once we have identified key labels for each frame in a video, along with potential character breaks and spaces, the issue of identifying typed words still remains. We can view this task in terms of the *noisy channel* problem, often encountered in speech recognition: given a sequence of observations (labeled frames), find the most likely sequence of intended words. This process is often referred to as *maximum-likelihood decoding* [14].

The noisy channel problem is often formulated in a Bayesian framework. Let $P(w|o)$ represent the probability of a word sequence w given an observation sequence o. Then the decoding problem is that of finding $\hat{w} = \text{argmax}_w P(w|o)$. Us-

[3] See `http://code.google.com/p/tesseract-ocr/`

[4] A threshold of $1.5t$ appears to be effective in practice.

ing Bayes' rule, this can be reformulated as

$$\hat{w} = \underset{w}{\operatorname{argmax}} \frac{P(o|w)P(w)}{P(o)} = \underset{w}{\operatorname{argmax}}\, P(o|w)P(w)$$

where $P(o|w)$ represents the probability of observing a sequence o given that the sequence w was intended. The prior $P(w)$ represents the probability of observing word sequence w in the language of interest. The denominator can be safely omitted as it does not depend on w.

To solve the noisy channel decoding problem, speech recognition systems have long employed *cascades* of component models, where each model represents one conceptual stage in the task of transcribing speech. For instance, one such cascade might consist of three models: 1) an *acoustic model* which transforms an acoustic signal into component sounds, 2) a *pronunciation model* which converts sequences of sounds to individual words, and 3) a *language model*, which governs the combination of words into phrases and sentences. In this case, $P(w)$ represents the language model. The likelihood $P(o|w)$ can be further decomposed into sub-models, such as the acoustic and pronunciation models, representing intermediate stages. Commonly, these sub-models are assumed to be independent.

We draw on a large body of work on speech recognition cascades that has proven to be very useful in our context. However, in traditional speech recognition systems, the interaction between components often cannot be modeled explicitly, i.e., each step is performed independently. Pereira and Riley [25] proposed an elegant solution to this problem, representing each model and submodel as a *weighted finite-state transducer (WFST)*, thereby allowing for decoding to range over the entire cascade simulataneously. While a full treatment of weighted finite-state transducers is beyond the scope of this paper, we remind the reader that a finite-state transducer is a finite-state machine with both an input and an output tape and thus represents a mapping between sequences from two alphabets; applying weights to each arc then allows for scoring each path through the transducer. A finite-state *acceptor* can be viewed as the special case where the input and output tapes are identical. Multiple finite-state automata can be combined in various ways: for example, a speech recognition cascade can be represented as the *composition* of individual transducers for each stage. The resulting cascade can then be composed with an acceptor representing an input sequence, which transforms the decoding problem into that of finding the shortest path through the resulting weighted finite-state transducer.

In what follows, we apply weighted finite-state transducers to solve the noisy channel decoding problem in a manner similar to that of speech recognition cascades. That is, we utilize a language model and define a number of component models which, when combined, provide a probabilistic mapping from frame label sequences to word sequence. More specifically, we first apply a frame parsing model, $\mathcal{F}$, which maps sequences of frame labels to character strings. We then apply an edit distance model, $\mathcal{E}$, which maps each character string to a (weighted) set of similar character strings and helps account for errors made by the typist, the recognition algorithm, and the frame parser. Next, a dictionary model $\mathcal{D}$ is applied, discarding those paths resulting in invalid words. Finally, the language model $\mathcal{L}$ is applied, accounting for unlikely words and sequences of words in English.

Each component model is represented as a weighted finite-state machine, and the application of each is performed by composition. The resulting cascade WFST is then composed with the input sequence, represented as acceptor $\mathcal{I}$, resulting in a weighted transducer which maps from the input sequence to word sequences. We then search this transducer for the shortest path, which corresponds to the most likely sequence of words given the input sequence. We use the OpenFST library[5] to construct, combine, optimize and search our model cascade.

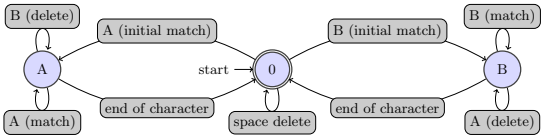


Figure 6: Simplified frame parsing WFST, with only two characters, that maps a sequence of labeled frames to characters.

The frame parsing WFST (Figure 6) allows for character break and space insertion as well as frame, break, and space deletion and is parametrized by weights on each of these actions. Each path starts at the zero state, representing the beginning of the frame label sequence. Upon encountering a frame labeled with a character, a transition is made to a character-dependent state (A or B, in this example) and that character is output. Subsequent frames with the same label are ignored freely, while those with a different character label are dropped with a penalty. In a typical path, once a character break appears in the frame stream, the transition back to the start state is made, from which the string can end or a new character can begin. Thus an input stream 'AA|BB' would be output as 'A|B' in a typical path. Other paths are possible, however, which insert or drop characters breaks and spaces with certain penalties; the same input would also generate, amongst other outputs, 'A' and 'A|A|B', albeit with a lower probability.

Our edit distance is based on the distance between keys on the keyboard and is intended to correct any misclassifications by the recognition algorithm. It can also automatically correct typing mistakes. The distance between two characters is straightforward. If both keys are in the same row, then the distance is the number of keys between them. If the keys are not in the same row, we calculate the distance as though they were in the same row (with rows aligned on the left-hand edge) and apply a multiplicative penalty for each row the two keys are apart. This distance is then normalized to be between zero and one; we take the additive inverse to obtain a probability estimate, which is weighted with a parameter. Similarly, the insertion and deletion costs are represented as probabilities and weighted with parameters, which allow for tuning the effects of the edit distance on the overall cascade. For efficiency, we limit the number of contiguous edits to two.

A simplified view of the edit distance WFST appears in Figure 7. For the edit distance WFST, the most likely path is a loop in the start state, in which case the output string is identical to the input string. However, an edit (substitution, insertion, or deletion) can be made, with a penalty, at any position in a word, resulting in a path which transitions to state 1; a second edit can then be made, transitioning to state 2. From either of these states, a match – or the end of

[5] `www.openfst.org`

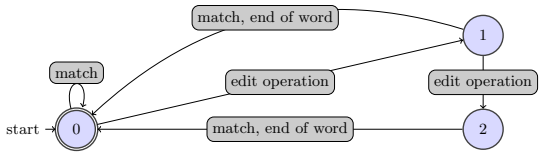


Figure 7: Simplified edit distance WFST mapping sequences of characters to similar sequences.

a word – is required to return to the zero state, which limits to 2 the number of edits which can be made in a row.

The dictionary used is based on the medium-sized word list from the Spell Checker Oriented Word Lists (SCOWL)[6], from which we removed roman numerals and the more obscure abbreviations and proper nouns. Finally, the language model used is a unigram model, i.e., simple word frequencies, trained on the well-known Brown corpus [6].

4. EVALUATION

Recall that our primary goal is to explore the feasibility of exploiting compromising reflections using low cost consumer devices, and to impose very few conditions on the capture environment. Towards this end, we experimented with capture devices ranging from a mid-range consumer grade device (a Canon VIXIA HG21 Camcorder, retailing for about $1000) to low cost, hand-held devices (Kodak PlayTouch and Sanyo VPC-CG20, costing $90 and $140, respectively). The Kodak PlayTouch camera, for instance, has a form factor very similar to a smartphone, thus allowing for unobtrusive capture in real-world settings.

Our capture settings ranged from cameras mounted near the ceiling of an indoor office environment to hand-held capture performed outdoors. For example, we recorded video footage at a bus stop as well as on a moving bus. In the indoor setup, the distance between the user and the camera was approximately 14 feet, while the outdoor capture was done at distances ranging from 4-7 feet (for instance, looking over a person's shoulder while sitting on a bus). At these distances, the pixel dimensions of the phone in the captured video ranged from about 49x75 to 114x149 (*width*x*height*). In addition to low resolution, these datasets present numerous other challenges: unstable video, motion blur, reflections from other objects, etc. See `http://cs.unc.edu/~rraguram/ispy` for some examples.

In total, we collected 18 videos (containing 39 sentences) from ten different users typing on the `iPhone`. Our experimental setup covered a number of practical use-cases designed to elicit a variety of typing styles (and speeds), and includes scenarios where subjects (a) typed short passages of text from *The Art of War* and David Kahn's *The Codebreakers* (b) simply typed whatever came to mind and (c) typed responses to text messages (e.g., '*What time shall we meet?*') sent to the phone. In each case, subjects were instructed to use the phone as they normally would. All subjects routinely use smartphones.

4.1 Evaluating Output Quality

We now turn our attention to how we measure the quality of our reconstructions. The problem we face here is similar to that found in both the automated speech recognition and machine translation (MT) communities, which has addressed many of the challenges associated with scoring the output of such systems. While humans are the target audience for MT systems (and therefore remain the ultimate arbiters of output quality), evaluations using human judges pose several obstacles. For one, evaluations using *experts* can be prohibitively expensive; conversely, hiring non-experts leads to issues with reliability and inconsistency. More pragmatically, progressive development necessitates frequent changes, evaluating the effectiveness of which requires rapid prototyping and testing.

Automated evaluation, on the other hand, allows system designers to quickly test new ideas while providing a consistent basis for comparing multiple approaches. Ideally, such automated evaluations would produce results similar to those of human experts, who typically assess the *adequacy*, or how well the appropriate meaning is conveyed, and *fluency* of a translation on a sentence-by-sentence basis. Similarly, state-of-the-art automated MT evaluation techniques score a *hypothesis* (i.e., the machine translation) by comparing it with one or more *reference* (i.e., expert) translations. The performance of these automated evaluation techniques is judged according to how well the assigned scores correlate with those assigned by experts.

Scoring our inferences. Before proceeding further, we note that automated MT evaluation remains an area of active research, with entire conferences dedicated to the topic. Nevertheless, one widely adopted metric for producing scores at the segment level is the Metric for Evaluation of Translation with Explicit ORdering [19]. METEOR accounts for position-independent matching of words (i.e., to model adequacy) and differences in word order (i.e., to model fluency). More specifically, the METEOR metric is the combination of a weighted f-score and a *fragmentation penalty*. The f-score is defined as the harmonic mean of unigram *precision* p and *recall* r. In this context, precision is the ratio of the number of (non-unique) words which occur in both the reference and the hypothesis to the total number of (non-unique) words in the hypothesis. Recall is the ratio of the number of words present in both hypothesis and reference translation to the number of words in the reference.

Denkowski and Lavie [5] have extensively explored the space of tunable parameters, and have identified different sets of values that correlate well with human evaluations on different tasks; we use the Human-Targeted Edit Rate parameter set with synonym matching disabled. As a guideline for METEOR scores, Lavie [18] suggests that scores of 0.5 and higher indicate *understandable* hypotheses, while scores of 0.7 and higher indicate *good* or *fluent* hypotheses.

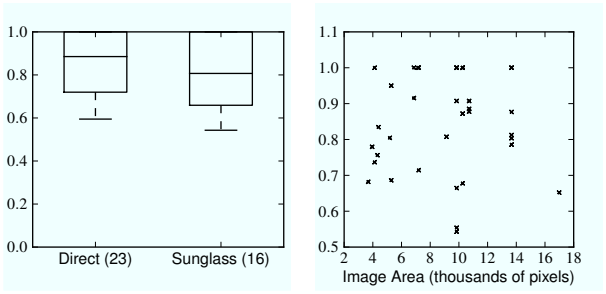
4.2 Results

A boxplot of the METEOR scores for our reconstructions of the sentences typed in our collected videos is provided in Figure 8(a). Notice that in both the direct and indirect cases, more than 35% (8/23 and 6/16, respectively) of our hypotheses achieve *perfect* scores, and none score below the 0.5 threshold representing "understandable" translations. We provide a few examples of our hypothesized transcripts in Table 1, where we also list the input as actually typed by the user and the reference text used for scoring.

[6] `wordlist.sourceforge.net`

	Sentence		Scenario	Score
1	Typed:	to be prepared beforehand for any contingency is the greatest of virtues	Sunglasses	
	Reference:	to be prepared beforehand for any contingency is the greatest of virtues	Canon	0.92
	Hypothesis:	to be prepared beforehand for any contingency _ the greatest of virtues	66x104	
2	Typed:	i freaked out by the *possibilitu* of someone else *reafing* this	Direct, on-bus	
	Reference:	i freaked out by the possibility of someone else reading this	Kodak	0.65
	Hypothesis:	i *created* out by the possibility of *committee* else reading *the*	114x149	
3	Typed:	i can levitate birds	Sunglasses	
	Reference:	i can levitate birds	Sanyo	0.54
	Hypothesis:	i can *hesitate* birds	92x107	

Table 1: Example hypotheses from our reconstruction process. Under 'Scenario' is given the details of the capture scenario for each hypothesis, including camera type and phone image resolution. Note that Sentence 2 was captured on a bus.



(a) METEOR scores (with number of sentences).

(b) Plot of input area size against METEOR scores.

System-Level Analysis. It is also instructive to consider evaluations at the level of entire corpora of documents, i.e., the system level. System-level analysis offers a different perspective and, in particular, smooths the dependency of the scoring on the length of the sentence. For instance, even a single mistake in a short sentence can lead to a relatively low METEOR score, as in Table 1 (Sentence 3). System-level analysis does not depend as strongly on the length of invididual sentences and can therefore alleviate this issue to some extent. The formulae are the same as at the sentence-level, but instead, (*i*) the system-level precision is calculated as the ratio of the sum of the counts of matched words over all sentences to the total number of words over all hypothesis sentences, and (*ii*) the fragmentation penalty is calculated based on the total number of contiguous subsequences and unigram matches over all sentences.

To better judge how well the system-level scores generalize, we also provide confidence intervals based on *bootstrap resampling*, a common statistical technique for estimating the distribution of a quantity, which consists of sampling (with replacement) from the set used to derive a statistic and calculating a *bootstrap statistic* based on the new sample. This process is repeated many times, resulting in an empirical distribution over the statistic of interest. For direct surveillance, we achieve a system-level METEOR score of 0.89, with a bootstrapped 95% confidence interval of $[0.84, 0.93]$. In the indirect surveillance case, we achieve a lower, yet still respectable, system score of 0.77, with a bootstrapped 95% confidence interval of $[0.70, 0.86]$.

Impact of Input Resolution. To gain a deeper understanding of the influence of the various input resolutions (of the phone's screen) on our ability to reconstruct the typed text, we plot the area (in pixels) of each input, against the ultimate METEOR score (Figure 8(b)). The figure shows no correlation, as evidenced by a correlation coefficient (Pearsons's r-value) of 0.07.

Isolated Word-Unit Matching. Lastly, there are a number of scenarios where an attacker might not wish to apply the dictionary matching and language modeling stages: for instance, when the text of interest is a secure password. Unfortunately, METEOR is not well-suited to evaluating accuracy in this scenario since we are interested in the ability to reconstruct isolated *word units*, i.e., sequences of contiguous non-space characters, rather than phrases. For this reason, we give instead precision and recall scores, based on the number of word units which match between what was actually typed and our reconstructed text. The results are for the same sets of sentences as in the evaluation above, but *without* the application of edit distance, dictionary, or language models. In this analysis, we achieve precision and recall, respectively, of 0.75 and 0.78 for direct surveillance and 0.64 and 0.65 for indirect surveillance – in all cases, the accuracy is high enough to recover more than half of any typed passwords. In addition, our single-character precision and recall scores are 94% and 98%, respectively, in the direct case, and 92% and 97% in the indirect case – certainly accurate enough for password guessing, particularly given that we have a reasonable prior distribution over characters to drive password space exploration.

5. SUMMARY & LIMITATIONS

We explore the feasibility of automatically reconstructing typed input in low resolution video, of, e.g., compromising reflections, captured in realistic scenarios. While our results are certainly disconcerting, it is prudent to note that there are some important issues that remain open. Low pixel resolution of the phone image is one of the key problems we encountered. It can be caused by a variety of factors, including camera aperture, wide angle lenses, motion blur, and large standoff distance. All of these make the phone's appearance so blurry that no reliable features can be extracted, and so our phone stabilization (Stage ❷) and alignment (Stage ❸) methods fail in certain cases. We believe this could be addressed by using more sophisticated (and expensive) capture techniques, as in [3], which addresses the allied problem of *capturing* clear images from reflections.

Finally, there are potential defenses against the attacks proposed in this work. One, in the indirect case, is the application of an anti-reflective coating, such as is common on modern eyeglasses, on the reflection surface. Reducing the brightness of the screen would also have a detrimental effect on any reconstruction. Finally, one might disable the

visual key press confirmation mechanism which we leverage in this work. Obviously, our approach is not applicable to situations where there is no visual key press confirmation. Hence, devices that lack this effect – for instance, tablets, or devices that use Swype – are not vulnerable to our attack. How to effectively handle these kinds of devices is an interesting direction to explore. Incorporating temporal information through optical flow could potentially extend our approach to cover these types of input. Lastly, as suggested by Backes et al. [3], one could use secondary reflections in the environment when direct line-of-sight to the target is infeasible.

Nevertheless, the fact that we can achieve such high accuracy underscores the practicality of our attack, and aptly demonstrates the threats posed by emerging technologies.

6. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments. We also thank Pierre Georgel for helpful suggestions in improving this work and Enrique Dunn, Jared Heinly, Alex Keng, Megha Pandey, Anusha Sethuraman, Rohit Shah, Vishal Verma, and Enliang Zheng for assisting in our data collection efforts. This work is supported in part by NSF grant CNS-0852649.

References

[1] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *Proceedings of IEEE Symposium on Security and Privacy*, 2004.

[2] M. Backes, M. Dürmuth, and D. Unruh. Compromising reflections-or-how to read LCD monitors around the corner. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2008.

[3] M. Backes, T. Chen, M. Duermuth, H. Lensch, and M. Welk. Tempest in a teapot: Compromising reflections revisited. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2009.

[4] D. Balzarotti, M. Cova, and G. Vigna. ClearShot: Eavesdropping on keyboard input from video. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2008.

[5] M. Denkowski and A. Lavie. Choosing the right evaluation for machine translation: an examination of annotator and automatic metric performance on human judgment tasks. In *Proceedings of the AMTA*, 2010.

[6] W. N. Francis and H. Kucera. Brown corpus manual. Technical report, Dept. of Linguistics, Brown University, 1979.

[7] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the* 2^{nd} *European Conf. on Computational Learning Theory*, pages 23–37, 1995.

[8] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *British Machine Vision Conference*, volume 1, pages 47–56, 2006.

[9] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. *European Conf. on Computer Vision*, pages 234–247, 2008.

[10] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[11] H. J. Highland. Electromagnetic radiation revisited. *Computer Security*, 5:85–93, June 1986.

[12] P. J. Huber. *Robust Statistics*. John Wiley & Sons, 1981.

[13] K. Jung, K. I. Kim, and A. K. Jain. Text information extraction in images and video: a survey. *Pattern Recognition*, 37(5):977 – 997, 2004.

[14] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2008.

[15] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. *Int. Conference on Pattern Recognition*, 2010.

[16] M. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *Privacy Enhancing Technologies*, 2004.

[17] M. G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[18] A. Lavie. Evaluating the output of machine translation systems. AMTA Tutorial, 2010.

[19] A. Lavie and M. J. Denkowski. The METEOR metric for automatic evaluation of machine translation. *Machine Translation*, 23:105–115, September 2009.

[20] K. Levi and Y. Weiss. Learning object detection from a small number of examples: the importance of good features. In *Computer Vision and Pattern Recognition*, 2004.

[21] J. Loughry and D. A. Umphress. Information leakage from optical emanations. *ACM TISSEC*, 5, August 2002.

[22] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2), 2004.

[23] NSA. TEMPEST: A signal problem. *Cryptologic Spectrum*, 2(3), 1972.

[24] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Machine Intelligence*, 24:971–987, July 2002.

[25] F. C. N. Pereira and M. Riley. Speech recognition by composition of weighted finite automata. *The Computing Research Repository*, cmp-lg/9603001, 1996.

[26] R. Raguram, J.-M. Frahm, and M. Pollefeys. A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus. In *European Conference on Computer Vision*, pages II: 500–513, 2008.

[27] S. Stalder, H. Grabner, and L. V. Gool. Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition. *Workshop on On-line Learning for Computer Vision*, pages 1409–1416, Sept. 2009.

[28] W. van Eck. Electromagnetic radiation from video display units: an eavesdropping risk? *Computer Security*, 4: 269–286, December 1985.

[29] P. A. Viola and M. J. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*, 2001.

[30] P. A. Viola and M. J. Jones. Robust real-time face detection. *Int. Journal of Computer Vision*, 57(2), 2004.

[31] M. Vuagnoux and S. Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *Proceedings of the* 18^{th} *USENIX Security Symposium*, 2009.

[32] L. Zhuang, F. Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM TISSEC*, 13, November 2009.