ClearShot: Eavesdropping on Keyboard Input from Video

Davide Balzarotti, Marco Cova, and Giovanni Vigna

Department of Computer Science,
University of California Santa Barbara
Santa Barbara, CA, USA
{balzarot,marco,vigna}@cs.ucsb.edu

Abstract

Eavesdropping on electronic communication is usually prevented by using cryptography-based mechanisms. However, these mechanisms do not prevent one from obtaining private information through side channels, such as the electromagnetic emissions of monitors or the sound produced by keyboards. While extracting the same information by watching somebody typing on a keyboard might seem to be an easy task, it becomes extremely challenging if it has to be automated. However, an automated tool is needed in the case of long-lasting surveillance procedures or long user activity, as a human being is able to reconstruct only a few characters per minute. This paper presents a novel approach to automatically recovering the text being typed on a keyboard, based solely on a video of the user typing. As part of the approach, we developed a number of novel techniques for motion tracking, sentence reconstruction, and error correction. The approach has been implemented in a tool, called ClearShot, which has been tested in a number of realistic settings where it was able to reconstruct a substantial part of the typed information.

1 Introduction

MARTIN BISHOP: `What's he doing?`
CARL ARBEGAST: `He's logging on the computer.`
MARTIN BISHOP (watching through a video camera):
`Oh, this is good.`
`He's going to type in his password`
`and we're going to get a clear shot.`

From the movie "Sneakers," 1992, Universal Pictures.

Spying on people has always been an effective way of obtaining information since the beginning of history. With the advent of technology, new ways of spying on somebody's communications have been devised, and, consequently, new counter-measures have been developed as well.

One of the most effective ways to protect electronic communication is the use of cryptography-based mechanisms. However, encrypting the communication does not help in protecting one's physical environment, that is, the room the person is sitting in and sometimes the devices attached to one's computer. For this reason, in the past there has been a corpus of research devoted to exploring how to pry into someone's communication by exploiting the side effects of communication and the devices attached to someone's computer. For example, TEMPEST (or Emission Security) is the term used to refer to the analysis of different types of emissions (electromagnetic and acoustic) that can be used to reconstruct the contents of communications [33, 44].

Even though TEMPEST research has been traditionally focused on analyzing the emissions of cables and monitors, recently research has been focused on a number of different "side sources" of information, such as the light reflected by the walls of a room [22], the timing of pressing keys [39], and even the sound emitted by the keyboard [2].

So far, there has not been any extensive study on using the output of video cameras that record someone typing on a keyboard to automatically derive the information being typed. In the '92 movie "Sneakers," a group of hackers-for-hire uses a telescopic video camera to record an unsuspecting victim while he is logging into his computer [37]. In a following scene, the hackers review the video and enter a lengthy debate about which keys were actually pressed. This scene made us wonder about the feasibility of a tool that would perform this tedious, error-prone task automatically.

An additional motivating factor in pursuing this research was the ubiquitous availability of web cams. In the early '90s, web cams were not as popular as they are now, and therefore the hackers in "Sneakers" had to resort to a powerful telescopic camera in order to observe the typing activity. However, nowadays it would be possible to obtain good quality video simply by exploiting the web cam attached to the victim's computer [13, 36]. Therefore, we developed an analysis tool that operates on the stream of images produced by an off-the-shelf web cam that records the typing activity

of a user.

Note that, differently from the situation represented in "Sneakers," we are not interested in the recovery of a single password (which could probably be easily recovered by a human being). Instead, our goal is to recover all the information entered by the user through the keyboard (e.g., email messages, instant messages, documents, and code). This type of activity would be error-prone and resource-intensive if performed by a human being analyzing manually each of the frames produced by the camera. Our experiments show that, for a human, reconstructing a few sentences requires lengthy hours of slow-motion analysis of the video.

Automatically recognizing the keys being pressed by a user is a hard problem that requires sophisticated motion analysis. Previous work in the computer vision field has produced algorithms that can perform well only when the user types on the keyboard using non-realistic movements (e.g., one finger at a time [9, 47]). In addition, several "enhanced" typing interfaces (e.g., projected keyboards) use an array of non-visual sensors to identify the keys being pressed. Our approach is different because it aims at reconstructing the typed information in a realistic setting and using exclusively the video information captured by an off-the-shelf web cam.

One can see the analysis described hereinafter as a sophisticated form of shoulder-surfing (maybe to bypass monitor privacy filters that would block direct analysis of the monitor display [1]), enabled by the availability of web cams. Note however, that the same analysis could be applied to the video output produced by a surveillance camera or a remote camera that records the victim through a window (as shown in "Sneakers").

The contributions of this paper are the following:

- We developed a novel eavesdropping technique that is able to reliably reconstruct the keyboard's keys pressed by a user by analyzing solely the video stream of the typing activity.
- We developed a number of novel techniques to correct errors in the reconstruction process, drastically improving the reconstruction rate.
- We developed a tool, called ClearShot, that operates on low resolution video and is able to reconstruct a substantial part of the typed information.
- To the best of our knowledge, we are the first to study the problem of recognizing keyboard activity from video in a security setting. We experimented with a number of techniques, showing the advantages and disadvantages of each approach.

The rest of the paper is structured as follows. Section 2 describes in details our assumptions, the threat model, and our approach to solving the problem of key pressing recognition. Then, Section 3 presents our experimental evaluation of the tool. Section 4 presents related work. Finally, Section 5 briefly concludes.

2 Approach

The goal of our work is to automatically reconstruct the text typed by a user starting from the video recording of the typing session. In the following, we refer to the person being recorded as "the user" or "the victim," interchangeably. In addition, we refer to the person recording the victim and analyzing the video as "the attacker."

We assume that the attacker is able to point a video camera at the keyboard of the victim. For example, the attacker might install a surveillance device in the room of the victim, might take control of an existing camera by exploiting a vulnerability in the camera's control software, or might simply point a mobile phone with an integrated camera at the laptop's keyboard when the victim is working in a public space, e.g., a café or an airport terminal. The rationale for monitoring the keyboard instead of pointing the spying device directly at the victim's screen is that there might not be a clear line of sight (e.g., the camera can only be planted on the ceiling above a worker's desk in a cubicle, or the screen may be covered with a privacy filter).

These scenarios considerably limit the assumptions we can make with respect to the environment in which the attack is performed. In particular, we can assume that only the video camera and its position are under our control. This means that the physical characteristics of the victim's hands, his/her typing speed and style, the lighting conditions, and the background features, such as color and texture, cannot be directly changed by us. In our experiments, we tested our tool with multiple users in a typical office environment.

Our analysis is comprised of two main phases (see Figure 1). The first phase analyzes the video recorded by the camera using computer vision techniques. For each frame of the video, the *computer vision analysis* computes the set of keys that were likely pressed, the set of keys that were certainly not pressed, and the position of space characters. Because the results of this phase of the analysis are noisy, a second phase, called the *text analysis*, is required. The goal of this phase is to remove errors using both language and context-sensitive techniques. The result of this phase is the reconstructed text, where each word is represented by a list of possible candidates, ranked by likelihood. In the following sections 2.1 and 2.2, we describe the details of the two phases of the analysis.

2.1 Computer Vision Analysis

Problem definition. *The problem of the computer vision analysis is determining the sequence of pressed keys, given the video recording of the typing session.*

At first sight, this problem appears similar to those tackled by *gesture recognition* research. The goal of gesture recognition is to identify specific human gestures and use them to convey information or to control devices [26]. Gesture recognition has been the focus of considerable research

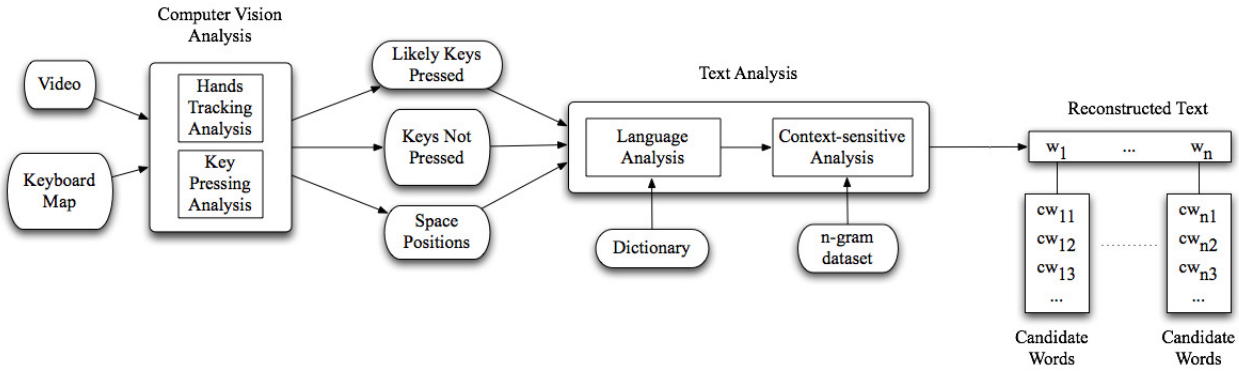


Figure 1. Overview of the analysis steps performed by ClearShot.

in the past years, and it has been successfully applied to automating a number of tasks, such as handwriting recognition [43], sign language interpretation [40], device control [14], and video surveillance [19]. However, the techniques employed in gesture recognition are not immediately applicable to our case, for a number of reasons.

First of all, for practical reasons, gestures are generally limited to a relatively small catalog of recognized movements. For example, the catalog may consist of static movements, such as making a fist and holding it in a certain position, or dynamic movements, such as pinching or waving. In most cases, the allowed movements are fairly constrained. Second, some gesture recognition systems assume a specific configuration of the physical environment, in terms, for example, of the intensity of lights, background texture, and color contrast. We assume that all these parameters are not under our control. Third, some systems take advantage of special input devices, such as tablets [43], data gloves [12, 27, 42], and body suits [30, 45]. Clearly, our approach cannot rely on the use of these devices. Finally, some of the existing techniques require an initial training phase before they can be used effectively. While it is possible to devise scenarios where a learning-based approach could be used to solve our problem (for example, a victim could be lured into typing a known text, which becomes part of a training dataset), we prefer to adopt a completely unsupervised solution.

We divide the computer vision analysis into two sub tasks: *hands tracking analysis* and *key pressing analysis*. The former returns information about the position of the user's hands, while the latter focuses on the keyboard and determines whether or not a key was pressed at a certain point in time. To implement these tasks, we used the Open Computer Vision library [18].

2.1.1 Hands Tracking Analysis

We experimented with several techniques to identify and track the movement of the user's hands on the keyboard. Hereinafter, we first describe a technique based on the extraction of specific features from the video frames considered in isolation, and then we present two techniques based on the analysis of the dynamic properties of the video.

Skin Model Analysis. It has been noted that it is often possible to precisely identify the hands in a video stream by leveraging their chromatic characteristics. For example, Starner and Petland observed that human hands have approximately the same hue and saturation, and vary primarily in their brightness [40]. Similarly, in our experiments we built an *a priori* model of skin color and we used this model to differentiate the hands from the background. Once the hands have been identified, it is easy to track their movement.

This method has the drawback of identifying the shape of the whole hand, while in our analysis we are mostly interested in the fingertips' positions. For this reason, we decided not to use this analysis technique and we focused on other approaches that characterize hand movement more effectively.

Optical Flow Analysis. Optical flow computation is a technique used to estimate the spatial position and movement of objects from patterns of image intensity [5]. This technique analyzes a series of images that have a small time step between them and calculates a vector field across each image plane. The vector field describes the distribution of apparent velocities of sets of points (e.g., brightness patterns) in the image.

We hypothesized that hand movement associated with typing would manifest as characteristic patterns in the optical flow. For example, to press a key, one finger starts moving, acquires speed, slows down and eventually stops when it hits the key, then it retracts from the key, inverting its velocity. Therefore, we computed the optical flow (using the Horn-Schunck algorithm [17]) and looked for matches with a number of typing patterns in the flow.

Unfortunately, our hypothesis was not confirmed. We identified several reasons for this. First, optical flow algorithms are sensitive to noise and changes in the brightness of input images, especially when applied to real image data (as

opposed to synthetic inputs) [5]. This introduces errors in the estimation of movement intensity and direction. Second, typing patterns are more complex than the ones we originally devised: when pressing a key, the entire hand is in motion and, often, several fingers are moving preparing to press subsequent keys. This makes it difficult to isolate typing patterns in a robust way.

Contour Analysis. Image segmentation is the process of partitioning an image into regions [35]. We use simple image segmentation techniques to detect and track the hands. Typically, during a typing session, the only objects moving in the area above and surrounding the keyboard are the user's hands: the keyboard is usually kept stable and no other objects are moved around the typing area. Therefore, we determine the contours of the parts of the hands that are moving by differentiating each frame with respect to the previous one. We approximate these regions with their bounding box, which we can then use as indicators of movement. As expected, moving regions are concentrated around the fingertips and the border of the hands.

2.1.2 Key Pressing Analysis

The techniques described so far focus on the hands of the user. Hereinafter, we describe two techniques that, instead, focus on the keyboard. The first leverages lighting features to determine changes in the status of a key (i.e., from non-pressed to pressed), and the second performs occlusion detection to determine if a key may be pressed or not.

Light-based Analysis. In mechanical keyboards, each key consists of a head, which is connected to the keyboard body by an intermediate plastic part. Keys are typically separated by empty spaces a few millimeters wide. Light diffuses differently on the top portion of a key than on its lateral part: in particular, the face of the key appears lighter under normal lighting conditions. The pressing of a key changes the light diffusion in the area surrounding the key.

We use this property to detect the pressing of a key. By using an algorithm developed by Suzuki and Abe [41], we first detect the contours of the keys on the keyboard. Then, we differentiate the contours on adjacent frames, and, if their difference is above a fixed threshold, we assume that the corresponding key is likely to have been pressed.

Occlusion-based Analysis. The pressing of a key can occur in two ways. First, a finger moves in the area of the key and presses it. Assuming that fingers normally rest in "home" position (the middle row of the keyboard), this is typical of keys on the first two rows of the keyboard, i.e., numbers and the letters from `Q` to `P` in a QWERTY keyboard. Alternatively, a finger is already over the key and simply presses it. This is typical of keys on the home row, i.e., the row with letters from `A` to `L`. In both cases, for a key to be pressed it is necessary that it is at least partially covered by a finger.

We leverage this observation to identify all keys that are certainly not pressed in a certain frame. We use the same key contour detection technique described before to identify the set of keys whose contours are completely visible in a certain time frame. These correspond to keys that are not occluded, and, therefore, that are certainly not pressed.

2.1.3 Analysis Output and Limitations

In our approach, we combine several techniques to take advantage of their relative strengths and minimize their shortcomings. By applying multiple techniques, we can also generate different outputs that aid the subsequent phase of the analysis. In particular, we decided to track the user hands using the contour detection technique. This provides us with the information of where movement is happening (e.g., determined by key pressing). We also use the light-based key pressing detection technique to obtain the list of keys that appear to be pressed. We then combine these two pieces of information, i.e., we retain only those areas of the keyboard where simultaneously hands are moving and keys are pressed (see Figure 2(a)). This analysis provides, for each frame of the video, a list of keys that are likely pressed. Second, we use the occlusion-based technique to obtain a list of keys that are certainly not pressed (see Figure 2(b)). Finally, we apply the light-based key pressing detection technique on the area of the space bar to detect the pressing of the space bar, and, therefore, the ending of a word.

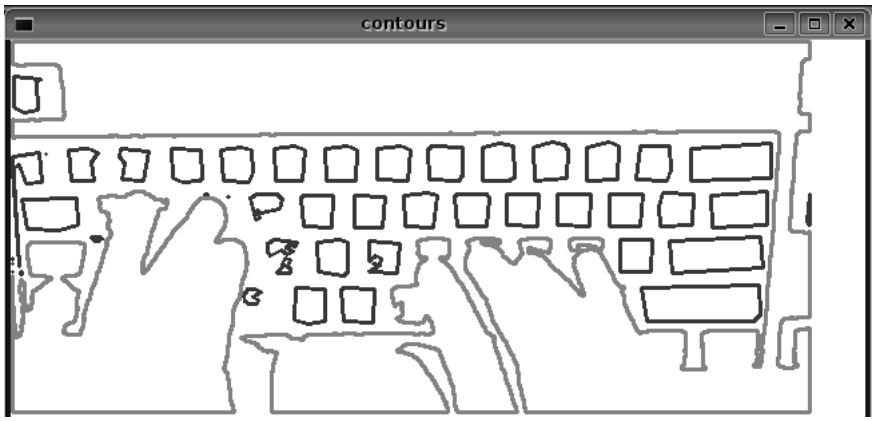
Note that all the techniques we use face many of the challenges that are typical of computer vision. For example, the contour detection algorithm provides only an approximation of the real hand contours. In certain cases, this does not allow us to distinguish whether a pressing involves a key or one of its adjacent keys. In addition, while we leverage occlusion properties in some parts of our analysis, occlusion hinders the detection of key pressings: for example, the light-based pressing detection technique does not perform well when a user's hand projects a shadow that makes the lighting uniform in a region of the keyboard. Finally, classifying the movement of hands on the keyboard is a somewhat fuzzy process: the fact that a finger moves over, or even touches a key, does not necessarily imply that that key has been pressed.

For these reasons, the results produced by this phase of the analysis are noisy. More precisely, the analysis may determine that a key was pressed when in fact it was not (*false pressing*), or fail to detect a key pressing (*missed pressing*). This motivates the need for the subsequent stage of processing: error correction based on analysis of the text.

2.2 Text Analysis

Problem definition. *The goal of the text analysis phase is to suggest a sequence of meaningful words starting from the*

(a) Hand tracking analysis. Rectangles identify regions in movement. Black rectangles are used for movements in the hands regions, grey rectangles for keys, white rectangles for regions where both hand and key movement happens. These rectangles identify likely key pressings.



(b) Key pressing analysis. Using occlusion-based techniques, the analysis determines keys that are not pressed, which are represented by the dark polygons.

Figure 2. Computer vision analysis.

set of candidate letters provided by the video analysis.

This task has many similarities with the traditional problem of spelling correction, initially formulated by Demerau [11] in 1964. The problem can be expressed as follows: given a language L and an unknown word s, find a word $w \in L$ that maximizes $P(w|s)$. This probability can be rewritten applying the Bayes' theorem in $P(s|w)P(w)/P(s)$. The constant denominator can be ignored, thus reducing the problem of spelling correction to the problem of finding the word that provides the best combined probability between the following two factors:

- $P(s|w)$, known as the *error model*. It expresses the probability that a user erroneously typed the string s instead of the correct word w. For example, if the user wanted to type the word `table`, it is reasonably more probable that he/she erroneously wrote the string `tabel` than the string `xavel`.
- $P(w)$, known as the *source* or *language model*. It models the probability that the word w appears in text written in a certain language. In other words, it is used to express the fact that, for example, it is more probable that a user typed `hammer` than `hamper`, even though they are both correctly spelled.

The first example of a error model was proposed by Demerau and Levenshtein [28] and was based on a measure of the distance between two strings, also known as *edit distance*. The edit distance consists of the minimum number of single character insertions, substitutions, deletions, and transpositions that are required to derive one word from the other. For example, `hello` has edit distance 1 from `helo` (one insert operation) and 2 from `elol` (one insert and one transpose operations).

This first error model has been subsequently refined and improved, for example by associating probabilities with individual edit operations, applying different operations depending on the surrounding letters, and extending the edit operations to sequences of multiple characters [8, 10, 32].

For a more comprehensive review of spelling correction techniques, please refer to Jurafsky and Martin's book [20] or Kukich's survey [25].

An alternative approach to building the error model consists of computing the distance on the phonetic representation of the words instead of comparing their written form. For example, both Aspell [3] and Vim's on-the-fly spell checker [34] adopt this technique to correct mistakes that result from the fact that sometimes users know how a certain word sounds but they do not know how it is spelled. Even though these approaches focus on detecting errors based on spelling ignorance, they can sometimes succeed in correcting very noisy input, such as `dtecttioorn` to `detection`, despite the high distance between the two strings.

Finally, the field of context-sensitive spelling correction [15, 31] focuses on the task of analyzing the surrounding context of a word in order to fix errors that result in valid words, such as the use of `than` instead of `then`, or the use of `peace` instead of `piece`.

The combination of these approaches can be very effective in detecting and automatically correcting both typographical errors, homophone confusion, and spelling mistakes. However, all these general-purpose techniques perform very poorly when applied to our text reconstruction problem, because they all rely on a set of assumptions that do not hold in our scenario. In particular, in our case, most of the errors come from inaccuracies in the video analysis phase, and not from user's typing mistakes. Our main source of errors is the movement of the user's hands on top of the keyboard and therefore errors tend to be more random and more difficult to predict. Moreover, while statistically most of the misspelled words contain only one mistake [11], in our text reconstruction we observed a consistently high level of noise, resulting in strings that are sometimes very far from the correct word.

For example, consider the sequence of letters `viaoeryih` extracted by the computer vision analysis from a video when the user typed the word `victory`. The presence of the vocals `a` and `e` in the middle of the word confuses sound-based approaches and the edit distance of 5 is too high to be recovered by any traditional technique.

Even though the computer vision analysis returns noisy results, we have two additional information sources that we can leverage in our text analysis: the association of characters with frames in the video and the set of keys that we can safely assume that have not been pressed.

In the following sections, we provide the details of our text analysis. More precisely in Section 2.2.1 we describe how we develop the *error model* and the *language model* that are necessary to determine (and maximize) $P(w|s)$. Then, in Section 2.2.2 we show how the word context can be leveraged to identify the set of most likely candidates for each word in a sentence.

2.2.1 Language Analysis

The goal of the language analysis is to determine the probability of a certain word, given the set of keys typed by a user, as identified by the computer vision analysis.

Error model. The first step of our language analysis consists of defining an error model that takes into account all the information that we collected during the analysis of the video. In particular, we have two different inputs from the previous phase: a vector that contains, for each frame, the list of keys that the video analysis identified as likely candidates to have been pressed by the user (hereinafter called the *key list*) and a vector that associates to each frame a list of keys that our analysis identified to be untouched (referred as the *exclusion list*).

We first analyze the key list and we identify the keys that appear in consecutive frames, which we call a *key grouping*. By analyzing these groupings, we identify *character models*, which represent place-holders for the characters in the word we are reconstructing. More precisely, the character models are created according to the following rules:

- If a key grouping does not overlap with any other key groupings, a new character model is created that contains only that key. For example, if for ten consecutive frames the only key in the key list is `D`, then a character model that contains `D` is created.
- In case of a partial overlap between two key groupings, we consider them to be consecutive, and we create two character models, one for each grouping, in the order in which the groupings appear. For example, if the user quickly presses two different keys in a row, say `A` and `S`, the key list analysis will produce a grouping of `A`s and a grouping of `S`s. Therefore, two character models will be created: one that contains `A`, and one that contains `S`.
- In case of a complete overlap between two or more different key groupings, we create a character model that contains both keys, since we cannot be sure which one was actually pressed by the user.
- If between two consecutive key groupings there is a number of empty frames that is greater than a certain threshold, we create an empty character model in that position. This models the fact that a pause in the middle of a word may result from the fact that the video analysis missed one or more pressings of the keyboard's keys.

For example, consider Table 1, which presents the key lists associated with each frame. The analysis identifies four key groupings, namely the `C`s spanning frames 2–7, the `H`s spanning frames 5–11, the `I`s spanning frames 9–10, and the `N`s spanning frames 26–27. According to the rules described above, we create a first character model containing

Frame	1	2	3	4	5	6	7	8	9	10	11	12	13	…	25	26	27	…
Key List		C	C	C	C	C	C									N	N	
					H	H	H	H	H	H	H							
									I	I								

Table 1. Example key lists generated by the video analysis.

the C key, a second character model containing both the H and I keys, an empty character model reflecting the absence of keys from frame 12 to frame 25, and finally a character model with the N key.

Once the character models have been created, it is necessary to assign to the keys in each model a score that reflects how confident we are that that particular key was actually pressed by the user and it is not noise introduced by the computer vision analysis.

In our experiments, we observed that, most of the time, actual key pressings result in key groupings that span many consecutive frames. Differently, the keys that are the result of noise introduced by our analysis tend to have a more spiky behavior, and generate key groupings that seldom span many frames. Therefore, we assign to each key in a character model a score that is proportional to the size of the associated key grouping, that is, the number of consecutive frames in which the key was present in the key list.

We then add to each character model two virtual keys: ε and $*$. We use ε to model an "empty" key, i.e., the fact that the user did not press any key. This is a way to express the fact that it is possible that all the keys contained in a character model are the consequence of noise introduced by the computer vision analysis. The score assigned to ε is inversely proportional to the sum of all the key scores inside that character model. By doing this, we model the fact that if a character model contains one or more "strong" keys (i.e., keys that lasted for many frames) it is unlikely that the user did not press anything for such a long time. In addition, this guarantees that the score of ε is less than the score of any "real" key.

The $*$ key is instead a wildcard that can represent both a single character and a combination of characters. This key is introduced to model the fact that the user may have pressed a key in the time frame associated with the character model, but maybe not the one that we detected analyzing the video. The score assigned to $*$ is proportional to the length of the longest key grouping in the character model. The process used to derive this score guarantees that it will be less than the score of any "real" key.

Once the character models have been determined, we connect each key in a character model with all the keys in the following character model. This can be represented as an acyclic graph, which we call the *word model graph*. In this graph, each path, called a *word template*, contains one and only one key for each character model. Figure 3 shows a sample word model graph that was constructed when the user typed the word `change`.

If we consider a word template as a string composed of the sequence of characters associated with each key of the path, we obtain a regular expression that can be used to match words in the dictionary. In our example, choosing the best candidate key in each character model, we obtain the template `ch*klge`. We say that a *word matches a template* if it matches the template's regular expression. We define the score of a template to be the sum of the scores of each node in the template.

Given a graph G and a word w, we define the score of w in G, denoted $score(w, G)$, the highest score among the templates in G that are matched by the word w. Note that this definition is well-formed, because it satisfies two properties: uniqueness (i.e., there is at most one score value for a word) and completeness (i.e., there is always at least a score value for any word).

Uniqueness is guaranteed by definition, since we only use the maximum of all possible template scores. Completeness is guaranteed by construction of the graph: each character model contains the wildcard key $*$. Therefore, any word model graph contains a template composed of all wildcards, which clearly matches any word in the dictionary. As a consequence, $\forall w, \forall G, score(w, G) > 0$.

Having defined the score of a word, we can now define the error model that we use in our analysis. Recall that the error model of the spelling correction problem is the probability that we observe a word s when the user intended to write the correct word w, that is $P(s|w)$. In our problem, the observation is represented by the graph G, which takes the place of the string s, and therefore the error model is $P(G|w)$. We define this probability as:

$$P(G|w) = \frac{score(w, G)}{\max_{j \in L} score(j, G)}$$

Intuitively, this formula relates the probability that we observe a graph G when the user typed the word w to the score of w in G. Since the denominator is a constant, the problem of finding the best candidate word that was typed by the user given a certain graph corresponds to finding the word that has the best score on the graph.

A straightforward way to calculate the best scoring word would be to just calculate the score of all the words in a dictionary. However, since we are only interested in the words with the highest scores, a more efficient solution is to enumerate the graph's templates starting with the one with the greater score (that can be easily find with a greedy algorithm that chooses the key with the highest score in each character model) and match the corresponding template against

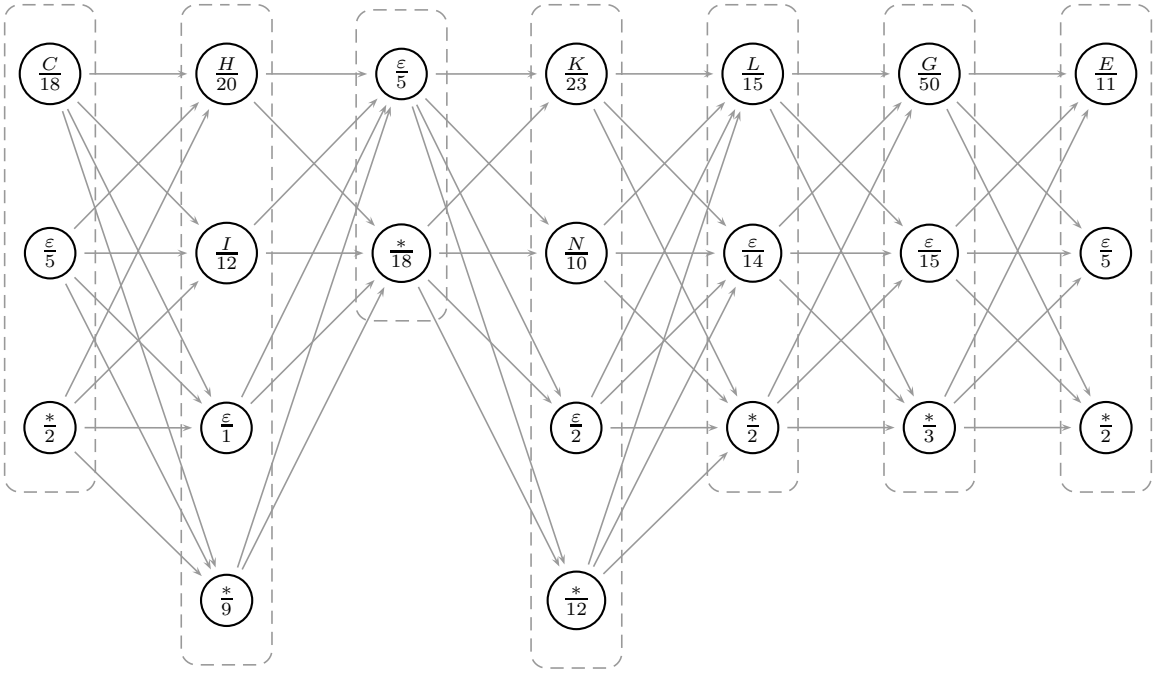


Figure 3. Word model graph obtained when the user types the word `change`. The dashed boxes identify the character models. Nodes represent individual keys in a model. For each key, we report the corresponding letter and score.

the dictionary words. We then move to the second highest path, and so on until a sufficiently large number of candidate words have been extracted.

In our example, the top two templates, i.e., `ch*klge` and `ch*kge`, did not match any existing word in the dictionary. The third one `ch*nge` returned two matches: `change` and `challenge`. The second word is obviously a side effect of having the $*$ node, which can match any possible string (in this case, the string `alle`). The impact of the wildcard keys on the final result can be reduced by integrating into the graph the information provided by the exclusion list. In particular, for each $*$ key in the graph, we retrieve from the exclusion list the set of keyboard's keys that could not have been pressed by the user in the frames associated with the $*$ key. This set is then used to restrict the range of possible values that the wildcard can assume. For example, for the third character model in our example, consulting the exclusion list we see that the key `L` was constantly not occluded on that time frame, and, therefore, it can be removed from the set of possible characters that can be substituted for the $*$ key. By integrating this additional information, `challenge` is not anymore a valid candidate word for the `ch*nge` template.

Language Model. In the previous section, we have determined the probability that a word model graph has been generated because a user was trying to type a specific word. Next, we have to determine the probability that this word has to appear inside the language, that is, the language model.

The language model can be determined in two ways. The first approach consists of training a probability model based on the content of a very large dataset containing a sequence of valid sentences. This has the advantage that, if the attacker knows the general subject of the text, he can choose a dataset focused on that particular subject. For example, if the attacker is spying on a computer programmer, a good solution would be to use a specialized training set that contains many language keywords. In this way, the language model would consider more probable for the user to write the word `class` than the word `classic`.

The second approach consists of extracting the frequency of each word from a dataset. In particular, we use the n-gram dataset provided by Google [7]. The dataset, distributed on six DVDs, contains the frequency of both single words and combinations of words (up to 5-grams) that have been extracted by Google by processing one trillion words from public web pages. This second approach performs better when the topic is not known or when the attacker needs a very large vocabulary (the dataset contains over thirteen millions of unique words, compared with a traditional dictionary that usually contains less than one hundred thousand entries). This dataset also has the advantages of already containing a large number of commonly misspelled words, making our analysis less sensitive to typographical errors introduced by the user.

2.2.2 Context-sensitive Analysis

The language analysis described in the previous section considered each word in isolation and produced a set of different alternatives for each word, which we call *word candidates*, sorted by their score. Even though this is very valuable information, the attacker still has to manually review each set trying to combine the different words to form meaningful sentences.

To some extent, this task can be automated by composing consecutive sets of word candidates in n-gram sequences. A similar approach was proposed by Mays, Damerau, and Mercer [31] in the area of context-sensitive spelling correction. The authors proposed to analyze a sentence using 3-grams in order to detect inappropriate uses of correctly-typed words. However, our problem is more difficult, since we do not even have a sentence but just a sequence of groups of words that we believe are the best candidates to describe the actual words typed by the user.

In our context-sensitive analysis, we take three sets of consecutive words as provided by the language analysis and we combine them together to form all the possible three-word sentences. We then extract the frequency of each sentence from the Google 3-gram dataset.

Consider, for instance, these ordered sets of words obtained by the previous analyses when the user typed "`we talk tomorrow`":

```
{we, mill, will}  {walk, table, talk}
    {tomorrow, tomato, automate}
```

Most of the sentences, like "`we table tomato`," are grammatically incorrect and therefore they are not present in the Google dataset. The word candidates that never appear in any valid 3-gram are discarded as wrong suggestions and the remaining ones are re-ordered to reflect the frequency in which they appear in combination with their neighboring words. The result of the 3-gram analysis is:

```
{we, will}   {talk, walk}  {tomorrow}
```

A first important effect of this analysis is the reduction in the size of the candidate sets. For example, `tomorrow`, which was already the best candidate word in the third set, after the context sensitive analysis is the only word remaining. Another effect is that `talk`, which was the least probable candidate in the second set, has been promoted to the first position. Both these results can greatly simplify the attacker's job of reconstructing the original text.

After the analysis has been applied to a window of three words, the window is moved ahead of one step and the process continues. It may happen that the analysis reaches a point where no valid 3-gram sequence can be constructed. This means that a *misfit set*, which is a set that does not contain the correct word, was involved in the process. However, the misfit set could also be one of those analyzed a few steps before realizing that no valid sequence can be constructed.

To better spot the misfit set, we move the sliding window back four steps and we temporarily switch to a 4-gram analysis. This analysis works exactly as the 3-gram one, just considering sequences of four words instead of three. This is much more precise and much more sensitive to the presence of a misfit set. In fact, while a valid 3-gram can be found by chance also when one of the sets is a misfit, this is much less probable in a 4-gram analysis. During the 4-gram analysis, when we reach the point in which no sequence can be constructed, we mark that last set as being a misfit and we restart the analysis from the word set following the misfit set. Unfortunately, 4-gram analysis is slower, and, therefore, we use this technique only when we suspect the presence of a misfit set. Once the misfit set is identified, the analysis switches back to using the faster 3-gram technique.

3 Evaluation

To evaluate our approach to the reconstruction of typed text from the video of typing activity, we are interested in addressing the following questions:

1. How difficult is it for a human analyst to analyze a video of a typing session and reconstruct its contents?
2. How well does ClearShot perform at the same task?

Given the attack scenarios we devised (e.g., the installation of a hidden camera in an office, or the eavesdropping with a camera-enabled mobile phone), we decided to record the typing activity from directly above the keyboard. We decided to use a simple web cam, but, of course, more advanced cameras could also be used, for example to spy from a distance.

3.1 Setup

The typing sessions were performed on a regular desktop computer. The keyboard used was a model SK-8110 keyboard, in black color, manufactured by Dell. As a recording device, we used an inexpensive, off-the-shelf Unibrain Fire-i web camera, capable of recording up to 15 frames per seconds at 640x480 resolution. We installed the camera over the monitor and configured it to record the keyboard. The keyboard image covered an area of roughly 600x200 pixels. The recordings were taken in an indoor environment, in normal lighting conditions: in particular, light was provided from fluorescent lamps attached to the ceiling and from windows in front of the desktop.

The analysis was performed on a Pentium 4, 3.60GHz machine with 2GB of RAM. We used a dedicated machine with similar processing power to store and access the dataset used in the context-sensitive text analysis. We used the 3-gram and 4-gram datasets from the Google's Web 1T corpus. In particular, we filtered out all entries containing non-alphabetic characters and loaded the remaining entries in a

MySQL database. After the filtering, the database contained 453M 3-gram entries (which used 17GB for the data and 9.7GB for the indexes), and 517M 4-gram entries (which used 19GB for the data and 14GB for the indexes).

3.2 Experiments

We recorded a typing session of two users, Alice and Bob. They were asked to transcribe two paragraphs for a total of 118 words from the beginning of John Fitzgerald Kennedy's inaugural address [21].

We chose the users because of their different typing styles: Alice types using mostly her index and middle fingers. It took her 3m:55s to complete the task. Bob is a touch typist: he types with eight fingers and uses the thumbs to press the space bar. It took him 3m:10s to copy the text. They were asked to type normally and were not given the text in advance to practice. As a consequence, their typing sessions contain "empty" intervals, where they interrupt the transcription to read the source text. Therefore, the duration of the experiment does not reflect the actual typing speed of the candidate. Finally, the two users also introduced some typographical errors: Alice had 5 words misspelled, Bob 6, and both used the backspace key to correct errors throughout the text.

3.2.1 Reconstruction Capability

To assess the difficulty of the eavesdropping task for a human analyst, we asked two people not involved with this project, Analyst 1 and Analyst 2, to manually recover the typed text from the video recording of Bob. They were told that the typed text was taken from a Kennedy's speech, but had no additional information. Also, none of them had previous knowledge of the text. Analyst 1 was able to complete the task in 59m (at a speed of approximately two words per minute), correctly recovering 89% of the actual text; Analyst 2 took 1h55m (at a speed of about one word per minute) and he correctly recovered 96% of the text. These results indicate that the manual analysis is very time-consuming and not completely error-free. Note that both the test subjects found the task very fatiguing and clearly stated that they would not be able to perform it for an extended amount of time, e.g., several hours.

We then ran ClearShot on the two recordings, and the tool produced a sorted list of word candidates. We measured the efficacy of our analysis in reconstructing the typed words by marking the position of the correct word in the list.

Table 2 shows the results of the analysis: for each user, we report the percentage of correct words proposed within the top 1, 5, 10, 25, and 50 choices. The last column shows the percentage of *missed words*, i.e., the cases in which the correct word is not within the first 50 candidates proposed by our analysis tool.

Note that for Alice the context-sensitive analysis had a limited effect on the detection capability of the tool: it increased by 3% the number of correct words proposed as first choice. The improvement was more consistent in Bob's case, where 7% more correct words ended as the first candidate words. The analysis had an even more significant effect in terms of reducing the effort required by a human analyst to review the results: it trimmed the length of the list of proposed words down to an average of 10 for Alice and 12 for Bob (from the 50 proposed by the language analysis), and for about 30% of the words in Alice's case (38% in Bob's case) the length of the list of proposed words is less than or equal to five.

Table 3 shows a sample of the output produced by our tool. It reports the first five candidate words produced by our tool for the first sentence of the original text. We highlighted the correct words using a bold face.

These results show that ClearShot can effectively reconstruct the contents of a typing session from its video in an automatic fashion. This capability can be leveraged by a human analyst in a number of ways. First, ClearShot allows one to quickly understand the general meaning and the topic of the text. For example, by simply looking at the set of candidate words, it is easy to distinguish a political speech from a computer code or a description of a summer vacation. Second, ClearShot's output can be used to automatically check if the typed text contains one or more interesting words. In this case, the context-based analysis can be avoided saving a considerable amount of time and the attacker only has to verify if the interested words are present in the list of the 50 candidates generated by the language-based analysis. Therefore, the values reported in the top 50 column of Table 2 (respectively 82% and 77%) also correspond to the detection capability of the tool in this scenario.

Finally, when the complete text typed by a user needs to be recovered, ClearShot can be used to significantly reduce the required manual effort: in fact, from its output, it is generally easy to pinpoint the parts of the recording where automatic analysis was not precise enough and manual examination of the video is required.

3.2.2 Performance

The performance results of ClearShot are shown in Table 4. Note that the current prototype is not optimized for speed: it is mostly implemented in Python, with only a few modules in C. Porting the tool to C would considerably improve its running time.

We observe that the computer vision phase of the analysis runs in about two to three times the actual recording time. The duration of the language-based analysis is mostly sensitive to the number of false pressings generated by the computer vision phase, as they increase the dimension of the word model graph and, therefore, the number of templates that have to be evaluated. The context-sensitive analysis is mostly influenced by the number of missed words, which force the tool to switch to the more expensive 4-gram analysis. However, it would be easy to parallelize the language

User	Analysis	Top 1	Top 5	Top 10	Top 25	Top 50	Missed
Alice	Language-based	43%	64%	73%	78%	82%	18%
	Context-based	46%	64%	72%	78%	82%	18%
Bob	Language-based	29%	57%	63%	73%	77%	23%
	Context-based	36%	58%	68%	73%	77%	23%

Table 2. Detection results.

Rank	w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11
1	the	**observe**	by	no	**a**	**victory**	**of**	try	**but**	**a**	certain
2	be	observed	may	in	and	victor	off	arts	buy	and	corporation
3	**we**	observer	any	**not**	at	victories	offer	**party**	butt	at	**celebration**
4	are	observers	day	on	as	victors	office	toy	bus	as	cartoon
5	he	observes	my	and	an	victorious	often	toys	bug	an	cartoons

Table 3. Output produced by ClearShot after running the language-based textual analysis when analyzing Alice's video. We only report the top 5 words generated and highlight correct words using a bold face.

analysis: by using k processing units, each analyzing different words, it is possible to speed up the running time of this phase by k times.

3.3 Tuning Parameters

Our tool has a few parameters that can be tuned to take into account differences in light conditions, typing characteristics of users, and keyboard position.

First, color thresholding is used during the contour detection process in order to convert the original image into a binary (black and white) image. Depending on the value chosen for the threshold level, the sensitivity of the analysis to the light can be changed. When this happens, the motion detection sensitivity changes as well: higher values identify more regions as being in motion, which increases the possibility of detecting all key pressings. However, this also increases the amount of noise generated.

Another parameter is the number of empty frames after which the computer vision analysis assumes that a pressing has been missed. This parameter is dependent on both the number of frames per second captured by the camera and the typing speed of the user: faster users take less time to move from one key to the next one, and vice versa.

The typing speed also influences the number of frames in which a key is identified as being pressed. As we have seen, this value is used to determine the score of keys in the character models. It is possible to tune the coefficients used in the key scoring system to compensate for different typing speeds.

Finally, in this initial prototype, the model of the keyboard, which contains the absolute position of each key in the video, is manually determined by examining the first frame of a recording. Clearly, this has to be generated once for each recording session.

4 Related Work

The work described in this paper draws upon or extends previous research in the fields of computer vision and spelling correction. We have described some of the related works in these areas in Section 2, so we will not discuss these fields any further here. Instead, hereinafter we focus on works that address the possibility of leveraging emissions generated from computing devices to eavesdrop on unsuspecting users. Such emissions have often been called *compromising emanations*. The first works in this direction date back to the '60s and have focused on electromagnetic radiations.

According to Highland, the security risks associated with electromagnetic radiation have been known in the military and intelligence communities since 1967 [16], and have received more widespread attention in 1985, when van Eck demonstrated that the screen content of a display could be effectively reconstructed at a distance using cheap and readily available equipment [44]. More recently, Kuhn and Anderson described a number of simple eavesdropping experiments performed with a TEMPEST receiver and a cheap AM radio. They also proposed an active attack consisting of a Trojan that creates a particular video pattern, which, in turn, causes the monitor to emit at a specific radio frequency [24]. In addition, Kuhn describes eavesdropping techniques that can be used to read cathode-ray tube (CRT) and flat panel displays at a distance [22, 23]. Loughry and Umphress discuss the use of LED status indicators on data communication equipment as an eavesdropping device. They also describe an active attack with a Trojan that manipulates the LEDs on a standard keyboard to implement a high-bandwidth covert channel [29]. Finally, Backes et al. have proposed to use the reflections of the screen's optical emanations in common objects to read the screen's contents

	Computer Vision Analysis *mm:ss*	Language-based Analysis *mm:ss*	Context-sensitive Analysis *mm:ss*	Total *mm:ss*	Words per Minute #
Alice	10:08	59:58	44:15	114:21	1.0
Bob	7:33	15:00	48:00	70:33	1.7

Table 4. Performance breakdown of ClearShot.

at a distance [4]. They use a telescope to pick up the reflections in objects commonly located in proximity of the screen, such as plastic bottles and eyeglasses. They use standard algorithms to improve the readability of the recovered image but they do not perform any analysis to automatically identify or reconstruct the eavesdropped contents.

In more recent years, researchers have investigated the use of emanations of different nature than electromagnetic. In particular, several works focused on acoustic emanations caused by a user typing on a regular keyboard. Asonov and Agrawal show that it is possible to differentiate the sound caused by the pressing of different keys, and employ a neural network to recognize the key being pressed [2]. They report about a 50% probability of finding the pressed key in the set of 4 keys proposed by the system in tests consisting of 10 clicks per key. Differently from our study, their experiments do not seem to address realistic typing patterns. Zhuang et al. present an attack that uses the statistical constraints of the English language to reconstruct single words from 10-minute sound recordings without any labeled training data [48]. Similarly to our work, they employ several error correction techniques to improve the performance of their keystroke classifier. They report a word recognition success rate in an unsupervised setting between 47% and 74% and higher in a supervised setting.

The work most similar to ours is the one from Berger et al. [6]. The authors analyze an audio recording of the clicks made by a user typing single words, and compute spatial constraints (equal, adjacent, near, distant) for each pair of keystrokes. They use a dictionary to identify words that satisfy the inferred constraints. They evaluate their technique as a way to reconstruct passwords derived from a dictionary and discuss its use as a building block for a long-text reconstruction system. Their experiments involve only single words. They analyze a set of 27 words, each 7–13 characters long, using 3 keyboards, and report about 40% probability of finding the correct word in the top 10 proposed words, about 60% in the top 25, 70% in the top 50. Our approach focuses directly on long text reconstruction and introduces a number of techniques that aid in achieving this goal.

Reconstructing the typed text from a video recording might seem simpler than performing a sound-based analysis. However, analyzing a video introduces a set of new challenges. In particular, extracting from a mute video the information of when a key pressing occurs is more difficult than extracting the same information from a sound recording. As a consequence, in a video-based analysis, basic information, such as how many characters compose each word, is not immediately available and has to be inferred. We expect that the combination of the two techniques would achieve very high detection rates.

Traffic analysis techniques have been used to eavesdrop on encrypted communications transmitted over a network. For example, Song et al. leverage the keystroke timing data observable in older SSH implementations to recover passwords typed in encrypted sessions [39]. In a test with 4 users typing 5 passwords of 6, 7, and 8 characters taken from a reduced alphabet, they report that the correct password was found in the top 0.1%–62.3% of the strings proposed by their system. Timing is used actively, as opposed to passively, by Shah et al. [38]: they introduce *JitterBugs*, a class of in-line interception mechanisms that covertly transmit data by perturbing the timing of input events likely to affect externally observable network traffic. Finally, Wright et al. analyze Voice over IP (VoIP) communications. They observe that different languages are encoded at different bit rates by Variable Bit Rate encoders and that packet sizes can be used as a predictor of the bit rate used to encode the corresponding frame. They use this information to identify the language spoken in encrypted VoIP traffic [46].

5 Conclusions

In this paper, we presented a novel approach to the automated extraction of information from a web cam video that records a user typing on a keyboard. The approach is based on several novel techniques for movement tracking, sentence reconstruction, and error correction. The approach has been implemented in a tool, called ClearShot, which is able to extract a substantial portion of the text being typed in a video, under certain assumptions.

Even though the automatic recognition of the keys pressed by a person based on video information only is a very complex and challenging task, preventing this attack is easy. The obvious solution is to place some kind of physical shield over the keyboard so that the keys can be seen only by the typist. This technique is sometimes used to protect keypads used to enter PINs at ATMs and POSs. However, this type of protections are not widely used for computer keyboards.

Future work will focus on improving the motion tracking algorithm so that it works reliably in a number of different

settings (i.e., different lighting conditions, different camera angles, and different camera types). In addition, we plan to explore how the context of the information being extracted (e.g., the use of specific keywords) can be leveraged to improve the selection of words among multiple alternatives.

As far as we know, this is the first tool of this kind. We envision that this tool could be of value in the case of long-lasting surveillance operations. In addition, we anticipate that some of the techniques developed to extract text from a typing video could be reused and adapted to other fields, such as computer vision and augmented reality.

Acknowledgments

This research was partially supported by the National Science Foundation, under grants CCR-0238492, CCR-0524853, and CCR-0716095.

References

[1] 3M Notebook Privacy Computer Filter PF15.2W. `http://www.3m.com`, 2007.

[2] D. Asonov and R. Agrawal. Keyboard Acoustic Emanations. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–11, 2004.

[3] K. Atkinson. GNU Aspell 0.50.5 Documentation. `http://aspell.net/0.50-doc/man-html/manual.html`, 2004.

[4] 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.

[5] J. Barron, D. Fleet, and S. Beauchemin. Performance of Optical Flow Techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[6] Y. Berger, A. Yeredor, and A. Wool. Dictionary Attacks Using Keyboard Acoustic Emanations. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 245–254, 2006.

[7] T. Brants and A. Franz. Web 1T 5-gram Version 1. Linguistic Data Consortium, Philadelphia, 2006.

[8] E. Brill and R. Moore. An Improved Error Model for Noisy Channel Spelling Correction. *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293, 2000.

[9] Y. Cetin. Cable Free Keyboard Apparatus Based on Computer Vision. WIPO Patent WO/2002/027457, April 2000.

[10] K. Church and W. Gale. Probability Scoring for Spelling Correction. *Statistics and Computing*, 1(2):93–103, 1991.

[11] F. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[12] S. Fels and G. Hinton. Glove-TalkII: an Adaptive Gesture-to-Formant Interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 456–463, 1995.

[13] S. Fogie. Axis communications 207w network camera web interface vulnerabilities. `http://www.securityfocus.com/bid/25678`, 2007.

[14] W. Freeman, K. Tanaka, J. Ohta, and K. Kyuma. Computer Vision for Computer Games. In *Proceedings of the International Conference on Automatic Face and Gesture Recognition*, pages 100–105, 1996.

[15] A. Golding and D. Roth. A Winnow-Based Approach to Context-Sensitive Spelling Correction. *Machine Learning*, 34(1):107–130, 1999.

[16] H. Highland. Electromagnetic Radiation Revisited. *Computers & Security*, 5(2):85–93, 1986.

[17] B. Horn and B. Schunck. Determining Optical Flow. *Artificial Intelligence*, 17:185–203, 1981.

[18] Intel. OpenCV: Open Source Computer Vision Library. `http://www.intel.com/technology/computing/opencv/`.

[19] Y. Ivanov and A. Bobick. Recognition of Multi-Agent Interaction in Video Surveillance. In *Proceedings of the International Conference on Computer Vision*, pages 169–176, 1999.

[20] D. Jurafsky and J. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. MIT Press, 2000.

[21] J. Kennedy. Inaugural Address, January 1961.

[22] M. Kuhn. Optical Time-Domain Eavesdropping Risks of CRT Displays. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–18, 2002.

[23] M. Kuhn. Electromagnetic Eavesdropping Risks of Flat-Panel Displays. In *Proceedings of the International Workshop on Privacy Enhancing Technologies*, pages 88–107, 2004.

[24] M. Kuhn and R. Anderson. Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations. In *Proceedings of the International Workshop on Information Hiding*, pages 124–142, 1998.

[25] K. Kukich. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[26] J. LaViola. A Survey of Hand Posture and Gesture Recognition Techniques and Technology. Technical Report CS-99-11, Brown University, 1999.

[27] C. Lee and Y. Xu. Online, Interactive Learning of Gestures for Human/Robot Interfaces. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2982–2987, 1996.

[28] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Physics*, 10(8):707–710, 1966.

[29] J. Loughry and D. A. Umphress. Information Leakage from Optical Emanations. *ACM Transactions on Information and System Security*, 5(3):262–289, 2002.

[30] T. Marrin and R. Picard. The 'Conductor's Jacket': A Device for Recording Expressive Musical Gestures. In *Proceedings of the International Computer Music Conference*, 1998.

[31] E. Mays, F. Damerau, and R. Mercer. Context Based Spelling Correction. *International Journal on Information Processing and Management*, 27(5):517–522, 1991.

[32] M. McIlroy. Development of a Spelling List. *IEEE Transactions on Communications*, 30(1):91–99, 1982.

[33] Tempest Fundamentals. NACSIM 5000 NSA-82-89, National Security Agency, February 1982. Classified.

[34] S. Oualline. *Vi IMproved – Vim*. New Riders, 2001.

[35] N. Pal and S. Pal. A Review On Image Segmentation Techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

[36] A. Pastor. Owning big brother (or how to crack into axis ip cameras). www.procheckup.com, 2007.

[37] P. Robinson. Sneakers. Universal Pictures, 1992.

[38] G. Shah, A. Molina, and M. Blaze. Keyboards and Covert Channels. In *Proceedings of the USENIX Security Symposium*, pages 59–75, 2006.

[39] D. Song, D. Wagner, and X. Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *Proceedings of the USENIX Security Symposium*, 2001.

[40] T. Starner and A. Pentland. Real-time American Sign Language Recognition from Video Using Hidden Markov Models. In *Proceedings of the International Symposium on Computer Vision*, pages 265–270, 1995.

[41] S. Suzuki and K. Abe. Topological Structural Analysis of Digitized Binary Images by Border Following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[42] T. Takahashi and F. Kishino. Hand Gesture Coding Based on Experiments Using a Hand Gesture Interface Device. *ACM SIGCHI Bulletin*, 23(2):67–74, 1991.

[43] C. Tappert, C. Suen, and T. Wakahara. The State of the Art in On-Line Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[44] W. van Eck. Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? *Computers & Security*, 4:269–286, 1985.

[45] A. Wexelblat. A Feature-Based Approach to Continuous-Gesture Analysis. Master's thesis, Massachusetts Institute of Technology, 1994.

[46] C. Wright, L. Ballard, F. Monrose, and G. Masson. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob? In *Proceedings of the USENIX Security Symposium*, 2007.

[47] Y. Wu, Y. Shan, Z. Zhang, and S. Shafer. Visual Panel: From an Ordinary Paper to a Wireless and Mobile Input Device. Technical Report MSR-TR-2000-112, Microsoft Research, 2000.

[48] L. Zhuang, F. Zhou, and J. Tygar. Keyboard Acoustic Emanations Revisited. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 373–382, 2005.