In Phishing and Countermeasures, eds. M. Jakobsson and S. Myers. Wiley-Interscience, 2007, pp. 221-240

6.7 CASE STUDY: ACOUSTIC KEYBOARD EMANATIONS

Li Zhuang, Feng Zhou and J. D. Tygar

Emanations produced by electronic devices have long been a topic of concern in the security and privacy communities [27]. Both electromagnetic and optical emanations have been used as sources for attacks. For example, Kuhn was able to recover the display on a CRT monitor by using indirectly reflected optical emanations [38]. Recently he also successfully attacked LCD monitors [39]. Acoustic emanations are another source of data for attacks. Researchers have shown that acoustic emanations of matrix printers carry substantial information about the printed text [27]. Some researchers suggest it may be

Table 6.5 Fields in the Riddle Demo's Hidden Form and Percentages of Unique Visits (out of 71) in Which the Browser's Autofill Feature Disclosed Personal Information.(No personal information is collected by the demo, so we do not know if the information would be valuable to a phisher; we only record the instances in which the server receives some value for each field. These numbers are for illustration purposes only; no formal user study was conducted and the demo was only advertised to a few colleagues interested in cybersecurity.)

Field	Victims
`First_Name`	9%
`Last_Name`	9%
`Email`	9%
`Address`	9%
`City`	9%
`State`	9%
`Zip`	8%
`Phone_Number`	8%
`Credit_Card_Number`	1%
`Password`	1%

possible to discover CPU operations from acoustic emanations [47]. Most recently, Asonov and Agrawal showed that it is possible to recover text from the acoustic emanations from typing on a keyboard [22].

Most emanations, including acoustic keyboard emanations, are not uniform across different instances, even when the same device model is used; and they are often affected by the environment. Different keyboards of the same model, or the same keyboard typed by different people emit different sounds, making reliable recognition hard [22]. Asonov and Agrawal achieved relatively high recognition rate (approximately 80%) only when they trained neural networks with text-labeled sound samples of the same keyboard typed by the same person. This is in some ways analogous to a known-plaintext attack on a cipher – the cryptanalyst has a sample of plaintext (the keys typed) and the corresponding ciphertext (the recording of acoustic emanations). This labeled training sample requirement suggests a limited attack, because the attacker needs to obtain training samples of significant length. Presumably these could be obtained from video surveillance or network sniffing. However, video surveillance in most cases should render the acoustic attack irrelevant, because even if passwords are masked on the screen, a video shot of the keyboard could directly reveal typed keys. Network sniffing of interactive network logins is becoming less viable since unencrypted login mechanisms are being phased out.

Is a labeled training sample requirement neccessary? The answer is no according to our recent research. This implies keyboard emanation attacks are more serious than previous work suggests. The key insight in our work is that the typed text is often not random. When one types English text, the limited number of English words limits the possible temporal combinations of keys, and English grammar limits the word combinations. One can first cluster (using unsupervised methods) keystrokes into a number of classes based on their sound. Given sufficient (unlabeled) training samples, a *most-likely mapping* between these classes and actual typed characters can be established using the language constraints.

This task is not trivial. Challenges include: 1) How can one model these language constraints in a mathematical way and mechanically apply them? 2) In the first sound-based clustering step, how can one address the problem of multiple keys clustered in the same class and the same key clustered into multiple classes? 3) Can we improve the accuracy of the guesses by the algorithm to match the level achieved with labeled samples?

Our work answers these challenges, using a combination of machine learning and speech recognition techniques. We show how to build a keystroke recognizer that has better recognition rate than labeled sample recognizers in [22]. We use only a sound recording of a user typing.

Our method can be viewed as a machine learning version of classic attacks to simple substitution ciphers. Assuming the ideal case in which a key sounds exactly the same each time it is pressed, each keystroke is easily given a class according to the sound. The class assignment is a permutation of the key labels. This is exactly an instance of a substitution cipher. Early cryptographers developed methods for recovering plaintext, using features of the plaintext language. Our attack follows the same lines as those methods, although the problem is harder because a keystroke sounds differently each time it is pressed, so we need new techniques.

We built a prototype that can bootstrap the recognizer from about 10 minutes of English text typing, using about 30 minutes of computation on a desktop computer with Pentium IV 3.0G CPU and 1G memory. After that it can recognize keystrokes in real time, including random ones such as passwords, with an accuracy rate of about 90%. For English text, the language constraints can be applied resulting in a 90-96% accuracy rate for characters and a 75-90% accuracy rate for words.

We posit that our framework also applies to other types of emanations with inherent statistical constraints, such as power consumption or electromagnetic radiation. One only need adapt the methods of extracting features and modeling constraints. Our work implies that emanation attacks are far more challenging, serious, and realistic than previously realized. Emanation attacks deserve greater attention in the computer security community.

6.7.1 Previous Attacks of Acoustic Emanations

Asonov and Agrawal are the first researchers we are aware of who present a concrete attack exploiting keyboard acoustic emanations [22]. Their attack uses FFT values of the *push peaks* (see Figure 6.27) of keystrokes as features, and trains a classifier using a labeled acoustic recording with 100 clicks of each key. After training, the classifier recognizes keystrokes.

Asonov and Agrawal's work is seminal. They opened a new field. However, there are limitations in their approach.

1. As we discuss in Section 12.1, their attack is for *labeled* acoustic recordings. Given that the attack works well only with the same settings (i.e. the same keyboard, person, recording environment, etc.) as the training recording, the training data are hard to obtain in typical cases. Training on one keyboard and recognizing on another keyboard of the same model yields lower accuracy rates, around 25% [22]. Even if we count all occasions when the correct key is among the top four candidates, the accuracy rate is still only about 50%. Lower recognition rates are also observed when the model is trained by one person and used on another. Asonov and Agrawal admit that this may not be sufficient for eavesdropping.

2. The combination of classification techniques leaves room for improvement. We found superior techniques to FFT as features and neural networks as classifiers. Figure 6.25 shows comparisons. The classifier is trained on the *training set* data and is then used to classify the training set itself and two other data sets. The Figure shows that the recognition rate with cepstrum features is consistently higher than that of FFT. This is true for all data sets and classification methods. The Figure also shows that neural networks perform worse than linear classification on the two test sets. In this experiment, we could only approximate the exact experiment settings of Asonov and Agrawal. But significant performance differences indicate that there are better alternatives to FFT and neural networks combination.

6.7.2 Description of Attack

In this section, we survey our attack without statistical details. Section 6.7.3 presents the attack in full.

We take a recording of a user typing English text on a keyboard, and produce a recognizer that can, with high accuracy, determine subsequent keystrokes from sound recordings if it is typed by the same person, with the same keyboard, under the same recording conditions. These conditions can easily be satisfied by, for example, placing a wireless microphone in the user's work area or by using parabolic microphones. Although we do not know in advance whether a user is typing English text, in practice we can record continuously, try to apply the attack, and see if meaningful text is recovered.

Figure 6.26 presents a high level overview of the attack.

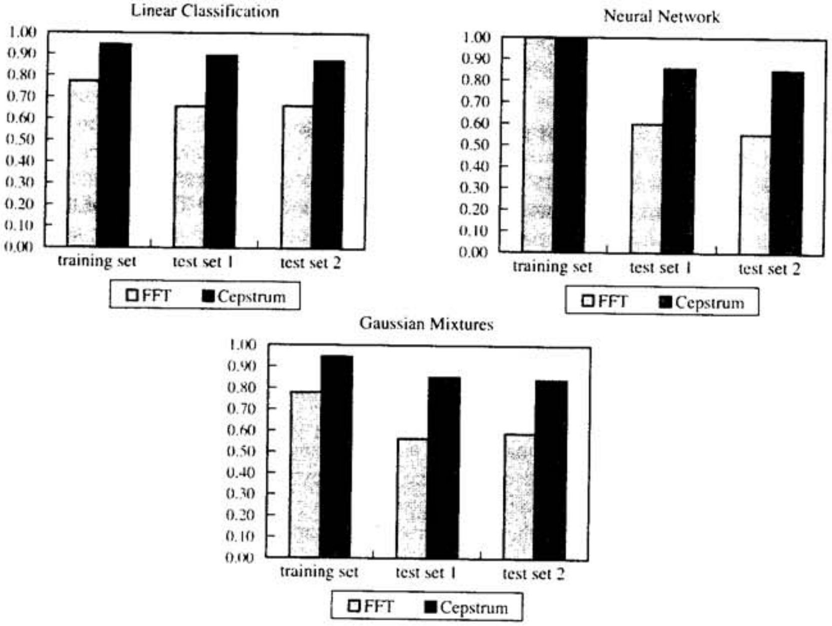
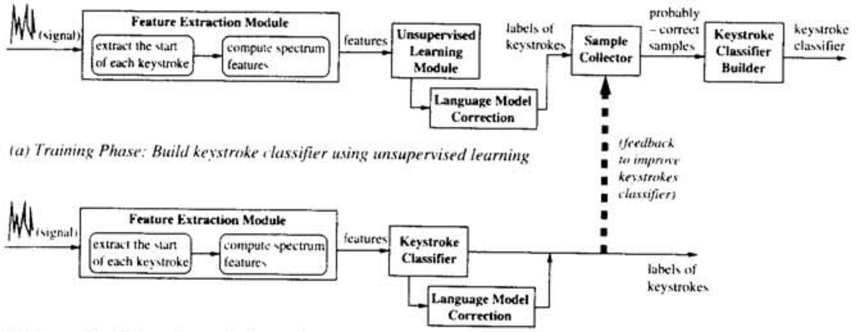


Figure 6.25 Recognition rates using FFT and cepstrum features. The Y axis shows the recognition rate. Three different classification methods are used on the same sets of FFT or cepstrum features.


(signal)
Feature Extraction Module
extract the start of each keystroke
compute spectrum features
features
Unsupervised Learning Module
Language Model Correction
labels of keystrokes
Sample Collector
probably – correct samples
Keystroke Classifier Builder
keystroke classifier
(feedback to improve keystrokes classifier)
(signal)
Feature Extraction Module
extract the start of each keystroke
compute spectrum features
features
Keystroke Classifier
Language Model Correction
labels of keystrokes

(a) Training Phase: Build keystroke classifier using unsupervised learning

(b) Recognition Phase: Recognize keystrokes using the classifier from (a).

Figure 6.26 Overview of the attack.

The first phase (Figure 6.26(a)) trains the recognizer:

1. *Feature extraction.* We use cepstrum features, a technique developed by researchers in voice recognition [49]. As we discuss below in Section 6.7.3, cepstrum features give better results than FFT.

2. *Unsupervised key recognition* using unlabeled training data. We cluster each keystroke into one of K classes, using standard data clustering methods. K is chosen to be slightly larger than the number of keys on the keyboard.

   As discussed in Section 12.1, if these clustering classes correspond exactly to different keys in a one-to-one mapping, we can easily determine the mapping between keys and classes. However, clustering algorithms are imprecise. Keystrokes of the same key are sometimes placed in different classes and conversely keystrokes of different keys can be in the same class. We let the class be a *random variable* conditioned on the actual key typed. A particular key will be in each class with a certain probability. In well clustered data, probabilities of one or a few classes will dominate for each key.

   Once the conditional distributions of the classes are determined, we try to find the most likely sequence of keys given a sequence of classes for each keystroke. Naively, one might think picking the letter with highest probability for each keystroke yields the best estimation and we can declare our job done. But we can do better. We use a Hidden Markov Models (HMM) [36]. HMMs predict a stochastic process with state. They capture the correlation between keys typed in sequence. For example, if the current key can be either "h" or "j" (e.g. because they are physically close on the keyboard) and we know the previous key is "t", then the current key is more likely to be "h" because "th" is more common than "tj". Using these correlations, both the keys and the key-to-class mapping distributions are efficiently estimated using standard HMM algorithms. This step yields accuracy rates of slightly over 60% for characters, which in turn yields accuracy rates of over 20% for words.

3. *Spelling and grammar checking.* We use dictionary-based spelling correction and a simple statistical model of English grammar. These two approaches, spelling and grammar, are combined in a single Hidden Markov Model. This increases the character accuracy rate to over 70%, yielding a word accuracy rate of about 50% or more. At this point, the text is quite readable (see Section 6.7.3.2).

4. *Feedback-based training.* Feedback-based training produces a keystroke classifier that does not require an English spelling and grammar model, enabling random text recognition, including password recognition. We use the previously obtained corrected results as labeled training samples. Note that even our corrected results are not 100% correct. We use heuristics to select words that are more likely to be correct. For examples, a word that is *not* spell-corrected or one that changes only slightly during correction in the last step is more likely to be correct than those that had greater changes. In our experiments, we pick out those words with fewer than 1/4 of characters corrected and use them as labeled samples to train a classifier. The recognition phase (Figure 6.26(b), described below) recognizes the training samples again. This second recognition typically yields a higher keystroke accuracy rate. We use the number of corrections made in the spelling and grammar correction step as a quality indicator. Fewer corrections indicate better results. The same feedback procedure is done repeatedly until no significant improvement is seen. In our experiments, we

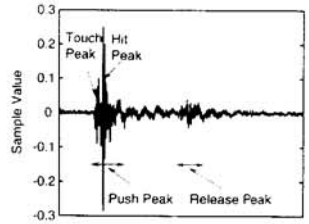


Figure 6.27 The audio signal of a keystroke.

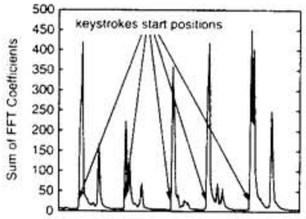


Figure 6.28 Energy levels over the duration of 5 keystrokes.

perform three feedback cycles. Our experiments indicate both linear classification and Gaussian mixtures perform well as classification algorithms [36], and both are better than neural networks as used in [22]. In our experiments, character accuracy rates (without a final spelling and grammar correction step) reach up to 92%.

The second phase, the recognition phase, uses the trained keystroke classifier to recognize new sound recordings. If the text consists of random strings, such as passwords, the result is output directly. For English text, the above spelling and grammar language model is used to further correct the result. To distinguish between two types of input, random or English, we apply the correction and see if reasonable text is produced. In practice, a human attacker can typically determine if text is random. An attacker can also identify occasions when the user types user names and passwords. For example, password entry typically follows a URL for a password protected website. Meaningful text recovered from the recognition phase *during an attack* can also be fedback to the first phase. These new samples along with existing samples can be used together to get an even more accurate keystroke classifier. Our recognition rate improves over time (see Section 6.7.3.3).

Our experiments include data sets recorded in quiet and noisy environments and with four different keyboards (See Table 6.7.4.1 and Table 6.9 in Section 6.7.4).

6.7.3 Technical Details

This Section describes in detail the steps of our attack. Some steps (feature extraction anc supervised classification) are used in both the training phase and the recognition phase.

Keystroke Extraction

Typical users can type up to about 300 characters per minutes. Keystrokes contain a push and a release. Our experiments confirm Asonov and Agrawal's observation that the period from push to release is typically about 100 milliseconds. That is, more than 100 milliseconds is left between consecutive keystrokes, which is large enough for distinguishing the consecutive keystrokes. Figure 6.27 shows the acoustic signal of a push peak and a release peak. We need to detect the start of a keystroke which is essentially the start of the push peak in a keystroke acoustic signal.

We distinguish between keystrokes and silence using energy levels in time windows. In particular, we calculate windowed discrete Fourier transform of the signal and use the sum of all FFT coefficients as energy. We use a threshold to detect the start of keystrokes. Figure 6.28 shows an example.

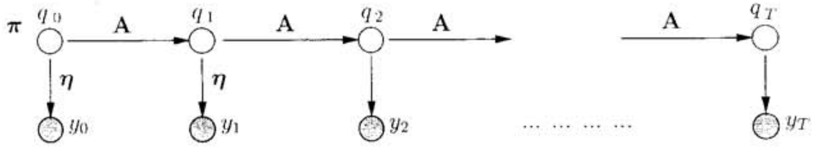


Figure 6.29 The Hidden Markov Model for unsupervised key recognition.

Features: Cepstrum vs. FFT

Given the start of each keystroke (i.e. `wav_position`), features of this keystroke are extracted from the audio signal during the period from `wav_position` to `wav_position` $+ \Delta T$. Two different types of features are compared in our experiments. First we use FFT features with $\Delta T \approx 5$ms, as in [22]. This time period roughly corresponds to the *touch peak* of the keystroke, which is when the finger touches the key. An alternative would be to use the *hit peak*, when the key hits the supporting plate. But that is harder to pinpoint in the signal, so our experiments use the *touch peak*.

As shown in Figure 6.25, the classification results using FFT features are not satisfactory and we could not achieve the levels reported in [22].

Next we use cepstrum features. Cepstrum features are widely used in speech analysis and recognition [49]. Cepstrum features have been empirically verified to be more effective than plain FFT coefficients for voice signals. In particular, we use Mel-Frequency Cepstral Coefficients (MFCCs) [37]. In our experiments, we set the number of channels in the Mel-Scale Filter Bank to 32 and use the first 16 MFCCs computed using 10ms windows, shifting 2.5ms each time. MFCCs of a keystroke are extracted from the period from `wav_position` to `wav_position` $+ \Delta T'$, where $\Delta T' \approx 40$ms which covers the whole push peak. As Figure 6.25 reports, this yields far better results than from FFT features.

Asonov and Agrawal's observation shows that high frequency acoustic data provides limited value. We ignore data over 12KHz. After feature extraction, each keystroke is represented as a vector of features (FFT coefficients or MFCCs). For details of feature extraction, see Appendix B.

6.7.3.1 Unsupervised Single Keystroke Recognition

As discussed above, the unsupervised recognition step recognizes keystrokes using audio recording data only and no training or language data.

The first step is to cluster the feature vectors into K classes. Possible algorithms to do this include K-means and EM on Gaussian mixtures [36]. Our experiments indicate that for tried K (from 40 to 55), values of $K = 50$ yield the best results. We use thirty keys, so $K \geq 30$. A larger K captures more information from the sound samples, but it also makes the system more sensitive to noise. It is interesting to consider future experiments using Dirichlet processes to predict K automatically [36].

The second step is to recover text from these classes. For this we use a Hidden Markov Model (HMM) [36]. HMMs are often used to model finite-state stochastic processes. In a Markov chain, the next state depends only on the current state. Examples of processes that are close to Markov chains include sequences of words in a sentence, weather patterns, etc. For processes modeled with HMM, the true *state* of the system is unknown and thus is represented with *hidden* random variables. What is known are *observations* that depend on the state. These are represented with *known* output variables. One common problem of interest in an HMM is the *inference problem*, where the unknown state variables are inferred from a sequence of observations. This is often solved with the Viterbi algorithm [45]. Another problem is the *parameter estimation problem*, where the parameters of the conditional distribution of the observations are estimated from the sequence of observations. This can be solved with the EM (Expectation Maximization) algorithm [26].

The HMM we use is shown in Figure 6.29[14]. It is represented as a statistical graphical model [36]. Circles represent random variables. Shaded circles (y_i) are observations while unshaded circles (q_i) are unknown state variables we wish to infer. Here q_i is the label of the i-th key in the sequence, and y_i is the class of the keystroke we obtained in the clustering step. The arrows from q_i to q_{i+1} and from q_i to y_i indicate that the latter is conditionally dependent on the former; the value on the arrow is an entry in the probability matrix. So here we have $p(q_{i+1}|q_i) = A_{q_i,q_{i+1}}$, which is the probability of the key q_{i+1} appearing after key q_i. The A matrix is another way of representing plaintext bigram distribution data. The A matrix (called the transition matrix) is determined by the English language and thus is obtained from a large corpus of English text. We also have $p(y_i|q_i) = \eta_{q_i,y_i}$, which is the probability of the key q_i being clustered into class y_i in the previous step. Our observations (the y_i values) are known. The output matrix η is unknown. We wish to infer the q_i values. Note that one set of values for q_i and η are better than another set if the likelihood (joint probability) of the whole set of variables, computed simply by multiplying all conditional probabilities, is larger with the first set than the other. Ideally, we want a set of values that maximize the likelihood, so we are performing a type of Maximum Likelihood Estimation [45].

We use the EM algorithm [26] for parameter estimation. It goes through a number of rounds, alternately improving q_i and η. The output of this step is the η matrix. After that, the Viterbi algorithm [45] is used to infer q_i, i.e. the best sequence of keys.

EM is a randomized algorithm. Good initial values make the chance of getting satisfactory results better. We found initializing the row in η corresponding to the Space key to an informed guess makes the EM results more stable. This is probably because spaces delimit words and strongly affect the distribution of keys before and after the spaces. This task is performed manually. Space keys are easy to distinguish by ear in the recording because of the key's distinctive sound and frequency of use. We mark several dozen space keys, look at the class that the clustering algorithm assigns to each of them, calculate their estimated probabilities for class membership, and put these into η. This approach yields good results for most of the runs. However, it is not necessary. Even without space keys guessing, running EM with different random initial values will eventually yield a good set of parameters. All other keys, including punctuation keys are initialized to random values in η. We believe that initialization of η can be completely automated, and hope to explore this idea in the future work.

[14] One might think that a more generalized Hidden Markov Model, such as one that uses Gaussian mixture emissions [36], would give better results. However, the HMM with Gaussian mixture emission has a much larger number of parameters and thus faces the "overfitting" problem. We found a discrete HMM as presented here gave better results.

6.7.3.2 Error Correction with a Language Model

As we discussed in Section 9.5.3.2, error correction is a crucial step in improving the results. It is used in unsupervised training, supervised training and also recognition of English text.

Simple Probabilistic Spell Correction

Using a spelling checker is one of the easiest ways to exploit knowledge about the language. We ran spell checks using *Aspell* [24] on recognized text and found some improvements. However stock spell checkers are quite limited in the kinds of spelling errors they can handle, e.g. at most two letters wrong in a word. They are designed to cope well with the common errors that human typists make, not the kinds of errors that acoustic emanation classifiers make. It is not surprising that their utility here is quite limited.

Fortunately, there are patterns in the errors that the keystroke classifier makes. For example, it may have difficulty with several keys, often confusing one with another. Suppose we know the correct plaintext. (This is of course not true, but as we iterate the algorithm, we will predict the correct plaintext with increasing accuracy. Below, we address the case of unsupervised step, where we know no plaintext at all.) Under this assumption, we have a simple method to exploit these patterns. We run the keystroke classifier on some training data and record all classification results, including errors. With this, we calculate a matrix E (sometimes called the confusion matrix in the machine learning literature),

$$E_{ij} = \hat{p}(y = i|x = j) = \frac{N_{x=j,y=i}}{N_{x=j}} \tag{6.1}$$

where $\hat{p}(\cdot)$ denotes estimated probability, x is the typed key and y is the recognized key, $N_{x=j,y=i}$ is the number of times $x = j, y = i$ is observed. Columns of E give the estimated conditional probability distribution of y given x.

Assume that letters are independent of each other and the same is true for words. (This is a false assumption because there is much dependence in natural languages, but works well in practice for our experiments.) We compute the conditional probability of the recognized word $\mathbf{Y}$ (the corresponding string returned by the recognizer, not necessarily a correct word) given each dictionary word $\mathbf{X}$.

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{i=1}^{\text{length of }\mathbf{X}} p(\mathbf{Y}_i|\mathbf{X}_i) \approx \prod_i E_{y_i,x_i} \tag{6.2}$$

We compute this probability for each dictionary word, which takes only a fraction of a second. The word list we use is SCOWL [25] which ranks words by complexity. We use words up to level 10 (higher-level words are obscure), giving us 95,997 words in total. By simply selecting the word with the largest posterior probability as our correction result, we correct many errors.

Because of the limited amount of training data, there will be many zeroes in E if Equation (6.1) is used directly, i.e. the matrix is sparse. This is undesirable because the corresponding combination may actually occur in the recognition data. This problem is similar to the zero-occurrence problem in n-gram models [37]. We assign an artificial occurrence count (we use 0.1) to each zero-occurrence event.

In the discussion above we assume the plaintext is known, but we do not even have an approximate idea of the plaintext in the first round of (unsupervised) training. We work

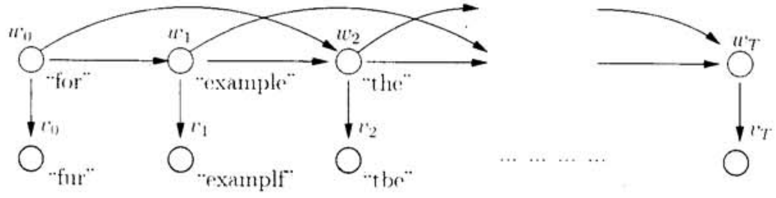


Figure 6.30 Trigram language model with spell correction.

around this by letting $E_{ii} = p_0$ where p_0 is a constant (we use 0.5) and distribute the remaining $1 - p_0$ uniformly over all E_{ij} where $j \neq i$. Obviously this gives suboptimal results, but the feedback mechanism corrects this later.

Adding an n-gram Language Model

The spelling correction scheme above does not take into account relative word frequency or grammar issues: for example, some words are more common than others, and there are rules in forming phrases and sentences. Spelling correction will happily accept "fur example" as a correct spelling because "fur" is a dictionary word, even though the original phrase is probably "for example".

One way to fix this is to use an n-gram language model that models word frequency and relationship between adjacent words probabilistically [37]. Specifically, we combine trigrams with the spelling correction above and model a sentence using the graphical model show in Figure 6.30. The hidden variables w_t are words in the original sentence. The observations v_t are recognized words. $p(v_t|w_t)$ is calculated using Equation (6.2) above. Note this HMM model is a second-order one, because every hidden variable depends on two prior variables. The conditional probability $p(w_t|w_{t-1}, w_{t-2})$ is determined by a trigram model obtained by training on a large corpus of English text.

In this model only the w_i values are unknown. To infer the most likely sentence, we again use the Viterbi algorithm. We use a version of the Viterbi algorithm for second order HMMs, similar to the one in [50]. The complexity of the algorithm is $O(TN^3)$, where T is the length of the sentence and N is the number of possible values for each hidden variable, that is, the number of dictionary words of the appropriate length. To reduce complexity, only the top M candidates from the spelling correction process of each word are considered in the Viterbi algorithm, lowering the cost to $O(TM^3)$. We use $M = 20$ in our experiments. Larger M values provide little improvement.

6.7.3.3 Supervised Training and Recognition Supervised training refers to training processes performed with labeled training data. We apply our feedback-based training processes iteratively, using in each iteration characters "recognized" in previous iterations as training samples to improve the accuracy of the keystroke classifier.

We discuss three different methods we use in our experiments, including the one used in [22]. Like any supervised classification problem, there are two stages:

- Training: input feature vectors and corresponding labels (the key pressed) and output a model to be used in recognition;
- Recognition: input feature vectors and the trained classification model and output the label of each feature vector (keystroke).

Neural Network

The first method is neural networks, also used by Asonov and Agrawal [22]. Specifically, we use probabilistic neural networks, which are arguably the best available for for classification problems [51]. We use Matlab's `newpnn()` function, setting spread radius parameter to 1.4 (this gave the best results in our experiments).

Linear Classification (Discriminant)

The second method is simple linear (discriminant) classification [36]. This method assumes the data to be Gaussian and try to find hyperplanes in the space to divide the classes. We use `classify()` function from Matlab.

Gaussian Mixtures

The third method is more sophisticated than linear classification (although it gave worse result in our experiments). Instead of assuming Gaussian distribution of data, it assumes that each class corresponds to a *mixture* of Gaussian distributions [36]. A mixture is a distribution composed of several sub-distributions. For example, a random variable with distribution of a mixture of two Gaussians could have a probability of 0.6 to being in one Gaussian distribution and 0.4 of being in the other Gaussian distribution. This captures the fact that each key may have several slightly different sounds depending on typing styling, e.g. the direction it is hit.

We also use the EM algorithm to train the Gaussian mixture model. In our experiment, we use mixtures of five Gaussian distributions of diagonal covariance matrices. Mixtures of more Gaussians provide potentially better model accuracy but need more parameters to be trained, requiring more training data and often making EM less stable. We find using five components seems to provide a good tradeoff. Using diagonal covariance matrices reduces the number of parameters. Without this restriction, EM has very little chance of yielding a useful set of parameters.

6.7.4 Experiments

Our experiments evaluate the attacks. In our first experiment, we work with four recordings of various lengths of news articles being typed. We use a Logitech Elite cordless keyboard

Table 6.6 Text recovery rate at each step. With different keyboards.

	recording length	number of words	number of keys
Set 1	12m17s	409	2514
Set 2	26m56s	1000	5476
Set 3	21m49s	753	4188
Set 4	23m54s	732	4300

in use for about two years (manufacturer part number: 867223-0100), a $10 generic PC microphone and a Soundblaster Audigy 2 soundcard. The typist is the same for each recording. The keys typed include "a"-"z", comma, period, Space and Enter. The article is typed entirely in lower case so the Shift key is never used. (We discuss this issue in Section 6.7.4.4.)

Table 6.6 shows the statistics of each test set. Sets 1 and 2 are from quiet environments, while sets 3 and 4 are from noisy environments. Our algorithm for detecting the start of a keystroke sometime fails. We manually corrected the results of the algorithm for sets 1, 2 and 3, requiring ten to twenty minutes of human time per data set. (Sets 1 and 2 needed about 10 corrections; set 3 required about 20 corrections.) For comparison purposes, set 4 (which has about 50 errors in determining the start of keystrokes) is not corrected.

In our second experiment, we recorded keystrokes from three additional models of keyboards. The same keystroke recognition experiments are run on these recordings and results compared. We use identical texts in this experiments on all these keyboards.

6.7.4.1 *English Text Recognition: A Single Keyboard* In our experiments, we use linear classification to train the keystroke classifier. In Table 6.7.4.1, the result after each step is shown in separate rows. First, the unsupervised learning step (Figure 6.26(a)) is run. In this unsupervised step, the HMM model shown in Figure 6.29 is trained using EM algorithm described above[15]. The output from this step is the recovered text from HMM/Viterbi unsupervised learning, and the text after language model correction. These two are denoted as *keystrokes* and *language* respectively in the table. Then the first round of feedback supervised training produces a new classifier. The iterated corrected text from this classifier (and corresponding text corrected by the language model) are shown in the row marked "1st supervised feedback". We perform three rounds of feedback supervised learning. The bold numbers show our final results. The bold numbers in the "language" row are the final recognition rate we achieve for each test set. The bold numbers in the "keystroke" row are the recognition rates of the keystroke classifier, without using the language model. These are the recognition rates for random or non-English text.

Table 6.7 Text recovery rate at each step. All numbers are percentages, where "Un" denotes "unsupervised learning", "1st" denotes "1st supervised feedback", "2nd" denotes "2nd supervised feedback", and "3rd" denotes "3rd supervised feedback".

		Set 1		*Set 2*		*Set 3*		*Set 4*	
		words	chars	words	chars	words	chars	words	chars
Un	keystrokes	34.72	76.17	38.50	79.60	31.61	72.99	23.22	67.67
	language	74.57	87.19	71.30	.87.05	56.57	80.37	51.23	75.07
1st	keystrokes	58.19	89.02	58.20	89.86	51.53	87.37	37.84	82.02
	language	89.73	95.94	88.10	95.64	78.75	92.55	73.22	88.60
2nd	keystrokes	65.28	91.81	62.80	91.07	61.75	90.76	45.36	85.98
	language	90.95	96.46	88.70	95.93	82.74	94.48	78.42	91.49
3rd	keystrokes	66.01	**92.04**	62.70	**91.20**	63.35	**91.21**	48.22	**86.58**
	language	**90.46**	**96.34**	**89.30**	**96.09**	**83.13**	**94.72**	**79.51**	**92.49**

The results show that:

- The language model correction greatly improves the correct recovery rate for words.
- The recover rates in quiet environment (sets 1 and 2) are slightly better that those in noisy environment (sets 3 and 4). But the difference becomes smaller after several rounds of feedback.
- Correctness of the keystroke position detection affects the results. The recovery rate in set 3 is better than set 4 because of the keystroke location mistakes included in set 4.
- When keystroke positions have been corrected after several rounds of feedback, we achieve an average recovery rate of 87.6% for words and 95.7% for characters.

To understand how different classification methods in the supervised training step affect the results, we rerun the same experiment on set 1, using different supervised classification methods. Table 6.8 shows our results. The best method is linear classification, then Gaussian mixtures, and then neural networks. Experiments with other data sets give similar results.

In the experiments above, we use recordings longer than 10 minutes. To discover the minimal amount of training data needed for reasonable results, we take the first data set

Table 6.8 Recognition rate of classification methods in supervised learning. All numbers are percentages, where "1st" corresponds to the first supervised feedback, "2nd" the second, etc.

		Neural Network		*Linear Classification*		*Gaussian Mixtures*	
		words	chars	words	chars	words	chars
1st	keystrokes	59.17	87.07	58.19	89.02	59.66	87.03
	language	80.20	90.85	89.73	95.94	78.97	90.45
2nd	keystrokes	70.42	90.33	65.28	91.81	66.99	90.25
	language	81.17	91.21	90.95	96.46	80.20	90.73
3rd	keystrokes	71.39	**90.81**	66.01	**92.04**	69.68	**91.57**
	language	**81.42**	**91.93**	**90.46**	**96.34**	**83.86**	**93.60**

Table 6.9 Text recovery rate at each step. With different keyboards.

		Keyboard 1		*Keyboard 2*		*Keyboard 3*	
		words	chars	words	chars	words	chars
unsupervised	keystrokes	30.99	71.67	20.05	62.40	22.77	63.71
learning	language	61.50	80.04	47.66	73.09	49.21	72.63
1st supervised	keystrokes	44.37	84.16	34.90	76.42	33.51	75.04
feedback	language	73.00	89.57	66.41	85.22	63.61	81.24
2nd supervised	keystrokes	56.34	88.66	54.69	86.94	42.15	81.59
feedback	language	80.28	92.97	76.56	91.78	70.42	86.12
Final	keystrokes	60.09	**89.85**	61.72	**90.24**	51.05	**86.16**
result	language	**82.63**	**93.56**	**82.29**	**94.42**	**74.87**	**89.81**

[15] Since EM algorithm is a randomized algorithm, it might get stuck in local optima sometimes. To avoid this, in each of these experiments, we run the same training process eight times and use results from the run with the highest log-likelihood.

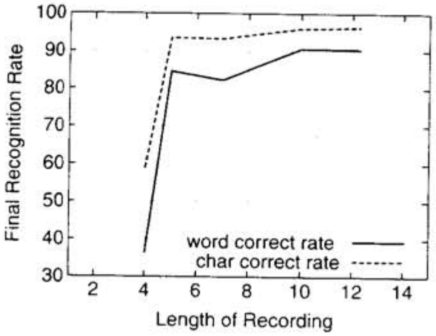


Figure 6.31 Length of recording vs. recognition rate.

(i.e. "Set 1" above) and use only the first 4, 5, 7 and 10 minutes of the 12-minute recording for training and recognition. Figure 6.31 shows the recognition results we get. This figure suggests that at least 5 minutes of recording data are necessary to get good results for this particular recording.

6.7.4.2 English Text Recognition: Multiple Keyboards To verify that our approach applies to different models of keyboards, we perform the keystroke recognition experiment on different keyboards, using linear classification in the supervised training step. The models of the keyboards we use are:

- Keyboard 1: Dell™ Quietkey® PS/2 keyboard, manufacturer part number 2P121, in use for about 6 months.
- Keyboard 2: Dell™ Quietkey® PS/2 keyboard, manufacturer part number 035KKW, in use for more than 5 years.
- Keyboard 3: Dell™ Wireless keyboard, manufacturer part number W0147, new.

The same document (2273 characters) is typed on all three keyboards and the sound of keystrokes is recorded. Each recording lasts about 12 minutes. In these recordings, the background machine fan noise is noticeable. While recording from the third keyboard, we get several seconds of unexpected noise from a cellphone nearby. The results are shown in Table 6.9. Results in the table show that the first and the second keyboards achieve higher recognition rate than the third one. But in general, all keyboards are vulnerable to the attack we present in this paper.

6.7.4.3 Example of Recovered Text Text recognized by the HMM classifier, with cepstrum features (underlined words are wrong),

```
the big money fight has drawn the shoporo od dosens of companies in
the entertainment industry as well as attorneys gnnerals on states,
who fear the fild shading softwate will encourage illegal acyivitt,
srem the grosth of small arrists and lead to lost cobs and dimished
sales tas revenue.
```

Text after spell correction using trigram decoding,

```
the big money fight has drawn the support of dozens of companies in
the entertainment industry as well as attorneys generals in states,
who fear the film sharing software will encourage illegal activity,
stem the growth of small artists and lead to lost jobs and finished
sales tax revenue.
```

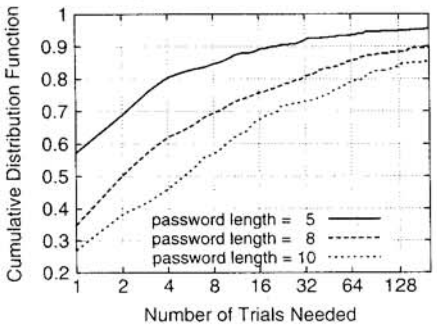


Figure 6.32 Password stealing: distribution of the number of trials required by the attacker.

Original text. Notice that it actually contains two typos, one of which is fixed by our spelling corrector.

```
the big money fight has drawn the support of dozens of companies in
the entertainment industry as well as attorneys gnnerals in states,
who fear the file sharing software will encourage illegal activity,
stem the growth of small artists and lead to lost jobs and dimished
sales tax revenue.
```

6.7.4.4 Random Text Recognition and Password Stealing We used the keystroke classifier trained by set 1 to mount password stealing attacks. All password input recorded in our experiment are randomly generated sequences, not user names or dictionary words. The output of the keystroke classifier for each keystroke is a set of posterior probabilities:

$$p(\text{this keystroke has label } i|\text{observed-sound}), \quad i = 1, 2, \ldots, 30.$$

Given these conditional probabilities, one can calculate probabilities for all sequences of keys being the real password. These sequences are sorted by their probabilities from the largest to the smallest. This produces a candidate list and the attacker can try one-by-one from the top to the bottom. To measure the efficacy of the attack, we use the position of the real password in this list. A user inputs 500 random passwords each of length 5, 8 and 10. Figure 6.32 shows the cumulative distribution function of the position of the real password. For example, with twenty trials, 90% of 5-character passwords, 77% of 8-character passwords and 69% of 10-character passwords are detected. As Figure 6.32 also shows, with seventy-five trials, we can detect 80% of 10-character passwords.

6.7.4.5 Attack Improvements The current attack does not take into account special keys such as Shift, Control, Backspace and Capslock. There are two issues here. One is whether keystrokes of special keys are separable from other keystrokes at signal processing time. Our preliminary experiments suggest this is possible; push peaks of keystrokes are easily separable in the recordings we looked at. The other issue is how modifier keys such as Shift fit into spelling correction scheme. We believe ad hoc solutions such as replacing Shift or Capslock keys with spaces will work. Backspace is also important. The ideal solution would be to figure out what the final text is after applying the backspaces. But that probably will complicate the error correction algorithms. So one could just recognize these keys and leave the "word" before and after out of error-correction because they are probably not full words. Here a bit of human aid could be useful because backspaces are relatively easy to detect by ear based on sound and context, although it is harder than spaces. Assuming this is possible, the classifier can be trained to recognize them accurately.

In future work, it is particularly interesting to try to detect keystrokes typed in a particular application, such as a visual editor (e.g. emacs) or a software development environment (e.g. Eclipse). Examining text typed in these environment presents challenges because more keys maybe used and special keys maybe used more often. Furthermore, the bigram or transition matrix A will be different. Nonetheless we believe that our techniques may be applicable to detecting keystrokes of users in these applications and indeed can even cover input as different as other small alphabet languages, such as Russian or Arabic, large alphabet languages, such as Chinese or Japanese, and even programming languages.

A possible alternative method for feedback training procedure is Hierarchical Hidden Markov Models (HHMMs) [31]. In a HHMM, HMMs of multiple levels, grammar level and spelling level in this case, are built into a single model. Algorithms to maximize global joint probability presumably will result in similar effectiveness as the feedback training procedure. This approach merits further investigation.

We have shown that the recognition rate is lower in noisy environments. Attacks will be less successful when, say, the user is playing music while typing. However, there is research in the signal processing area that separates voice from other sound in the same channel. For example, sophisticated Karaoke systems can separate voice and music. These techniques may also apply here.

Another way to improve keyboard related attacks is to use other types of side channel information, e.g. timing information. Timing information includes the time between two keystrokes, the last time of a keystroke, etc. (See Dawn Song, David Wagner and Xuqing Tian's study [48].) Combining multiple side channels may yield a stronger attack.

6.7.4.6 Defenses To defend against attacks, one can ensure the physical security of the machine and the room. Given the effectiveness of modern parabolic microphones, it must be ensured both that no bugging device is in the room and also that sound cannot possibly be captured from outside the room. The usage of quieter keyboards, as suggested by [22] may also reduce vulnerability. However, the two so-called "quiet" keyboards we use in our experiments prove ineffective against the attack.

The more important message, however, is that the practice of relying only on typed passwords or even long passphrases should be reexamined. One alternative is two-factor authentication that combines password or pass-phrase with smart cards, one-time-password tokens, biometric authentication and etc. However two-factor authentication does not solve all our problems. Typed text other than passwords is also valuable to attackers.

Asonov and Agrawal suggest that keyboard makers could produce keyboards having keys that sound so similar that they are not easily distinguishable. They claim that one reason keys sound different today is that the plate underneath the keys makes different sounds when hit at different places. If this is true, using a more uniform plate may alleviate the attack. However, it is not clear whether these kinds of keyboards are commercially viable. There is the possibility that more subtle differences between keys can still be captured by an attacker. Further, keyboards may develop distinct keystroke sounds after months of use.

Conclusion

Our new attack on keyboard emanations needs only acoustic recording of typing using a keyboard and recovers the typed content. Compared to previous work that requires clear-text labeled training data, this attack is much more general and serious in nature. More important, the techniques we use to exploit inherent statistical constraints in the input and to perform feedback training can be applied to other emanations with similar properties.

REFERENCES

1. http://www.rootsweb.com.
2. Archive.org 20-nov-2001: Bureau of vital statistics, general and summary birth indexes. http://web.archive.org/web/20001120125700/, http://www.tdh.state.tx.us/bvs/registra/birthidx/birthidx.htm.
3. Archive.org 21-jun-2001: Bureau of vital statistics general and summary birth indexes. http://web.archive.org/web/20000621143352/, http://www.tdh.state.tx.us/bvs/registra/birthidx/birthidx.htm.
4. Archive.org birth/death index mainpages for 19-nov-2001 and 05-jun-2002. Comparinghttp://web.archive.org/web/20011119121739/, http://www.tdh.state.tx.us/bvs/registra/bdindx.htmto, http://web.archive.org/web/20020605235939/http://www.tdh.state.tx.us/bvs/registra/bdindx.htm.
5. Census 2000 briefs. www.census.gov/population/www/cen2000/briefs.html.
6. Google phonebook search for "smith" in zipcode 75201 (dallas,tx). http://www.google.com/search?pb=r&q=Smith+75201.
7. National voter act of 1993. http://www.fvap.gov/laws/nvralaw.html.
8. Rootsweb.com ftp server with complete copies of both the marriage and death indexes. ftp://rootsweb.com/pub/usgenweb/tx/.
9. Searchsystems.net listing of texas counties' online public record offerings. http://searchsystems.net/list.php?nid=197http://searchsystems.net/list.php?nid=344.
10. Social security death index. http://ssdi.genealogy.rootsweb.com/.
11. Texas department of health, bureau of vital statistics, marriage indexes. http://www.tdh.state.tx.us/bvs/registra/marridx/marridx.htm.
12. Texas secretary of state voter information. http://www.sos.state.tx.us/elections/voter/index.shtml.
13. Texas state property records. http://www.txcountydata.com.
14. http://www.friendster.com.
15. www.BankOne.com.
16. www.captcha.net.
17. www.Fleet.com.
18. www.namesecure.com.
19. www.orkut.com.
20. Phishing Archive Anti-Phishing Working Group. Your account at ebay has been suspended. http://www.antiphishing.org/phishing_archive/07-26-04_Ebay_(your_account_at_ebay_has_been_suspended).html, July 2004.

21. Phishing Archive Anti-Phishing Working Group. Unauthorized access to your washington mutual account. `http://www.antiphishing.org/phishing_archive/02-24-05_Wamu/02-24-05_Wamu.html`, February 2005.

22. Dmitri Asonov and Rakesh Agrawal. Keyboard Acoustic Emanations. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–11, 2004.

23. Phishing Research at IU. Phishing research group, experiment blog. `http://www.indiana.edu/$\sim$phishing/blog`.

24. Kevin Atkinson. GNU Aspell, 2005. `http://aspell.sourceforge.net/`.

25. Kevin Atkinson. Spell Checker Oriented Word Lists, 2005. `http://wordlist.sourceforge.net/`.

26. Jeff A. Bilmes. A Gentle Tutorial of the EM Algorithm and Its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report ICSI-TR-97-021, International Computer Science Institute, Berkeley, California, 1997.

27. R. Briol. Emanation: How to keep your data confidential. In *Proceedings of Symposium on Electromagnetic Security For Information Protection*, pages 225–234, 1991.

28. Facebook. `http://www.thefacebook.com`.

29. E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In S. Jajodia and P. Samarati, editors, *7th ACM Conference in Computer and Communication Security*, pages 25–32, 2000.

30. A. Ferguson. Fostering e-mail security awareness: The West Point carronade. *Educause Quarterly*, 28, 2005.

31. Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

32. S. Gaw and E. Felten. Reuse and recycle: Online password management (poster). In *SOUPS Symposium On Usable Privacy and Security*, July 2005.

33. Gartner Inc. Gartner study finds significant increase in e-mail phishing attacks. `http://www.gartner.com/5_about/press_releases/asset_71087_11.jsp`, 2004.

34. T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer. Social phishing. `http://www.indiana.edu/$\sim$phishing/social-network-experiment/phishing-preprint.pdf`, Forthcoming.

35. M. Jakobsson, T. N. Jagatic, and S. Stamm. Phishing for clues: Inferring context using cascading style sheets and browser history. `http://www.browser-recon.info/`.

36. Michael I. Jordan. *An Introduction to Probabilistic Graphical Models*. 2005. In preparation.

37. Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.

38. Markus G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–18, 2002.

39. Markus G. Kuhn. Compromising Emanations: Eavesdropping Risks of of Computer Displays. Technical Report UCAM-CL-TR-577, Computer Laboratory, University of Cambridge, 2003.

40. MySpace.com. `http://www.myspace.com`.

41. Texas Department of Health. Bureau of vital statistics, divorce indexes. `http://www.tdh.state.tx.us/bvs/registra/dividx/dividx.htm`.

42. Texas Department of Health. Bureau of vital statistics, general and summary death indexes. `http://www.tdh.state.tx.us/bvs/registra/deathidx/deathidx.htm`.

43. Texas Department of Health. Divorce trends in texas, 1970 to 1999. `www.tdh.state.tx.us/bvs/reports/divorce/divorce.htm`.

44. B. Ross, D. Boneh, and J. C. Mitchell. A simple solution to the unique password problem.

45. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

46. Bruce Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, 2000.

47. Adi Shamir and Eran Tromer. Acoustic Cryptanalysis, 2004. `http://www.wisdom.weizmann.ac.il/~tromer/acoustic/`.

48. Dawn Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proceeding of the 10th USENIX Security Symposium*, pages 337–352, 2001.

49. SVR Group. HTK Speech Recognition Toolkit, 2005. Speech Vision and Robotics Group of the Cambridge University Engineering Department. `http://htk.eng.cam.ac.uk/`.

50. Scott M. Thede and Mary P. Harper. A second-order hidden Markov model for part-of-speech tagging. In *Proceedings of the 37th conference on Association for Computational Linguistics*, pages 175–182, 1999.

51. Philip D. Wasserman. *Advanced Methods in Neural Computing*. Wiley, 1993.

52. Wikipedia. Google bomb. `http://en.wikipedia.org/wiki/Google_bomb`.

53. D. Worthington. eBay redirect becomes phishing tool. `http://www.betanews.com/article/eBay_Redirect_Becomes_Phishing_Tool/1109886753`, March 2005.