Toward Speech-Generated Cryptographic Keys on Resource Constrained Devices
(Extended Abstract)

Fabian Monrose* Michael K. Reiter† Qi Li* Daniel P. Lopresti* Chilin Shih*

Abstract

Programmable mobile phones and personal digital assistants (PDAs) with microphones permit voice-driven user interfaces in which a user provides input by speaking. In this paper, we show how to exploit this capability to generate cryptographic keys on such devices. Specifically, we detail our implementation of a technique to generate a repeatable cryptographic key on a PDA from a spoken passphrase. Rather than deriving the cryptographic key from merely the passphrase that was spoken—which would constitute little more than an exercise in automatic speech recognition—we strive to generate a substantially stronger cryptographic key with entropy drawn both from the passphrase spoken and how the user speaks it. Moreover, the cryptographic key is designed to resist cryptanalysis even by an attacker who captures and reverse-engineers the device on which this key is generated. We describe the major hurdles of achieving this on an off-the-shelf PDA bearing a 206 MHz StrongArm CPU and an inexpensive microphone. We also evaluate our approach using multiple data sets, one recorded on the device itself, to clarify the effectiveness of our implementation against various attackers.

1 Introduction

Futuristic mobile computing platforms will offer, and in some cases will require, methods of user input other than a keyboard, mouse or joystick. This is especially true for head-mounted displays and other wearable computers (e.g., see [22]). For such futuristic devices, and even for next-generation PDAs and programmable mobile phones, voice is a leading contender for the dominant user input medium.

We argue that if voice prevails in this sense, then this poses a challenge for securing data on these devices. On the one hand, if our experience with laptop computers and mobile phones is any indication, then these devices will be stolen frequently: Laptop theft is already the second leading quantifiable cost to enterprises from IT-related security threats [19]. Similarly, mobile phones are the object of theft in four of every ten personal robberies in several cities in the United Kingdom, and these areas logged a fourfold increase in personal robberies involving mobile phones between 1998–99 and 2000–01.[1] These trends suggest that encryption of any sensitive data on such devices is prudent. On the other hand, presuming that these devices will not be tamper-resistant, the cryptographic key with which such encryption can be performed would need to be derived from the voice input of the user, presumably some form of spoken passphrase. Unfortunately, spoken passphrases are likely to have far less entropy than typed ones, due to their need to be pronouncible and due to other forms of information loss in a spoken representation (e.g., capitalization and punctuation).

In this paper we describe an implementation of an approach to derive a repeatable cryptographic key from spoken user input, in which the entropy of the key is drawn from both the passphrase that is spoken and the speech patterns of the user while speaking it. In this way, even if the entropy of the passphrase space is small, the variability across users' vocal tracts will pose an additional obstacle to the cryptanalysis of the key. Moreover, our approach uses techniques designed to withstand an at-

*Bell Labs, Lucent Technologies, Murray Hill, NJ, USA; {fabian,qli,dpl,cls}@research.bell-labs.com

†Department of Electrical and Computer Engineering and Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA; reiter@cmu.edu

[1] See http://news.bbc.co.uk/hi/english/uk/newsid_1440000/1440863.stm.

tacker who captures and reverse-engineers the device on which the key generation takes place (but not while it is taking place), i.e., an attacker who has full access to the stable storage of the device. Our general approach for achieving this was discussed in [16], though as only an initial step toward this goal, that work evaluated the approach only for the generation of 46-bit keys, using only utterances recorded over phone calls, and without regard for the difficulties faced in implementing the approach on resource-constrained devices. Here, we provide a somewhat more realistic evaluation of this approach using a full implementation on an off-the-shelf PDA (the Compaq IPAQ), using data recorded on that PDA, and targeting 60-bit cryptographic keys. We detail numerous changes and refinements we needed to make the approach viable on this platform. We will also give evidence to suggest that the adversary gains little by knowing the user's passphrase, and that the advantage the adversary gains by additionally recording the user saying phrases other than the passphrase is less than one might expect.

We caution the reader, however, of several limitations of our analysis and our approach. First, though we demonstrate the reliable re-generation of a key using an 60-bit characterization of the user's utterance, we do *not* claim to necessarily achieve keys with 60 bits of entropy. Indeed, one can draw few conclusions regarding key entropy from the limited user studies [16, 17] that we have been able to perform. That said, our studies suggest that the technique we have implemented does draw significant entropy from passphrase utterances, and at the very least should offer greater entropy than the passphrase space alone. Second, as our approach strives to re-generate a cryptographic key whenever the legitimate user utters her passphrase, it is necessarily vulnerable to an attacker who can obtain both the user's device and a high-quality recording of the user uttering her passphrase; this exposes all the user's keying material, after all. While any biometric is vulnerable to such an attack, we raise this point here to emphasize the primary attacker with which we are concerned: the attacker who captures the device but that does not have access to the user. That said, we do show, somewhat surprisingly, that knowledge of the user's passphrase seems to help the attacker little, and that even recordings of the user saying other phrases helps only marginally.

2 Background

In this section we give the background necessary to describe our contributions in this paper. Our general approach to generating a cryptographic key from a spoken utterance of a passphrase is described in [16] and is based on an approach initially developed for generating a key from a typed password and the user's keystroke timings during the entry of that password [15]. How this paper relates to these prior publications is described below.

In our approach, the system is initialized by first generating a cryptographic key K, and then generating a collection of $2m$ *shares* of K using a *generalized secret sharing scheme* (e.g., see [24] for a survey of such schemes), where m is a system parameter. These shares are aligned in a $m \times 2$ table that is stored on stable storage. The shares must be generated and placed within the table so that K can be reconstructed from any set of m shares consisting of one share from each row of the table.

Upon entry of the passphrase, the system measures m *biometric features* of the user's entry of the passphrase. We denote the i-th feature by ϕ_i, and denote the value of feature ϕ_i on the ℓ-th (successful or unsuccessful) attempt to log into this user's account (i.e., generate this user's key) by $\phi_i(\ell)$. In the case of a spoken passphrase, the features ϕ_i are features extracted from the user's utterance as described in [16]. For the ℓ-th login attempt, the system then generates a bit string $b_\ell \in \{0,1\}^m$ from $\phi_0(\ell), \ldots, \phi_{m-1}(\ell)$; b_ℓ is called a *feature descriptor*, and the i-th bit of b_ℓ, denoted $b_\ell(i)$, is determined from the i-th feature $\phi_i(\ell)$. Algorithms for generating b_ℓ from $\phi_0(\ell), \ldots, \phi_{m-1}(\ell)$ are proposed and evaluated in [16], but for the purposes of this paper, the reader can think of b_ℓ being determined by

$$b_\ell(i) \leftarrow \begin{cases} 0 & \text{if } \phi_i(\ell) < \tau_i \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

where τ_i denotes some fixed threshold value. The system then attempts to reconstruct K using the table elements at positions $\langle i, b_\ell(i) \rangle_{0 \le i < m}$. When the login index ℓ is not necessary in the discussion, we will typically omit it.

After a history of feature descriptors from successful logins is observed (i.e., logins in which the key was successfully reconstructed), those elements of the table that are not typically accessed by the user are perturbed randomly. So, for example, if the

feature descriptors b induced by a user's biometric measurements are such that $b(4) = 1$ consistently, then the $\langle 4, 0\rangle$ element of the table is randomly altered. If $b(i)$ is sufficiently consistent that element $\langle i, 1-b(i)\rangle$ in the table is perturbed in this way, then $b(i)$ is called a *distinguishing feature*. We will give a precise characterization of distinguishing features in Section 5.

The correct user, when inducing feature descriptors consistent with those she has induced in the past, should not encounter any of the altered elements in the table. We have found, however, that in practice it is necessary for the system to attempt reconstruction using feature descriptors within some Hamming distance $c_{\max}$ of the induced feature descriptor, to correct for up to $c_{\max}$ "errors" by the user (e.g., see [15]). This results in up to

$$\sum_{i=0}^{c_{\max}} \binom{m}{i} \tag{2}$$

secret sharing reconstructions per key (re)generation attempt.

Security of this technique requires that an adversary who captures the device be unable to efficiently differentiate a random table element from a valid share of K. If this is the case, then an adversary may be forced to simply guess feature descriptors until he finds K. If d table elements were randomized (i.e., there are d distinguishing features), then 2^{m-d} out of the 2^m possible feature descriptors yield K, and so the probability that a randomly chosen descriptor will reconstruct the secret is 2^{-d}.

Specific secret sharing schemes for populating this table were investigated in [15]. That paper also included an evaluation of this approach with feature descriptors of length $m = 15$ derived from the keystroke timings of a user while typing an 8-character password. There, we evaluated an implementation in which the table was additionally encrypted with the password; in this way, the technique serves to render a dictionary attack against the password up to 2^{15} times more difficult. Our subsequent work on voice features [16] described algorithms for generating feature descriptors from the user's voice while speaking a passphrase. It further evaluated the security and reliability of the resulting technique with feature descriptors of length $m = 46$ derived from preexisting recordings of users over a phone line. However, in contrast to the keystroke case, here our evaluation presumed a table that was *not* encrypted with the passphrase, in order to avoid the costs of automatically recognizing the spoken passphrase (to decrypt the table). In this case, $m = 46$ does not provide nearly enough security for important applications.

In this paper we address the computational challenges of performing key reconstruction on a resource-constrained PDA with more realistic parameters than our previous voice study explored. Specifically, we evaluate our implementation of this approach for feature descriptors of length $m = 60$, and argue that regenerating the key K can be reliably achieved on a 206 MHz StrongARM processor by correcting for up to $c_{\max} \approx 5$ errors (in the sense described above). The challenges in achieving this are the front-end signal processing needed to keep $c_{\max}$ small so that expression (2) remains manageable, and in devising a secret sharing scheme and corresponding reconstruction algorithm that permits this reconstruction to occur in a reasonable amount of time on this platform. Consequently, we focus on these contributions in this paper, and refer the reader to [16] for the algorithmic details comprising other steps of the key (re)generation process.

3 Front-end Signal Processing

As described in Section 2, the front-end signal processing performed by the device is critical for efficiently generating a key from the user's utterance of her passphrase. Intuitively, the goal of this signal processing is to translate the sound uttered by the user—which is received in the device as a series of amplitude samples from its microphone and analog-to-digital (A/D) converter—into a representation suited for the generation of a feature descriptor (using the algorithms of [16]). Ideally, this signal processing should yield a representation that is as "clean" as possible, in that inter-word silence and background noise affect this representation as little as possible. The less silence and background noise in the representation after signal processing, the more consistent the user's utterances will be (thereby increasing d and the security of the scheme) and/or the less error correction will be necessary in the later stages of key generation (i.e., the smaller $c_{\max}$ and expression (2) can be).

Of course, the benefits of signal processing in terms of producing a clean representation of the user's utterance must be weighed against the computational

cost of the signal processing itself. The challenge is to find the right balance of eliminating environmental effects early via signal processing, versus relying on the error correction in the key generation step to compensate for the effects of noise and silence in the user's utterance. In this section we describe the series of signal processing steps that we believe best achieves this balance. These steps are pictured in Figure 1 and described textually below.

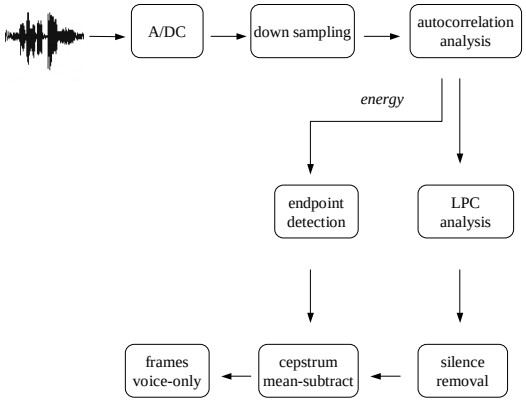

Figure 1: Outline of the front-end modules used for capturing the speech and processing the signal to generate a sequence of frames comprising the voice-only portions in the utterance.

As the speaker utters her passphrase, the signal is sampled at a predefined *sampling rate*, which is the number of times the amplitude of the analog signal is recorded per second. The minimum sampling rate supported by our target platform, the IPAQ^{TM} (see Section 5.1), is 32 kHz; i.e., 32,000 samples are taken per second.[2] Each sample is represented by a fixed number of bits. Obviously, the more bits there are per sample, the better is the resolution of the reconstructed signal, but the more storage is required for saving and processing the utterance. In our implementation, we represent the signal using 16 bits per sample. Therefore, the amount of storage required per second of recorded speech is

$$32000\ \frac{\text{samples}}{\text{second}}\ \times\ 2\ \frac{\text{bytes}}{\text{sample}}\ =\ 64\ \frac{\text{kilobytes}}{\text{second}} \quad (3)$$

Since the utterance of one of our tested passwords can easily be 6 seconds or more, the storage requirements for processing even a single utterance can be significant for a resource-constrained device. This is especially true since, as we have found by experience, writing to stable storage while the recording is ongoing can introduce noise into the recording. In our case, this particularly posed an issue for our experimental evaluation in which we needed to acquire many samples from the speaker; see Section 5.1.

To make subsequent processing on the device efficient, our implementation first makes several modifications to the recorded speech to reduce the number of samples. For example, we *down-sample* the 32,000 samples per second to only 8,000 samples per second, effectively achieving an 8 kHz sampling rate. For most voice-related applications, a sampling rate of 8 kHz is sufficient to reconstruct the speech signal. In fact, nearly all phone companies in North America use a sampling rate of 8 kHz [21].

Down sampling must be performed with some care, however, due to the *sampling theorem* [20]. The sampling theorem tells us that the sampling rate must exceed twice the signal frequency to guarantee an accurate and unique representation of the signal. Failure to obey this rule can result in an effect called *aliasing*, in which higher frequencies are falsely reconstructed as lower frequencies. Down sampling to 8 kHz therefore implies that only frequencies up to 4 kHz can be accurately represented by the samples. Therefore, when down sampling to 8 kHz we use a *low-pass digital filter* with cutoff at 4 kHz to strip higher frequencies from the signal. That is, this filter takes sound of any frequencies as input and passes only the frequencies of 4 kHz or less.

After down sampling, the signal is broken into 30 millisecond (ms) windows, each overlapping the next by 10 ms. Within each window are 240 samples (since 8,000 samples/second $\times$ 0.03 seconds = 240 samples). Overlapping windows by 10 ms avoids discontinuities from one window to the next, and additional smoothing is performed within each window to yield as smooth a signal as possible.

The goal of the next signal processing steps is to derive a *frame* from each window. A frame is a 12-dimensional vector of real numbers called *cepstral coefficients* [20], which have been shown to be a very robust and reliable feature set for speech and speaker recognition. These cepstral coefficients are obtained using a a technique called *autocorrelation analysis*. The basic premise behind autocorrelation analysis is that each speech sample can be approximated as a linear combination of past speech sam-

[2] The IPAQ is claimed to support lower sampling rates, but we have been unable to get them to work correctly under Linux.

ples. The extraction of a frame of cepstral coefficients using autocorrelation analysis involves highly specialized algorithms that we do not detail here, but that are standard in speech processing (linear predictive coding [10]).

A side effect of generating frames is a calculation of the *energy* of the signal per window. The energy of a window is proportional to the average amplitudes of the samples in the window, measured in decibels (dB). Energy can be used to remove frames representing silence (which has very low energy) from the frame sequence, via a process called *endpoint detection* [13]. The silence portions of the feature frames are then removed and the voice portions concatenated.

Final modifications to the frame sequence are made via a technique called *cepstral mean subtraction*. In this technique, the component-wise mean over all frames is computed and subtracted from every frame in the sequence. Intuitively, if the mean vector is representative of the background noise or the channel characteristics in the recording environment, then subtracting that mean vector from all the frames yields a frame sequence that is more robust in representing the user's voice.

After this, the speech data is segmented and converted to a sequence of bits (a feature descriptor) as described in [16]. This feature descriptor is used to regenerate the secret key from the previously stored table of shares, as described in Section 2.

4 Encoding Scheme

The second component on which we focus here is the particular secret sharing scheme we use to populate the $m \times 2$ table described in Section 2 and from which the key K is reconstructed. We quickly found in the implementation of this technique on the resource-constrained IPAQ that the secret sharing schemes suggested in [15] would not suffice. That paper suggests three different schemes. One is sufficiently resource efficient for our purposes but also has potential security weaknesses (see [15, Sections 5.1–5.2]), and while the other two address this weakness [2], they are simply too computationally intensive during reconstruction to permit the degree of error correction we require. Therefore, to achieve sufficiently inexpensive reconstruction, we devised a secret sharing scheme that would permit fast reconstruction and that appears to be secure for our purposes.

We emphasize that the type of security we require for our secret sharing scheme is different from the typical security definition of a secret sharing scheme. The latter definition is, informally, that an adversary not possessing a sufficient set of shares be unable to reconstruct the secret. In our case, however, the adversary who captures the device is in possession of all shares in the table, and so clearly possesses enough shares to reconstruct the secret. Our security requirement is rather that the adversary be unable to efficiently *find* a sufficient set of *valid* shares in the table, i.e., a set containing a valid share from each row of the table and no invalid (random) elements. Ideally, the best the adversary could do would be to repeatedly try reconstruction with a randomly chosen set containing one element from each row. However, because the invalid shares are placed according to an unknown distribution determined by the biometric features of the user—and not uniformly at random—it is impossible to formally reduce the security of such a scheme to a well-known cryptographic problem. (Obviously there are distributions that would leave the scheme trivially breakable.) As such, until we find a better way to model security, we are stuck with heuristically secure schemes; the approach described in this section is one. Nevertheless, we will comment in detail about our current knowledge of the security of this scheme in Section 4.4.

4.1 Initialization

Here we do not describe the secret sharing scheme is its full generality, but rather describe how it is instantiated for our particular purposes. When a device is initialized for a user, the device first selects a random $(m-1)$-element column vector $\underline{s} \in \mathbb{Z}_p^{m-1}$ for p a prime. Here, p can be small; $p = 2^{31} - 1$ is a suitable choice, so that arithmetic on 32-bit processors is very fast. The vector $\underline{s}$ is a secret vector, the recovery of which is necessary to obtain the key K. For example, K can be defined as $K = h(\underline{s})$ for h a one-way function, or K can be encrypted with $h(\underline{s})$.

The second step in initialization is to generate a random $2m \times (m-1)$ matrix $\underline{U} = (u_{ij})_{0 \le i < 2m, 0 \le j < m-1}$ where each $u_{ij} \in \mathbb{Z}_p$. $\underline{U}$ is a data structure that

is stored in addition to the table, though if $\underline{U}$ is generated pseudorandomly, then storing the seed of the pseudorandom process is sufficient for $\underline{U}$ to be regenerated when needed. The $2m$-element table $\underline{t} = (t_0, t_1, \ldots, t_{2m-1})^T \in \mathbb{Z}_p^{2m}$ is then generated as $\underline{t} \equiv_p \underline{U}\underline{s}$ (where "$\equiv_p$" denotes equivalence modulo p). That is, $\underline{t}$ is the table as described in Section 2; intuitively, the element in the i-th "row" and j-th "column" for $0 \leq i < m$ and $j \in \{0, 1\}$ is t_{2i+j}.

To complete initialization, $\underline{s}$ is deleted, and $\underline{U}$ (or the seed needed to regenerate it), the table $\underline{t}$, and prime p are stored for the next key regeneration attempt. In addition, $y = h'(\underline{s})$ is stored for some (different) one-way and collision-resistant function $h' \neq h$, so that when $\underline{s}$ is reconstructed, it can be recognized as correct.

After a sufficient number of successful key reconstructions (see Section 4.2), the table $\underline{t}$ is "hardened" as described in Section 2: if over a number of successful key reconstructions, each induced feature descriptor b is consistent on the i-th feature (i.e., $b(i)$ is usually the same, as specified more precisely in Section 5), then element $t_{2i+(1-b(i))}$ is assigned to be a random element of $\mathbb{Z}_p$. This should usually not affect reconstruction for the correct user, since that user typically selects $t_{2i+b(i)}$.

4.2 Key reconstruction

As described in Section 2, the input from the user, in this case an utterance, is used to generate an m-bit feature descriptor b. The key regeneration program constructs a vector $\underline{t}_b \in \mathbb{Z}_p^m$ by selecting the corresponding elements of the table, i.e., $(\underline{t}_b)^T = (t_{2i+b(i)})_{0 \leq i < m}$. It similarly constructs a $m \times (m-1)$ matrix $\underline{U}_b = (u_{2i+b(i),j})_{0 \leq i < m, 0 \leq j < m-1}$. Note that solving

$$\underline{U}_b \underline{s}' \equiv_p \underline{t}_b \tag{4}$$

for $\underline{s}'$ would yield $\underline{s} = \underline{s}'$ if b contained no user errors, and the fact that $\underline{s}' = \underline{s}$ could be confirmed by testing whether $y = h'(\underline{s}')$. However, since instantiating and solving (4) for all the different feature descriptors b' within Hamming distance $c_{\max}$ from b would require too much time on the target device, our error correction strategy takes a different approach.

This faster approach derives from the observation that the equation $\underline{U}_{b'} \underline{s}' \equiv_p \underline{t}_{b'}$ for a feature descriptor b' contains m equations in $m-1$ unknowns, and thus is over-defined. This is intentional, and allows b' to be rejected very quickly if this equation has no solution. Specifically, let $\tilde{U}_{b'}$ be the $m \times m$ matrix whose first $m-1$ columns are the same as $\underline{U}_{b'}$ and whose last column is $\underline{t}_{b'}$. Then, a solution exists only if $\det(\tilde{U}_{b'}) \equiv_p 0$, and so if $\det(\tilde{U}_{b'}) \not\equiv_p 0$ then b' can be discarded from further consideration. Using a recursive algorithm, it is possible to check $\det(\tilde{U}_{b'}) \equiv_p 0$ for feature descriptors b' within $c_{\max}$ errors of b using only one additional Gaussian elimination step per new b'. Due to this feature, our implementation can test over 6×10^6 feature descriptors b' per second when $m = 60$.

Two further observations are worth noting here. First, the determinant of even a randomly chosen matrix is 0 mod p with probability p^{-1}, which is not negligible since p is chosen to be small. Therefore, if $\det(\tilde{U}_{b'}) \equiv_p 0$ and so we proceed to solve $\underline{U}_{b'} \underline{s}' \equiv_p \underline{t}_{b'}$ for $\underline{s}'$, it remains necessary to confirm $\underline{s}'$ by checking $y = h'(\underline{s}')$. Second, there is a small chance that (4) is not solvable even if b is consistent with all the user's distinguishing features, because $\underline{U}_b$ might not have full rank. Under the assumption that $\underline{U}_b$ is chosen uniformly at random, it follows that $\underline{U}_b$ has rank $m-1$ with probability $\prod_{j=2}^{m}(1-p^{-j}) = 1 - p^{-2} + O(p^{-3})$. This probability is much smaller than the probability that the system cannot be solved because the feature descriptor b induced by the user's utterance contains more than $c_{\max}$ errors, and thus can be ignored.

4.3 Performance

For performance measurements we choose to benchmark key reconstruction on a 206 MHz StrongArm and a 500 MHz Ultra II. The first processor is that currently available in the IPAQ$^{\text{TM}}$, on which our current implementation runs. The second processor is in line with the current trends[3] in the hand-held market, and thus, allows us to forecast expected performance on future PDAs. Our performance benchmarks consists of a collection of `C/C++` modules cross-compiled for the ARM, comprising of signal processing code, a patched cryptographic library based on `cryptolib1.2` [11], and a matrix manipulation package `newmatv9.0` [5] for implementing the segmentation algorithms outlined in [16].

[3] The Intel 80200 processor based on the Arm compliant XScale$^{\text{TM}}$ microarchitecture that supports 400, 600, and 733 MHz CPUs is expected to be widely available soon (see http://developer.intel.com/design/iio/prodbref/80200.htm).

To test the performance of our key reconstruction routines we devised the following experiment. First, an $m \times 2$ table of shares of the key K is generated as outlined in Section 2, i.e., as a user's key regeneration table would be initialized in practice. Then, d rows of the table (features) are selected at random to be distinguishing, and one element of each of these d rows is perturbed randomly—as if the user were consistent in utilizing the other element of this row (see Section 2). Finally, a feature descriptor b is chosen with the property that $c \leq c_{\max}$ of the d distinguishing features (chosen at random), say $b(i_1), \ldots, b(i_c)$, are "errors", i.e., are set to select the *randomly perturbed* elements of rows $i_1, \ldots, i_c$. The key reconstruction process performs a number of reconstructions that depends on c in its attempts to correct for such errors; the number of reconstructions performed in the worst case is shown in expression (2) of Section 2. Our benchmark is the amount of time required to reconstruct the key K on average, which is a rough measure of the time required to perform

$$\frac{\binom{m}{c}}{2} + \sum_{i=0}^{c-1} \binom{m}{i} \tag{5}$$

secret sharing reconstructions and test each for correctness. Note that for $c = c_{\max}$, (5) is less than (2) by $\binom{m}{c}/2$ since on average, reconstruction succeeds after searching half of the feature descriptors that correct b on exactly c locations.

Our results for this benchmark, shown in Figure 2, are significantly less than multiplying (5) by the time to perform and test a single reconstruction in our secret sharing scheme. The reason is due to the significant optimizations that can be achieved as described in Section 4.2.

4.4 Security

One potential security weakness of this scheme is the fact that an adversary who captures the device can conceivably reconstruct $\underline{s}$ from not just one element of the table per row, but instead using *any* m elements of the table. For example, if the adversary had reason to believe that the first $m/2$ rows contained no distinguishing features—and thus all table elements in these rows were valid—then the adversary could set $t'[i] = t[i]$ for $0 \leq i < m$ and $\underline{U'} = (\underline{u}_i)_{0 \leq i < m}$ and use these to solve for $\underline{s}$. Therefore, it is important to use only features that are more likely than not to be distinguishing.

We now describe the fastest attack on our scheme of which we are aware. An attacker who captures the device on which the key is generated but who has no information about the user's distinguishing features may attack the system by repeatedly guessing a feature descriptor b at random and solving (4). If there are d distinguishing features then each guess will be successful with probability 2^{-d}, but will require the attacker to solve a system of m linear equations, which is quite time consuming. A faster approach is to choose feature descriptors $b_0, b_1, \ldots$ such that each differs from the last in one bit. Then, computing $\underline{U}_{b_i}^{-1}$ requires only one new Gaussian elimination step per b_i, and the further optimizations outlined in Section 4.2 can also be applied in this case.

The expected time for this attack to succeed can be computed as follows. Assume that b_i and b_{i+1} differ in exactly one position that is chosen at random. Let $G(c)$ for $c \geq 2$ denote the expected number of Gaussian elimination steps performed until reaching a b_i with no errors (i.e., that is consistent with all of the user's distinguishing features), assuming that b_0 has c errors. Note that b_{i+1} has a different number of errors than b_i with probability d/m, and if it has a different number of errors, then it decreases the number of errors (by one) with probability c/d and increases it with probability $(d-c)/d$. Hence, we get the following equations for $G(c)$.

$$\begin{aligned} G(2) &= 0 \\ G(c) &= \frac{m}{d} + \frac{c}{d} G(c-1) + \frac{d-c}{d} G(c+1) \\ &\quad \text{for } 2 < c \leq d \end{aligned}$$

Solving for these linear equations at $c = d/2$ yields the expected number of Gaussian elimination steps before recovering the key, since a random feature descriptor contains an expected $c = d/2$ errors. Moreover, after the Gaussian elimination step for each b_i, $m(m-1)$ multiplications are required to test b_i on average. So, the total cost of this attack is as shown in Figure 3. (Actually, this is a conservative lower bound, since the cost of each Gaussian elimination step itself is not counted.)

In the empirical evaluation that we perform in Section 5, we evaluate our approach at $m = 60$, which is the smallest value of m shown in Figure 3. Here, if 70% of the features are distinguishing, then this attack conservatively requires an expected 2^{44} multiplications. If 80% of the features are distinguishing, then this attack requires at least 2^{50} multiplications on average. Obviously the security of the scheme improves as m and d/m are increased, and this is a

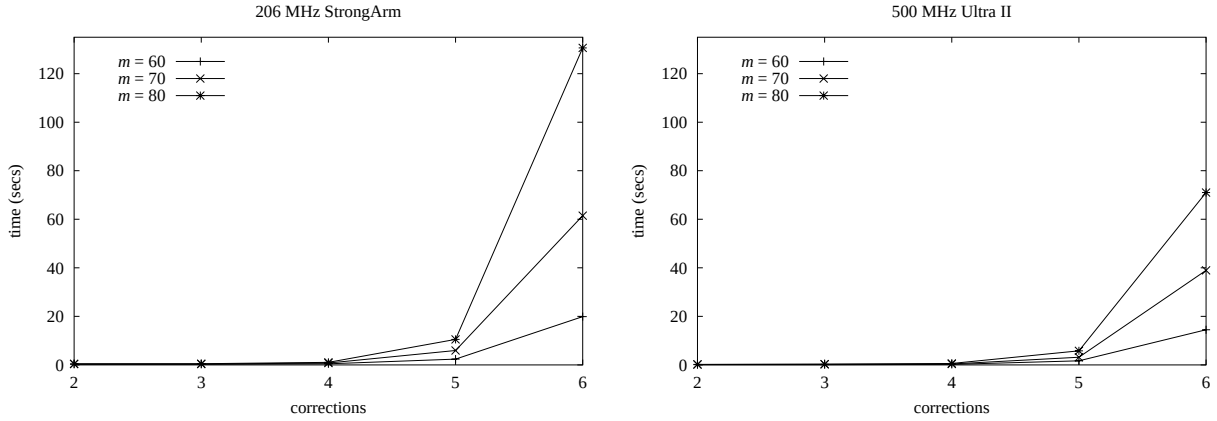

Figure 2: Key reconstruction performance for $m \in \{60, 70, 80\}$. Depicted are average (of 50 executions) elapsed times for key reconstruction on the IPAQ (left) and a 500 Mhz processor which reflects upcoming PDA processor trends (right).

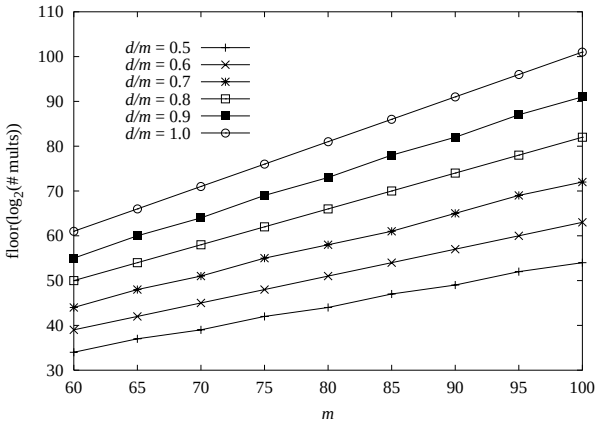

Figure 3: A lower bound on the expected number of multiplications to exhaustively search for the key, assuming that the d distinguishing features are uniformly distributed; see Section 4.4.

goal of our ongoing work.

5 Empirical Results

In this section we empirically evaluate the security of our technique using two different data sets. In these evaluations we attempt to conservatively characterize the security of our technique against an attacker who captures the device. It is clear from Section 4.4 that the number d of distinguishing features is central to the security of our scheme, in that if d is small, then our scheme is vulnerable to a key recovery attack via exhaustive search (see Figure 3). Therefore, in order to demonstrate that our approach is plausibly secure, it is necessary to demonstrate that a high number of distinguishing features can be achieved using our techniques. In addition, we also attempt to characterize the degree to which additional knowledge aids the attacker's quest for the key, in the form of either knowing the passphrase said by the user or having recordings of the user saying phrases other than her passphrase.

We remind the reader that large d is not *sufficient* for strong security. For example, even if all features are distinguishing ($d = m$) for all users, but all users' feature descriptors are identical (and the attacker knows this), then an attacker who captures a user's device can trivially determine the key. Therefore, it is equally important that users' feature descriptors vary widely—or more precisely, are drawn from a distribution with high entropy. An entropy evaluation of user's utterances from phone recordings of users saying the same passphrase is described in [16, 17], and these studies suggest that the entropy available in user utterances it substantial even when users say the same passphrase. As already noted, however, since that study involves only recordings of users taken over phone lines, and since that study is limited to $m = 46$ features, it is insufficient in several ways. Unfortunately, the data sets with which we are presently working (see Sections 5.1 and 5.2) include too few users to enable meaningful measurements of the entropy of users'

feature descriptors, and so here we report results for distinguishing features only.

In order to calculate the average number of distinguishing features per user, it is of course necessary to define when a feature is distinguishing. Let μ_i and σ_i denote the mean and standard deviation of feature ϕ_i over the recent history of *successful* logins.[4] Then we say that the i-th feature is distinguishing if $|\mu_i - \tau_i| > k\sigma_i$ for some parameter $k > 0$. Note that if feature i is distinguishing, then either $\tau_i > \mu_i + k\sigma_i$ and so usually $b(i) = 0$ for the user (see (1)), or $\tau_i < \mu_i - k\sigma_i$ and so usually $b(i) = 1$ for the user. Intuitively, the parameter k tunes the "sensitivity" of the scheme, in that a small k implies more distinguishing features, and a large k implies fewer. Obviously k must be tuned to balance achieving a high number of distinguishing features with enabling the user to successfully regenerate his key reliably, since a higher number of distinguishing features is advantageous for security but also requires increasingly similar utterances to regenerate the key. The parameter k will play a central role in our evaluation.

The features ϕ_i that we use in the balance of this paper are described in [16, Section 3.2]. Each is defined by comparing the position of a vector characterizing a segment of the utterance to a fixed plane. This plane is a parameter of our scheme, and though we will rarely mention it below, it is important for the reader to be aware that the data we present is based on a plane selected, based on our data, to optimize our measures in certain ways. On the one hand, this means that our data presents what *could be* achieved with a good selection of this plane, and is thus optimistic in this regard. On the other hand, since this plane is selected by searching through a small set of candidate planes, (infinitely) many planes are omitted from this search. Consequently, it is likely that planes yielding better measures exist. The experimentation we have conducted thus far does not permit us to conclude how to select this plane in general, and this continues to be an area of our ongoing work.

[4] The "recent history" is defined to be the last h successful logins for some parameter h. Records of the last h successful logins can be stored in a file encrypted with the key that is reconstructed on a successful login. The parameter h will not play a role in our analysis here.

5.1 Evaluation of IPAQ™ recordings

Our first data set was collected on a device much like those we envision for use with our scheme, namely a Compaq IPAQ™ H3600 series. The IPAQ is a personal digital assistant with a 206 MHz StrongArm CPU, a touch screen LCD, 16 MB flash ROM, 16 MB SDRAM, a Philips audio codec UDA1341T (i.e., an A/D with data compression), built-in microphones, a compact flash type-II expansion slot, and a speaker output jack. The audio codec has an integrated analog front-end including automatic gain control, which adjusts the signal strength based on the level of the spoken utterance. The sound driver is full duplex and Open Sound System (OSS) compliant [29], though we encountered some signal problems when recording samples at 8 kHz. For this reason we recorded our samples at 32 kHz and then downsampled to 8 kHz offline; see Section 3.

As illustrated in (3) of Section 3, the storage requirements for merely saving the utterance of a passphrase can be significant. To overcome the storage limitations of this particular IPAQ in light of this requirement—and in particular, to permit saving multiple utterances in our user testing—we used a 1 GB IBM Microdrive™ (in the compact flash expansion slot) as a stable store. However, to avoid recording noise from the Microdrive on disk seeks, the recordings are first written to a primary partition in volatile memory. When the memory capacity is reached, the Microdrive is automatically mounted, the data flushed to an `ext2` file system on the drive, and then unmounted. In the event that a wireless connection can be established, the Microdrive can be replaced with a wireless network card, and the data written to a remote mount point.

The IPAQ was used to record utterances from ten users. All ten users were recorded saying the same passphrase multiple times, which in this case was the address of Carnegie Mellon University: "Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pennsylvania 15213". The user was approximately one foot away from the IPAQ's microphone. The user was required to wait for at least one second between pressing the "record" button of our recording application and speaking, so as to not interleave the voice signal with the device's attempts to perform automatic gain control. (Since the automatic gain control converges within 0.5 seconds on the IPAQ, we later discarded the first half second of recorded speech.) Each utterance was separated

from the next by approximately one minute. The acoustic environment in which these utterances were recorded was a standard office environment, and as such, background noise was significant. Six recordings of each user—the "training" utterances—were used to determine the distinguishing features for that user. The remaining recordings for each user—the "testing" utterances, of which there were six from each user on average—were each used to generate a feature descriptor. Comparing each feature descriptor to the same user's distinguishing features, to determine the number of distinguishing features with which the feature descriptor was consistent, counted as a "true speaker" trial. Comparing each feature descriptor to *another* user's distinguishing features counted as an "imposter" trial.

The results of this analysis are shown in the left side of Figure 4. This graph demonstrates the average number of distinguishing features per user as a function of k, and the average number of these that the feature descriptor of a true speaker or an imposter matched.[5] The error bars on the true speaker and imposter curves show one standard deviation above and below the average.

There are several points worth noting about these results. First, the gap between the "distinguishing features" and "true speaker" points indicates the number $c_{\max}$ of error corrections that would need to be performed during the key regeneration process to achieve a reasonably low false reject rate. For example, if $k = 0.6$, then $c_{\max} \approx 5$ should achieve a reasonable false reject rate, and correcting 5 errors is feasible on today's devices (see Section 4.3). Unfortunately, this data suggests that choosing $k = 0.6$ yields fewer distinguishing features than we would like for security ($d/m \approx 0.5$ only). A second point worth noting is that the human imposters, even when saying the same passphrase as the true user, did not match significantly more of the true user's distinguishing features than if they had simply guessed a random feature descriptor (shown by the "random guessing" line in Figure 4, which is simply half the "distinguishing features" line).

[5] The points for a given value of k were generated using a plane chosen to maximize the number of distinguishing features and, among all such planes, to maximize a weighted average of the number of features matched by the "true speaker" and missed by the "imposter". See the last paragraph before Section 5.1 for a discussion of this plane.

5.2 Evaluation of Speaker-J recordings

In order to evaluate a different model of impersonation, i.e, one where the attacker has knowledge of the speaker being impersonated, we explored a second data set. Our second data set is one collected within the context of a different research effort, and so consequently we had less control over the availability of phrases said by the same user multiple times. This data set consists of recordings of a professional speaker, here called "Speaker-J", taken in a professional studio (i.e., a room with virtually no background noise) using a high-quality microphone. The same microphone was used throughout data collection to ensure flat and consistent frequency response. Consequently, this data set is of a much higher quality than the data set described in Section 5.1. This data consists of recordings collected for two separate experiments, but both spoken by the same speaker, Speaker-J. Part I consists of approximately 1600 sentences (roughly 1 hour of speech) and includes the speaker reading (with consistent voice quality and head-to-microphone distance) both standard newswire text and a prepared script that covers rare combinations of speech sounds. Part II consists of 38 minutes of speech consisting of a group of sentences repeated 7 times, all recorded in a single day. The elapsed time between the two datasets was approximately 1 year.

To evaluate our technique, each of these phrases (in part II) were used as a passphrase in our scheme: five recordings were used to generate the speaker's distinguishing features for a given phrase, and two recordings were used to simulate the speaker attempting to regenerate his key. Specifics about the chosen passphrases can be found in Table 1.

The right side of Figure 4 shows the resulting "distinguishing features" and "true speakers" curves. These curves are analogous to the curves in the left side of Figure 4 with the same labels: the first characterizes the number of distinguishing features for this user based on the training utterances, and the second gives the average number of features on which a feature descriptor generated from a test utterance matched the distinguishing features. If we look at $k = 0.6$, we again see that $c_{\max} \approx 5$ should approach a reasonable false negative rate (and is plausible by Section 4.3). Moreover, according to this data set, the distinguishing features at $k = 0.6$ are approaching a better range for security, with $d/m \approx 0.6$. On the other hand, the higher quality

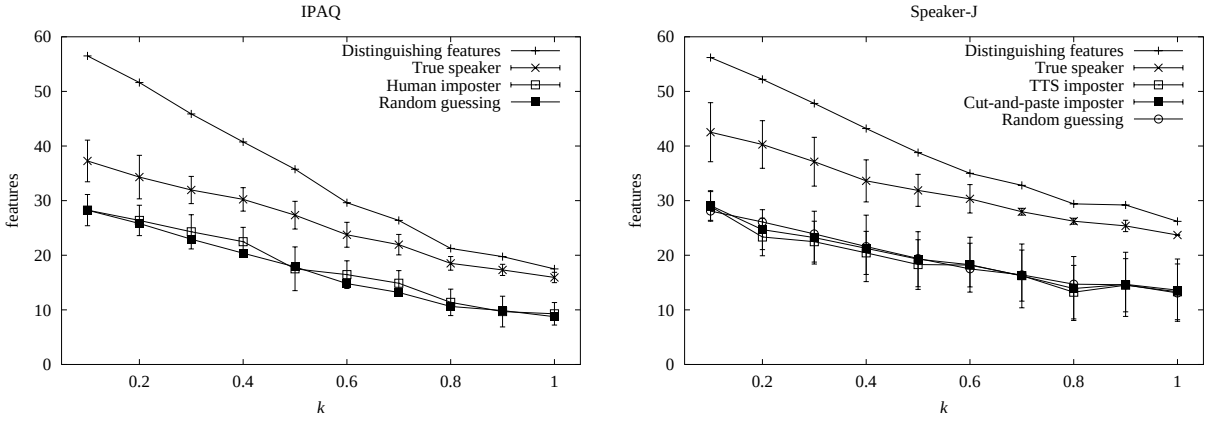

Figure 4: Evaluation of two data sets; see Sections 5.1 and 5.2.

of these recordings may provide a more optimistic picture than would be realized in practice.

Part I of the Speaker-J data set is very rich, and moreover, the research effort that generated this data set carefully annotated the speech, identifying the beginning, ending and midpoint of each *phoneme* it contains. Informally, phonemes are the basic units in the sound system of a language; in the case of English, there are about 50 phonemes. With these annotations, *diphones* can be extracted from the recorded speech. A diphone is a portion of speech beginning in the middle of one phoneme and ending in the middle of the next phoneme; a diphone thus is an example of how the user's speech transitions from one phoneme to another. Diphones reveal much about the voice patterns of the user who uttered them. So, part I of the Speaker-J data set provides the opportunity to attempt a different form of attack against our system, namely one that would simulate an attacker who had a corpus of recordings of the user saying many things other than the passphrase itself. A question we attempt to answer is whether these recordings assist the attacker significantly in finding the user's key.

Specifically, consider an attack in which the attacker wishes to test a candidate passphrase (in our case, selected from part II) but does not how the user speaks it. The attacker uses a text analysis module from a text-to-speech (TTS) system [28] to translate the text of the passphrase into a string of phonemes that realize the passphrase (i.e., a pronunciation for the text), along with other important information that is typically used when synthesizing speech (e.g., the duration and the pitch contour for each phoneme). Of course, any of these features may not match exactly what the user says when she speaks the passphrase. For example, a given word can be pronounced a number of different ways. So, even given the correct passphrase as input, there is no guarantee the text analysis module will yield a string of phonemes that matches the way the user speaks the passphrase. Moreover, the duration and pitch predictions made by the text analysis might differ significantly from what the real user sounds like.

Nevertheless, suppose the attacker possesses a corpus of recordings of the user speaking various phrases other than the passphrase (in our experiment, part I of the Speaker-J data), annotated to identify phonemes and diphones. The attacker can then attempt to construct how the user would say the passphrase, using techniques derived from a concatenative text-to-speech synthesis system (e.g., [12]), in one of the following ways:

Cut-and-paste imposter Concatenate the raw speech samples (diphones, or longer segments) as-is from the inventory. There are various forms of this. On the one hand, the attacker may make no modifications to duration or pitch of the resulting speech. This yields speech that can sound very much like the true speaker, though there can be severe discontinuities at the concatenation boundaries. In addition, such an approach can yield noticeable differences in the recording levels within the passphrase. On the other hand, the attacker can perform minimal signal processing

to match the loudness levels and smooth the discontinuities.

TTS imposter Use a traditional TTS signal processing back-end to synthesize the passphrase. Note that this is designed to produce nice sounding speech, but that it also makes use of the duration and pitch predictions that are output from the text analysis module. If these predictions do not correspond to the way the user actually speaks, this step might impede the attack. For example, the user may have an idiosyncratic way of saying a particular word, either in her passphrase or in the instance in the attacker's recordings of the user.

We experimented with four types of cut-and-paste attacks and two types of TTS attacks. The results of these tests are shown in the right side of Figure 4. The curves labeled "TTS imposter" and "Cut-and-paste imposter" capture the best attacks of each type that we discovered. As the curves demonstrate, these attacks both performed similarly, and outperformed random guessing in some cases. However, it appears that the attacks as we conducted them would fall short of breaking our scheme.

Though part I of the Speaker-J data set consists of 1600 sentences, it is not the case that an attacker would need to assemble of corpus of user recordings of this extent to attack a typical passphrase. Table 1 approximates the average number of sentences and their cumulative duration that the attacker would need to record to obtain the diphones in each of the five passphrases we examined. These numbers were obtained by randomly selecting sentences from part I of the Speaker-J dataset until the needed diphones were obtained.

passphrase number	passphrase phonemes	sentences needed
0	24	340 (805 secs)
1	52	455 (1071 secs)
2	29	1297 (3104.88 secs)
3	27	152 (367.320 secs)
4	18	415 (951.421 secs)

Table 1: Approximate number of sentences attacker would need to record to obtain diphones necessary to reconstruct each passphrase tested.

As speech synthesis technology improves, the size of the corpus of user recordings required to significantly narrow the search for the user's key will only decrease. However, TTS and cut-and-paste attacks of the types we performed require an *annotated* corpus, and achieving this annotation is a very manually intensive process that is typically conducted by speech experts. In the case of the Speaker-J data set, it is estimated that 200 expert-hours of effort was invested in achieving the annotated data set. (It takes about one hour to manually segment one minute of speech.) This is already a significant barrier to an attacker wishing to utilize these avenues of attack. Though automatic labellers are available (e.g., [30]), their performance is poor, and we expect it would substantially increase the error rates for the attacks outlined herein. We do expect, however, that the success of such attacks will increase even for our own data sets, as we explore in more detail ways to improve the effectiveness of these attacks. In the full version of this paper we will provide a more detailed analysis of these threats on a per-passhprase basis. We hope that this analysis will be useful for designing effective countermeasures.

6 Related Work

The only prior work of which we are aware on the topic of generating cryptographic keys from voice utterances is that in which the cryptographic key is merely the text of what is spoken, recognized using standard techniques in automatic speech recognition. (Actually, we are unaware of specific systems that even just do this, but since it is an immediate extension of using a typed password as a cryptographic key, we treat it as obvious prior work.) To our knowledge, our work is the only research toward determining a repeatable cryptographic key that draws entropy from *how* the user speaks the passphrase. How this paper contributes over our own prior publications on this topic was described in Section 2.

That said, there has been work on generating cryptographic keys from biometrics other than voice. The first such work of which we are aware is due to Soutar et al. [25, 26, 27], who describe methods for generating a repeatable cryptographic key from a fingerprint using optical computing and image processing techniques. These techniques generate a key from a two-dimensional image (a fingerprint being the obvious example), but do not seem to be well-suited to the task we pursue here. Solutions based

on this technology are marketed by Bioscript (see http://www.bioscrypt.com/).

A different approach to generating a repeatable key based on biometric data is due to Davida, Frankel, and Matt [4]. In this scheme, a user carries a portable storage device containing (i) error correcting parameters to decode readings of the biometric (e.g., an iris scan) with a limited number of errors to a "canonical" reading for that user, and (ii) a one-way hash of that canonical reading for verification purposes. This canonical reading, once generated, can be used as a cryptographic key, or can be hashed together with a password (using a different hash function) to obtain a key. Juels and Wattenberg [9] generalized and improved the Davida et al. scheme through a novel modification in the use of error-correcting codes, thereby shrinking the code size and achieving higher resilience. These techniques are a different approach for generating cryptographic keys from biometric readings, and reach a correspondingly different set of tradeoffs. Notably, whereas our techniques permit a user to reconstruct her key even if she is inconsistent on a majority of her feature descriptor bits (not uncommon when using voice as a biometric [6]), these techniques do not.

More distantly related work is that of Ellison et al. for generating a cryptographic key based on answers to questions posed to a user [7]. The work is premised on the assumption that questions can be posed that the legitimate user will answer one way but others attempting to impersonate the user will answer another way. Their construction resembles one instance of our techniques, namely that of [15, Sections 5.1–5.2], and in this way their scheme achieves a degree of resilience to forgotten answers. However, Bleichenbacher and Nguyen [2] have shown that the Ellison et al. scheme is insecure, whereas our constructions appear to be much stronger. Another construction similar to that in [15, Sections 5.1–5.2] was used in the design of a forensic database, where a person's medical record can be decrypted only once a DNA sample of the person is obtained (e.g., at a crime scene) [3]. However, this scheme is also insufficient for our purposes, due to the same inadequacies as the scheme of [15, Sections 5.1–5.2].

Impersonation attacks using recordings of the user speaking phrases other than her passphrase, as we explored in Section 5.2, have been previously studied for the purpose of fooling speaker verification systems (e.g., [8, 18, 14, 23]). The approach taken in these works are somewhat different from our exploration here, however. Notably, in [14, 23], the authors describe synthesizing a passphrase using a speaker-independent model, and then adapting the pitch and duration of the synthesized passphrase based on relatively few recordings of the user. The authors give evidence that even these simple attacks can make it difficult to set acceptance thresholds for a speaker verification system. In future work we hope to explore how these techniques can be applied in the context of our work.

7 Conclusion

The viability of (re)generating strong cryptographic keys from voice utterances remains unproven. While we believe that the work presented in this paper offers steps toward achieving this goal and evidence that it can be reached, some critical advances are still required. Notably, our analyses using $m = 46$ in [16] and $m = 60$ here are insufficient for security as required in commercial applications, and our future work will continue to focus on reaching $m = 80$ or higher. More extensive user trials to evaluate the entropy of users' distinguishing features is also needed. And, there remain numerous open questions as to how to tune an implementation of our approach in practice. The analysis of Section 5 hides many parameters from view, and the relationship between these parameters, security and usability need to be further explored.

Acknowledgements

We are especially grateful to Daniel Bleichenbacher for his essential contributions in the design, analysis, and implementation of the encoding scheme in Section 4. Additional analysis of the key reconstruction scheme presented therein will appear in [1]. We also thank Sape Mullender, Peter Bosch and Qiru Zhou for their assistance in analyzing anomalies in the kernel sound modules on the various platforms we have experimented with in this effort. We are thankful to Greg Kochanski for numerous insightful discussions, and Minkyu Lee, for providing us with signal processing code that attempts to correct the discontinuities in the cut-and-paste attacks. We also thank Monica Ullagadi for collecting data.

References

[1] D. Bleichenbacher. Work in progress, 2002.

[2] D. Bleichenbacher and P. Nguyen. Noisy polynomial interpolation and noisy Chinese remaindering. In *Advances in Cryptology – Proceedings of EUROCRYPT '2000* (LNCS 1807), Springer-Verlag, pages 53–69, 2000.

[3] P. Bohannon, M. Jakobsson, and S. Srikwan. Cryptographic approaches to privacy in DNA databases. In *Proceedings of the 2000 International Workshop on Practice and Theory in Public Key Cryptography*, January 2000.

[4] G. I. Davida, Y. Frankel, and B. J. Matt. On enabling secure applications through off-line biometric identification. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 148–157, May 1998.

[5] R. B. Davies. A matrix library in `C++`. September 1997. `http://www.webnx.com/robert`.

[6] G. R. Doddington, W. Liggett, A. F. Martin, M. Przybocki, and D. A. Reynolds. Sheep, goats, lambs and wolves: A statistical analysis of speaker performance in the NIST 1998 speaker recognition evaluation. In *Proceedings of the* 5^{th} *International Conference on Spoken Language Processing*, November 1998.

[7] C. Ellison, C. Hall, R. Milbert, and B. Schneier. Protecting secret keys with personal entropy. *Future Generation Computer Systems* 16:311–318, 2000.

[8] D. Genoud and G. Chollet. Speech pre-processing against intentional imposture in speaker recognition. In *Proceedings of the 1998 International Conference on Spoken Language Processing*, pages 105–108, December 1998.

[9] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *Proceedings of the* 6^{th} *ACM Conference on Computer and Communication Security*, pages 28–36, November 1999.

[10] Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1997.

[11] J. B. Lacy, D. P. Mitchell, and W. M. Schell. CryptoLib: cryptography in software. In Proceedings of the 4^{th} UNIX Security Symposium, Oct. 1993.

[12] M. Lee, D. P. Lopresti and J. P. Olive. A text-to-speech platform for variable length optimal unit searching using perceptual cost functions. In *Proceedings of the* 4^{th} *ISCA Research Workshop on Speech Synthesis*, August–September 2001.

[13] Q. Li and A. Tsai. A matched filter approach to endpoint detection for robust speaker verification. In *Proceedings of IEEE Workshop on Automatic Identification Advanced Technologies* (AutoID'99), pages 35–38, October 1999.

[14] T. Masuko, T. Hitotsumatsu, K. Tokuda and T. Kobayashi. On the security of HMM-based speaker verification systems against imposture using synthetic speech. In *Proceedings of the European Conference on Speech Communication and Technology*, Budapest, Hungary, vol.3, pages 1223–1226, September 1999.

[15] F. Monrose, M. K. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. *International Journal of Information Security* 1(2):69–83, February 2002.

[16] F. Monrose, M. K. Reiter, Q. Li and S. Wetzel. Cryptographic key generation from voice (extended abstract). In *Proceeedings of the 2001 IEEE Symposium on Security and Privacy*, May 2001.

[17] F. Monrose, M. K. Reiter, Q. Li and S. Wetzel. Using voice to generate cryptographic keys: A position paper. In *Proceedings of Odyssey 2001, The Speaker Verification Workshop*, June 2001.

[18] B. L. Pellom and J. H. L. Hansen. An experimental study of speaker verification sensitivity to computer voice altered imposters. In *Proceedings of the 1999 International Conference on Acoustics, Speech, and Signal Processing*, March 1999.

[19] R. Power. 2001 CSI/FBI computer crime and security survey. *Computer Security Issues & Trends* VII(1), 2001.

[20] L. Rabiner and B.H. Juang. *Fundamentals of Speech Recognition*, Prentice Hall, 1993.

[21] R. D. Rodman. *Computer Speech Technology*. Artech House, Norwood, MA, 1999.

[22] A. I. Rudnicky, S. D. Reed, and E. H. Thayer. SpeechWear: A mobile speech system. In *Proceedings of the* 4^{th} *International Conference on Spoken Language Processing*, pages 538–541, October 1996.

[23] T. Satoh, T. Masuko, T. Kobayashi, and K. Tokuda. A robust speaker verification system against imposture using an HMM-based speech synthesis system. In *Proceedings of the European Conference on Speech Communication and Technology*, vol.2, pages 759–762, Aalborg, Denmark, September 2001.

[24] G. J. Simmons. An introduction to shared secret and/or shared control schemes and their application. In *Contemporary Cryptology: The Science of Information Integrity*, IEEE Press, 1992.

[25] C. Soutar and G. J. Tomko. Secure private key generation using a fingerprint. In *Cardtech/Securetech Conference Proceedings*, vol. 1, pages 245–252, May 1996.

[26] C. Soutar, D. Roberge, A. Stoianov, R. Gilroy, and B.V.K. Vijaya Kumar. Biometric encryption$^{\text{TM}}$ using image processing. In *Optical Security and Counterfeit Deterrence Techniques II* (Proc. SPIE 3314), pages 178–188, 1998.

[27] C. Soutar, D. Roberge, A. Stoianov, R. Gilroy, and B.V.K. Vijaya Kumar. Biometric encryption$^{\text{TM}}$ – Enrollment and verification procedures. In *Optical Pattern Recognition IX* (Proc. SPIE 3386), pages 24–35, 1998.

[28] R. W. Sproat. *Multilingual text-to-speech synthesis: The Bell Labs approach*. Kluwer Academic Pulishers, 1998.

[29] J. Tranter. *Linux Multimedia Guide*. O'Reilly and Associates, Inc. September, 1996.

[30] C. W. Wightman and D. T. Talkin. The aligner: Text-to-speech alignment usign Markov models. In *Progress in Speech Synthesis*, Chapter 25, pages 313–323, Springer-Verlag, 1997.