# Keystroke Timing Attacks in a Free-Text Environment

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Joshua Thomas Pereyda

May 2014

Major Professor: Dr. Paul Oman, Ph.D.

## Authorization to Submit Thesis

This thesis of Joshua Pereyda, submitted for the degree of Master of Science with a major in Computer Science and titled "Keystroke Timing Attacks in a Free-Text Environment," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor: _____ Date: _____
Paul Oman, Ph.D.

Committee Members: _____ Date: _____
Gregory Donohoe, Ph.D.

_____ Date: _____
Ahmed Abdel-Rahim, Ph.D.

Department Administrator: _____ Date: _____
Gregory Donohoe, Ph.D.

Discipline's College Dean: _____ Date: _____
Larry Stauffer, Ph.D. P.E.

Final Approval and Acceptance

Dean of the College of Graduate Studies: _____ Date: _____
Jie Chen, Ph.D.

# Abstract

Many secure systems, such as SSH, encrypt communications but fail to obscure inter-keystroke timing data. Attacks on confidentiality based on inter-keystroke timing data have been demonstrated in the literature. However, these past attacks have worked only with fixed data collected in a controlled setting. In order to evaluate the usefulness of this attack in real world applications, it is necessary to examine data typed in a more natural environment, "free text" data.

This thesis explores for the first time keystroke timing attacks in a free text environment. Approaches to timing attacks found in the literature are combined with a free text data set. This thesis demonstrates several difficulties and limitations particular to free text. It also demonstrates for the first time a successful predictive attack on free text timing data. This research pushes keystroke timing attacks closer to a real world setting, demonstrating their potential and limitations in live systems.

# Acknowledgments

I cannot but thank the God who gave me life, breath, and every good thing I have, and his Son, my God and master, Jesus Christ.

I must thank my parents, Eddie and Cindy Pereyda, to whom also I owe a great debt, including the better part of all my writing and speaking abilities.

I am also indebted to my major professor, Dr. Oman, with whom I've been very honored to work. Thank you for the counsel, patience, and encouragement.

Thanks to my committee members, Dr. Gregory Donohoe and Dr. Ahmed Abdel-Rahim, and to the researchers who willingly offered to share the data sets they had earned through no small effort: Dr. Kathryn Hempstalk, Dr. Daniele Gunetti, and Dr. Claudia Picardi. Thanks also to Julia and Ruth Reardon for their generous service and sound advice, and to my mom and little brother, Christopher Pereyda, for their proofreading.

This work stands on the shoulders of countless laborers, for whom I am grateful, and only some of whom are named in the bibliography.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1: Introduction

## 1.1 Keystroke Timing Attacks

Encryption has a wide range of uses in the modern computing world, not the least of which is to hide transmitted data from malicious eavesdroppers. This information may be transmitted locally (in the case of Bluetooth, for example) or over the internet (as with online purchases), and may contain credit card numbers, passwords, personal data, and more. Encryption algorithms and related protocols have therefore been, as one might expect, the subject of numerous attacks.

Attacks against encryption algorithms may look for a mathematical or procedural weakness in the way data is hidden. Attacks against protocols attempt to find a weakness in the way the details of communication are handled — for example, if a protocol sends the encryption key in plaintext, then the data hidden by the protocol can be compromised no matter how strong the encryption algorithm is. One type of attack, known as a side channel attack, does not target the encryption algorithm or its protocol directly. Rather, side channel attacks target some side effect of the algorithm's implementation, such as power usage, acoustic vibrations, or electromagnetic emanations. One particular type of side-channel attack, which we will refer to as a keystroke timing attack, targets certain specific uses of encryption.

For example, whenever a user types on a wireless keyboard, each keystroke typically results in a transmission. While some keyboards may lack encryption entirely, it is common for consumer keyboards to make use of encryption through Bluetooth, or some other protocol. Even though the details of the key-presses are hidden through encryption, the fact that each keystroke results in its own transmission presents a potential vulnerability — information may be gained by paying attention to the time between keystrokes. Likewise, interactive Secure Shell (SSH) sessions, which transmit individual keystrokes across a network, are open to this vulnerability [28]. This keystroke timing

data may in fact be gathered through a number of means, including particular kinds of eavesdropping on multi-user systems [38].

However the timing data is acquired, the data must be analyzed in order to gather meaningful information. Since users typically hit different keys at different speeds, the timing between different keystrokes may provide a hint as to what is being typed. We will use the phrase "keystroke timing analysis" to refer to the process of getting useful information out of keystroke timing data, as it has been used before [15]. In some situations, keystroke timing analysis may be used as a means of authentication. Individual users' typing patterns are recorded and a profile is created, the idea being that a new user who steals the account will have a different typing pattern that can be identified. In other situations, it can be used to compromise confidentiality in a system involving human typing. In this case, a user's timing data is recorded and a profile is created for the purpose of determining what it is that the user typed. Various statistical means, such as Hidden Markov Models (HMMs), may be used to attempt to deduce information about this typed data.

## 1.2 Goals of This Thesis

These keystroke timing attacks have been addressed more than once already [28] [8] [38], but the depth of the research remains relatively limited. In particular, previous attacks have relied on very controlled training data and have, for the most part, targeted only random passwords. The goal of this thesis is to explore keystroke timing analysis when dealing only with free text, that is, unstructured input. Specifically, I will address three major questions. First, do keypair timings in a free text data set form normal distributions? This question has been addressed twice before [28] [38], but not for a free text data set. Second, are word breaks in free text distinct enough to be easily identified? This has been addressed once, but again, only when using data from a scripted laboratory experiment [38]. Third, does the attack become less effective when free text training

data is used? This question has not yet been addressed in the literature.

## 1.3 Organization of This Thesis

Chapter 2 summarizes previous research in this field, including previous attack demonstrations and the related topic of authentication through keystroke timing analysis. Chapter 3 explores various methods through which this attack may be expanded, some of which will be attempted here and some of which will not. Chapter 4 details experimental attempts at improved keystroke timing analysis. Chapter 5 summarizes the experimental results, draws conclusions about the potential of the attack, and suggests future research on the topic.

# Chapter 2: Prior Research

## 2.1 Literature Review

The keystroke timing attack was first demonstrated in 2001 by Dawn Xiaodong Song, David Wagner, and Xuqing Tian [28]. That technique was improved upon by Kehuan Zhang and XiaoFeng Wang in a 2009 publication [38]. Two papers from 2010 and 2011 document similar attacks against personal identification numbers (PIN) systems [11] [37]. The study of authentication through keystroke timing analysis has produced a great deal of research, most of which is only tangentially related [15] [6] [24] [7] [14] [13] [21] [29] [30]. The goal of authentication through keystroke analysis is different enough that most of the classification methods are not immediately applicable for content analysis. However, some insights and data may be very useful. The prior art will be reviewed in detail through the rest of this chapter.

### 2.1.1 1998 Attack Against Trusted Path

In 1998, Jonathan T. Trostle documented a keystroke timing attack against the trusted path feature [33]. As an aside, Trostle introduces the term "Trojan horse non-persistence," defined loosely as the property of a system that keeps user-level Trojan programs from being automatically relaunched at startup or in normal workflow. This property is relevant since a system with Trojan horse non-persistence will be less vulnerable to the information-gathering attacks discussed in his paper. His means of harvesting timing information was through a user-level Trojan, which they note could even be a Java applet. The trusted path feature is designed to create a safe path between the user and the OS which cannot be spoofed and along which things like passwords cannot be intercepted. An example of the trusted path feature is ctrl+alt+delete in Windows; the login/unlock screen is another trusted path. The attack is said to be against the trusted path because it attempts to recover password information as it is passed along this path (say, as the

user unlocks the screen). Trostle primarily describes two types of timing attacks. The first attack takes advantage of the fact that keystrokes cause sizable system interrupts which are easily distinguishable from other interrupts, at least on a quiet system. The attack works by sitting in a loop and periodically checking a clock to identify interrupts. On the testing workstation, keystroke interrupts were found to last about 80–97 microseconds, and to be followed up by a second interrupt about 100 milliseconds later. This timing pattern makes the keystrokes fairly easy to identify. Trostle reports that a timing attack can identify the password length in only a few trials. Beyond this, they note that capital characters follow a distinct timing pattern with the shift key, revealing exactly which characters are capitalized. Using this data, they estimate that a ten character random password is reduced from 65 to 55 bits of entropy. Beyond determining the length and case of the characters, the inter-keystroke timings themselves are not used in the attack.

The second timing attack takes advantage of cache behavior in the system. In systems that use the X11 windowing system, the X Server handles each keystroke that the user types. In handling the keystroke, the system reads a keymap in order to give meaning to the keycode. Since this system used a cache with 32 byte lines, it would load not only the needed mapping, but also several surrounding mappings. The key to the attack is that, if the key typed by the user has a mapping already in the cache, it will take less time to process. Thus, if one character in the password is observed to take less time to handle in the X Server event, we know that it is in the vicinity of a previously typed character. This attack can significantly reduce the search space for the repeated character, since it is known that it is close to a preceding character. Combining these two attacks, the author asserts that a ten character random password can be reduced from 65 bits of entropy down to anywhere between 23 and 45 bits, depending on how characters in the password repeat. For a seven character random password, the password entropy can go as low as 17 bits, making online attacks potentially feasible, depending on how exactly

a system is set up. One major caveat is that this attack requires a much larger number of samples than the first attack. While the first attack can run on "only a few" trials, the second attack can require many more, though the exact number is unclear (likely in the 20–50 range). The example data given indicates that at least 15 trials were required on the testing system, but it is not clear exactly how many are needed or if the number is consistent. It is also noted that there are two other vectors for similar cache timing attacks, though they are not explored in detail.

A third set of experiments are also described which hint that the duration of the OS interrupt may vary with the keystroke, but only when it is echoed back to the user. It is not clear why this is, and the attack is not fully developed, but it will, in any case, require a significant number of trials. Trostle concludes with several suggested countermeasures; given the suggestions, it seems that this timing attack is difficult to fully avoid. In the conclusion, it is emphasized that a network authentication protocol resistant to offline brute force attacks is key, and a necessary accompaniment to strong password policies. The combination of these two security measures will keep this attack from being extremely successful.

### 2.1.2 2001 Attack Against SSH

The first paper documenting the successful implementation of an attack based on inter-keystroke timings was inspired by a vulnerability in SSH [28]. It is noted early on, however, that this vulnerability may apply to any application which attempts to encrypt interactive traffic. Song et al. found two primary flaws: One is that cipher blocks are padded only to eight bytes, which can reveal somewhat fine-grained data about the size of individual messages. The other is that individual keystrokes are sent from the client to the server one packet at a time, with the exception of certain control keys such as Ctrl and Shift. This vulnerability can reveal more subtle but potentially more significant data in the form of inter-keystroke timings, permitting the successful documentation and

execution of an inter-keystroke timing attack. Song et al. note that the latency incurred by SSH and the operating system in getting the packet onto the network are, in general, negligible [28, Section 1].

The first vulnerability, that messages are only padded to eight bytes, can be exploited in certain SSH-specific attacks. For example, when a user types the command "su" and enters their password, the traffic follows a specific pattern. There is a message from the client to the server for both 's' and 'u', along with an echo message from the server (showing the characters on the screen as the user types). The next message from the user, sending the return character, generates a larger reply from the server. This is followed by a series of one-way messages (the su command in Unix does not show the password being typed, and the characters are therefore not echoed back to the client), followed by a final reply from the server. This exchange is distinct enough to make it very obvious when a user is typing their password. Other applications, including PGP and SSH (when used a second time after the user has logged in), have similar patterns revealing when their passwords are being typed. These basic attacks streamline the attack process by making it easy to identify password-timing sequences.

The timing vulnerability itself reveals keystroke latencies and not keystroke durations (time between key down and key up events), due to the fact that SSH does not transmit the key up events. Song et al. focus primarily on password data as opposed to other forms of traffic. They decided to test using artificial, randomly-generated passwords, recording how the users typed it after becoming comfortable with the password. They found that users typed passwords in clumps of three or four letters, with longer pauses between the clumps. This behavior would distort the timing measurements for digraphs that go between the boundaries, so they decided to collect the initial data more systematically. Rather than typing entire passwords at a time, users would type character pairs one at a time, going back to the home row after each pair. Each pair was typed 30–40 times. 142 pairs were tested. Their analysis was based on the mean and standard deviation,

and the initial qualitative analysis showed promising results. As an example, they gave results for `(v,o)` and `(v,b)`, each with a roughly Gaussian distribution. The first pair averaged under 100 ms while the second was closer to 200, with fairly non-overlapping distributions. Having recorded the timings, they identified five major categories, given in order from quickest to slowest:

- Two letter keys typed on alternating hands.
- A letter and a number key, typed on alternating hands.
- Two letters on the same hand but different fingers.
- Two letters typed with the same finger.
- A letter and a number on the same hand.

For initial analysis, they divided the entire latency range into six time chunks (<100ms, 100–150ms, etc.) and identified which categories were more likely to be in each block. For example, keypairs in the 250–300 or >300ms category are most likely from the fifth category, while those in the <100ms category are most likely from one of the first three categories.

Since inter-keystroke timings for a given character pair seem to follow a Gaussian distribution, the authors decide to estimate the likelihood of a digraph from a latency measurement by computing each character pair's mean latency and standard deviation. Graphing all of the normal distributions this way, they identified that most latencies fall between 50 and 250 milliseconds, and that the average standard deviation was about 30 milliseconds (142 character pairs were tested). The graph also demonstrated a severe overlap between normal distributions, meaning that the inference of a keystroke would be a non-trivial task. Before attempting to infer key sequences from the keypair latencies, Song et al. proceeded to estimate the information gain obtained from key latencies, an upper bound on the information that could be gained from any analysis. By comparing

all of the normal distributions, they found that the average information gain in their experiments was 1.2 bits per character pair, though the specific gain for any pair varies widely depending on the measured latency (the lowest point being 100ms, where the gain is closer to 0.5 bits). Although this was computed using only 142 character pairs, the authors assert that the information gain should be similar even when all letter and number pairs are involved. Citing a 1950 paper by Claude E. Shannon [26], they note that the entropy of written English is only 0.6–1.3 bits per character. Thus, the 1.2 bits of potential information per character pair may be very significant (however, it is noted, it is not clear if this gain applies when analyzing natural text or not). Since the information gain is not uniform, some keystroke timings will reveal more information than others. They note that character pairs containing number keys or two keys typed with the same finger may be the easiest to spot and may make a piece of text more vulnerable. The authors also point out that the mean standard deviation for the character pairs is about 30ms, less than the typical standard deviation for round-trips on the internet, 10ms. This means that the attack is feasible even at long distance.

Having established some basic statistical properties of inter-keystroke latencies, the authors attempt to infer actual character sequences from the timing data. They use a Hidden Markov Model and a modified Viterbi algorithm, and find that the information gain is "nearly optimal" with this method. According to the authors, HMMs are used to describe "a finite-state stochastic process" that has the Markov property: that the probability of going from the current state to any other state depends only on the current state, not any of the previous states. In this case, the assumption is that the chance of moving from one letter to the next depends only on the current letter and not on the previous letters. This assumption holds with random passwords, but not with natural text — though this does not necessarily make HMMs useless for natural text. The Markov Model consists of a sequence of states, each state representing a character pair. The Hidden Markov Model describes a situation in which the current state itself cannot

be observed, but only some output. In this case, the HMM outputs are inter-keystroke timings, each state being associated with one timing. A key assumption in any HMM, reflecting the Markov Model assumption, is that the probability distribution of an output depends only on its associated state, not on any previous states. This assumption, the authors comment, might hold for some situations, but may fail in others if the typing of some characters changes the position of the hand, influencing the typing of the next characters. Even given this potentially imperfect assumption, the HMM is still able to yield helpful results. They assume that there are only 142 possible character pairs (the same ones that they gathered data for), and that the probability of a certain latency coming from a certain character pair is given by the Gaussian distribution $\mathcal{N}(\mu_q, \sigma_q)$, where $\mu_q$ is the mean latency for a character pair $q$, and $\sigma_q$ is the standard deviation.

Given this model, Song et al. use a specialized $n$-Viterbi algorithm to find the most likely character sequence given a sequence of latencies. For any set of latencies, they are able to calculate how likely a given character sequence is. The goal is to find the sequence of characters with the highest likelihood. The naive approach would be to calculate this probability for each possible character sequence, giving a complexity of $O(|Q|^T)$, where $Q$ is the total number of possible character pairs and $T$ is the length of the timing sequence (word length $-1$). The Viterbi algorithm uses a dynamic approach, and achieves a running time of $O(|Q|^2 T)$. An important caveat is that, due to significant overlap in the probability distributions of the inter-keystroke timings, the probability that the "most likely" sequence is actually the correct sequence is quite small. This means that they need to find not only the most likely character sequence, but the $n$ most likely character sequences, in hopes that the real sequence will be somewhere in this list. Therefore, they extend the Viterbi algorithm to an $n$-Viterbi algorithm. This algorithm outputs the $n$ most likely character sequences in $O(n|Q|^2 T)$ time. In a password attack, they proceed to run an aided brute force attack. Utilizing the list of sequences rendered by the $n$-Viterbi algorithm, they move from top to bottom guessing passwords.

To measure the effectiveness of their approach, they compute the likelihood that the real character sequence will be in the first $n$ sequences rendered by the algorithm. If the success probability is relatively high for a small $n$, the algorithm is very effective. If a high value of $n$ is required, the algorithm is less successful. The upper bound on the effectiveness of the algorithm is about $1.2T$ bits of information (where $T + 1$ is the length of the sequence, and $T$ is the number of character pairs), corresponding to the theoretical information-gain of 1.2 bits per character pair. To start, they analyze the case where $T = 1$. To simplify the calculations, they assume that each character pair's timing distribution has the same standard deviation — a good approximation, they claim, since most keypair distributions have a standard deviation between 25 and 35 milliseconds. They determined that, for the first character pair, the real sequence appears in the first 70 guesses with 90% probability. Thus, the information gain is approximately $log_2(|Q|/n^*) = log_2(142/70) \approx 1$ bits of information per character pair ($n^*$ is the threshold, 70, corresponding to the 90% success rate; $Q$, 142, is the number of possible 1-character passwords). Therefore, the authors claim, this $n$-Viterbi algorithm is near-optimal. In general, the information gain for a password of length $T + 1$ is approximately $Tlog_2(|Q|/n^*) \approx T$ bits, close to the optimal $1.2T$ bits.

In Section 5 of their paper, Song et al. describe the attack's implementation and experimental results. The attacker program they build is called Herbivore. Specifically, it targets the situation in which an SSH user opens another SSH session from within their current one. In this situation, each password character is transmitted one at a time over the network. Using some SSH-specific vulnerabilities (see the first two paragraphs of this summary), these passwords are easy to identify. To measure the effectiveness of this approach on whole passwords, they divide the position of the real password in the candidate list by the total number of possible passwords, giving some relative placement in the list of all possible passwords. The smaller this number, the more effective the attack, and the fewer guesses an attacker needs to find the password. The

$n$-Viterbi algorithm's complexity is linear with respect to $n$, but the number of potential passwords grows exponentially as the sequences get longer. Since the algorithm only gives a constant factor information gain, the required number of guesses $n$ will also grow exponentially. Thus, the algorithm is expected to be computationally expensive for long passwords — besides this, significant growth in memory usage was also observed. Their experiments indicated that individuals tend to type long passwords in clumps of three to five characters at a time, with longer pauses between the clumps. If these longer timings between segments were used in their analysis, they would probably not be helpful. Therefore, it makes sense to break passwords up into smaller subsequences, dividing on the longer pauses, which are easily identified. Each segment is modeled as its own HMM, and the sequences are combined at the end to form candidate passwords. This approach will both avoid skew due to the longer timings and reduce computation time due to long passwords.

In the experiment, Song et al. train the program on keystroke pairs typed one at a time, but the attack is run on passwords typed all at once. Users are made to type the passwords several times in order to gain familiarity before the actual test run. All passwords are chosen uniformly at random from the available test data. This is the hardest case for an attacker, so the authors expect that less secure passwords will be even more vulnerable to the attack. Tests were run for ten different passwords, each eight long. The mean position of guessed passwords in the candidate list was 2.7% (one outlier was up above 15%, while the lower passwords got down to 0.3%). The median position was about 1%, meaning that half the time the password will be in the top 1% of the candidate list. Therefore, the authors assert, an attacker using Herbivore will, on average, only need to search 1/50 times as many passwords as an unaided brute-force attack. This corresponds to an information gain of 5.7 bits per password, or about 0.81 bits per character pair, slightly less than the predicted gain. The authors attribute this difference to the discrepancy between keystroke timings when typed one at a time vs.

typed all at once in a password. While the testing was performed on a limited set of characters, Song et al. expect that the results will carry over even with an increased character space, and assume that the one of information gain per character pair will remain. If this is the case, the 50 times reduction in search space for a brute force attack will hold for eight passwords. On their example system (840 MHz Pentium III), an eight password chosen from lowercase letters and numbers will take, on average, 65 PC-days to crack (at 250,000 guesses per second). Using the timing attack, that average will be reduced to 1.3 days — a significant improvement.

An acknowledged weakness in this experiment is that the attackers are assumed to have sufficient data on the target user. The question is: Can the attack still be effective even without training data for a specific user? The obvious approach is to borrow training data from another user. To test the effectiveness of this approach, the authors collected two users' statistics for comparison. For those two users, 75% of the character pairs had about the same latency between both users. In other words, the authors say, the average difference between typical latencies (for these two users) is less than one standard deviation. Besides this, the five categories of keystroke pairs (letters typed with two hands, letters typed on the same hand, etc.) behaved very similarly between both users. Based on this data, the authors expect that the timing attack can be effective on users even in the absence of specific training data. In order to test this hypothesis, the authors recorded password timing data from five for four users, including the two for whom data was gathered. They then ran two sets of attacks against all users; the first one used User A's data, and the second User B's. Song et al. observe that the attack works somewhat better when a specific user's training data is available. They also observe that some users' training data will work better than others' when attacking a given user. For example, in their experiment, User C be attacked more successfully using User A's data than User B's. However, the most important observation is that one person's data can be used to attack another person. In their results, in fact, the attack

was generally successful for all four of the users. This knowledge could significantly reduce the overhead required to implement an attack.

Before moving on to countermeasures and related work, Song et al. note that Herbivore could be used to attack other types of text, such as English text or terminal commands. Since the entropy of written English is very low, only about 0.6–1.3 bits per character according to Shannon [26], the attack, which gains about one of information per character pair, may be even more effective here.

In their summary on related work, the authors cite an attack by Kocher in 1995 [19] as well as the 1998 attack by Trostle [33]. They also cite a 2001 paper by Dug Song and "Solar Designer" who independently identified some of the same flaws [8]. Besides pointing out another SSH 1.x-specific flaw, Song and Designer investigated the use of the attack against non-password data (Unix commands), and discussed an SSH-specific method of dealing with backspace characters in a sequence.

Song et al. propose a few countermeasures for SSH against this attack. One part of the attack takes advantage of the fact that the server does not echo back passwords that are being typed. The server could prevent this exploit by echoing back a dummy packet when echoing is turned off. The major element of the attack, though, takes advantage of the fact that keystrokes are sent one packet at a time, revealing the typist's timing information. A naive solution would be to add a random delay to transmitted packets. Besides adding a significant latency for the user (the noise must be significant compared to inter-keystroke latencies), this attack is vulnerable to an averaging attack. If a significant number of sequences can be recorded and averaged, an attacker will be able to effectively filter out most of the noise. Song et al. instead propose that a packet always be sent at regular intervals, say, every 50 milliseconds. Multiple keystrokes may be buffered together, and a dummy packet would be sent if no keystrokes were typed. It is important that the server respond regularly, even to dummy packets, to prevent information-leaking from the response packets. This method would effectively

eliminate leaked timing information at the cost of some network overhead. Even with 50 milliseconds between packets, though, the traffic overhead (at the time the paper was written, and with the system the authors were using) is only 1280 bytes/second, small enough to fit even on a low-bandwidth connection.

### 2.1.3 Designer and Song's 2001 Passive Analysis of SSH Traffic

Solar Designer (pseudonym for Alexander Peslyak) and Dug Song document three of passive attacks on SSH in their 2001 article, "Passive Analysis of SSH (Secure Shell) Traffic" [8]. In the first attack, Designer and Song are able to acquire the length of passwords in an SSH session — either in the session authentication itself, or for applications used in the session. In SSH-1, the length field for encrypted information is sent in plaintext, effectively revealing the password length. In SSH-2, the username is sent with the password, and the length field is encrypted. However, the data is still only padded to 8-byte boundaries (depending on the settings), so the attackers can get some idea about the password length. While Song et al. were able to find the lengths of passwords entered within an SSH session, they did not find the length of the initial SSH password, as in this attack. Designer and Song list three mitigations for this attack. The first proposal is to add null characters onto the ends of passwords to hide the real password length. Strings in C are terminated by a null character, so the password used by the server would stop at the first of these padding nulls. The second proposal, coming from Simon Tatham, is to send multiple messages of increasing length to hide the real password message. The other messages would be SSH_MSG_IGNOREs, which are ignored by the protocol; since the message type is hidden to an attacker, the password length would not be easy to detect. This method is effective with a fairly small amount of overhead, but some implementations, the authors note, may not handle SSH_MSG_IGNORE correctly. In the final solution, specific to SSH-2, a pair of SSH_MSG_IGNORE and SSH_MSG_USERAUTH_REQUEST packets are constructed so that their total length

is fixed. The packets may then be sent to the transport layer together, and the attacker will see only one fixed-size packet. This solution was independently proposed by more than one SSH-2 implementation author; it effectively hides the password length with little overhead and "without reliance on artifacts of protocol implementation."

The second category of attacks takes advantage of the behavior of interactive sessions in SSH. As Song et al. observed in their paper, when password characters are not echoed back to the user (a common practice in Unix applications), the packets in an SSH session only flow one way. Using this vulnerability, Designer and Song were able to easily discern the length of passwords entered within an SSH session. The simple solution to this attack is for the server to always send echo packets, using SSH_MSG_IGNORE packets when a password is being entered. Though this is a fix to this specific timing vulnerability, the authors note that there may be many other ways of figuring out when a password is being entered — e.g., monitoring for related network traffic. They also note that the delays between packets leak information about the types of characters typed. As an example, if the delay is greater than usual, the second character likely took two keystrokes to type (as with a capital letter, for example). While noting the same vulnerability as Song et al., Designer and Song do not go into the same detail in their article. They go on to point out that the size of response packets can be a hint at when commands are typed. As a command is typed, the server echoes back each character in a small packet. If the command's output is large, however, the response after the user hits enter will be noticeably larger. In this way, the number of small packets prior to the large one reveals the length of the command entered. SSH-1 also reveals when backspace characters are being typed, since they generate a 3-character response. Designer and Song note that the inter-keystroke timings could then be used to figure out exactly which command of a certain length was typed. This sort of attack could be particularly effective compared to the generic password-cracking attempted by Song et al., due to the very small solution set when choosing between different Unix commands. The authors expect that any fixes

to these timing vulnerabilities would be impractical for SSH due to increased overhead, contrary to the claims of Song et al.

The final attack involves compression. Compression would disrupt most of the attacks in the last two categories by changing the length of the plaintext. For example, compressing a 24-byte packet may decrease its length to 16 bytes, while compressing another 24-byte packet may leave its length the same. In this way, it is no longer trivial to tell the length of the information being transmitted. However, this change in length is not random and actually depends on the contents of the plaintext, which means that compression may actually open up the protocol to other types of attacks. An example of this type of attack, if compression in SSH-2 is enabled, the SSH_MSG_USERAUTH_REQUEST will be compressed. This means that the length of the message will depend (at least in part) on the entropy of the user's password. The solution to this problem is to send the authentication packets without compression, a non-trivial task in SSH-2.

After explaining the attacks and potential fixes, Designer and Song reference the independent work of Song, Wagner, and Tian [28]. Next, Designer and Song list SSH implementations to which fixes have been applied, including OpenSSH, PuTTY, TTSSH, and Cisco. Some details are provided, but the exact extent of the fixes is not spelled out. The authors also provide an unofficial patch for SSH 1.2.x, source code included. Finally, the authors provide a tool, SSHOW, to demonstrate many of the weaknesses documented in the article. They close by crediting Markus Friedl, Theo de Raadt of OpenSSH, Simon Tatham (the author of PuTTY), and Niels Möller (LSH); Song, Wagner, and Tian; and Ariel Futoransky of CORE-SDI.

### 2.1.4 2009 Attack — Keystroke Eavesdropping on Multi-User Systems

This 2009 paper by Kehuan Zhang and XiaoFeng Wang [38] is in many ways a successor to the 2001 publication by Song et al. [28]. They take the timing attacks documented there further in many ways, but rather than attacking SSH, they implement an attack on multi-user systems. That is, the attack assumes local access to a machine and targets other users on the same system. When a keystroke causes a system call, its extended stack pointer (ESP), which can be read publicly, changes. By reading this pointer, an attacker can learn when a keystroke happens and gather inter-keystroke timing data. From there, a timing attack may be used to learn information about what was actually typed. They use an HMM and successfully attack both passwords and English words.

Introducing the attack, Zhang and Wang comment on the security issues associated with multi-user systems in general. Multi-user systems (e.g., most Windows and Linux desktop systems) are designed so that more than one person may be logged into it at the same time. The potential privacy risks are clear, and most systems take steps to protect users from each other. In general, users are not usually allowed to access other users' data. However, for the sake of convenience, some exceptions are made. For example, the process status command `ps` lists the currently running processes on a Unix system. While this is helpful and necessary for some purposes, it does allow users to see information about each other's running processes, such as PID, executable name, memory usage, and start time. While this sort of vulnerability may seem minor, Zhang and Wang seek to demonstrate that more serious attacks can be developed. Their attack exploits information from *procfs*, the process file system used in Linux, BSD, Solaris, and other Unix-like systems. Procfs contains virtual files (many accessible to all users by default) used by programs like `ps` and `top`; these files contain information about the processes including certain register values. Their attack utilizes information from procfs

to infer the timing of keystrokes the program receives. This information includes the ESP and extended instruction pointer (EIP), found in the Linux file `/proc/pid/stat`, where `pid` is the process's ID. Applications often respond to keystroke input by making certain system calls. When a call is made, the ESP and EIP values will both change, and a sequence of ESP/EIP values forms a sort of signature for keystrokes. By continuously monitoring these values, an attacking program can observe these sequences and tell when a keystroke happens. This attack is very feasible on a multi-core system; a victim program can run on one processor while the attacker runs on another. The authors describe the feasibility of their attack compared to other, similar attacks, describing it on the whole as more feasible. After summarizing their contributions, the authors give an overview of the attack in Section 2.

The attack is described as having two phases. In the first phase, the keystroke timing information is collected; in the second phase, it is analyzed to infer something about the typed sequence. In the first phase, a target application's binary is analyzed to obtain the ESP/EIP sequences that indicate a keystroke event. Next, the application's `stat` file is monitored at runtime to obtain a sequence of ESP/EIP values, from which the inter-keystroke timings are deduced. In the second phase, this timing data is fed into an analysis program that outputs some information about what was typed. After going through a brief example, the authors list the set of assumptions/requirements in their attack. They first assume that an attacker is able to execute programs locally on a target machine; this is the case for most users or for an attacker who has compromised one user-level account. Second, they assume that the system has multiple processors or multiple cores. This is necessary since most system calls start and end within one time slice on a single-core processor. Since single-core processors make different processes take turns, the attacking process will not be able to monitor the target as it is running — it will only be able to see the target's state at the end of its turn. In a multi-core system, however, the attacker can literally run at the same time as the target program, allowing

it to get much more fine-grained information about the target's stack and instruction pointers from the `stat` file. Finally, they assume access to the user's procfs files (a reasonable assumption for Linux) and some amount of training data. They suggest a couple ways of acquiring training data as an attacker. First, using the same method used in the actual attack, an attacker could listen for keystrokes and spot certain processes (such as `su`, the super user program, and `ls`) that fork observable processes on the system. The attacker would then know if the victim typed "ls" or some other set of characters. Another idea is to use insider data such as emails sent to the attacker as known plaintext (in a sense).

In Section 3, the authors go into detail on how the timing information is acquired (phase one of the attack). First, they describe how exactly they obtain the ESP/EIP patterns that characterize keystrokes. During a system call, the extended instruction pointer (EIP) always points to a certain location (the vDSO), and the ESP provides more specific information that helps them identify a keystroke. To find these exact ESP patterns, they use one of two automated techniques. The first technique, differential analysis, is used with applications that are deterministic in that identical input will yield identical ESP sequences any time you run it. The differential analysis technique observes ESP sequences in a running application (using a program analyzer, `strace` in their example), and compares sequences associated with keystrokes with those that are not. The difference between the keystroke sequences and the non-keystroke sequences yields the signature ESP sequence. To check for false positives, they search ESP sequences without keystrokes for instances of the signature sequence — if it is found, the key sequence will yield false positives when running. To perform this check, the authors create a trace window based on the time between the first and last system calls of the key ESP sequence. This trace window is slid along the trace of ESP values. At each step, values within the window are compared against the characteristic sequence to find the longest common subsequence (the longest common subsequence (LCS) problem is

well-known and can be solved efficiently). If a complete match is found, the sequence will not work. If the sequence does work, the longest common subsequence also provides a threshold when looking for real keystrokes. For example, if a characteristic ESP sequence of length six is matched in four of its values in the absence of keystrokes, we know we need to match at least five values to positively identify a keystroke event. The second technique, instruction-level analysis, is used in non-deterministic programs (e.g., GUI applications): Non-deterministic programs (in the context of Zhang and Wang's paper) are programs that may yield different ESP patterns for the same keystroke input on different runs. If they were analyzed with `strace` as in the first technique, the ESP sequence for keystrokes in one run may be very different from the sequence found in another run. In Linux, most X-Window applications are developed with a certain toolkit, GTK+, that uses a function like `gtk_main_do_event(event)` to handle keystroke events. The analyzing program looks for keystroke-related calls to this function; when one happens, the system calls are recorded to get the characteristic ESP sequence. Once the executable receives another event, the recording stops; this prevents other GUI-related events from interfering with the recorded sequence. The pattern is checked for false positives as in the first technique.

Once the ESP patterns of a keystroke in a given program are known, the next step is to monitor the `stat` file of that program while the victim is using it. Since the `stat` file must be monitored through system calls (which can take a lot of time), the attacking program may end up with an incomplete ESP trace, but in Zhang and Wang's case this partial trace was enough to deduce timing information. If the trace program monitors the `stat` file continuously, it will take a significant amount of CPU time, potentially making it easily discoverable. Zhang and Wang give two solutions. First, the attacker can use information from procfs (the process file system) to concentrate its efforts at opportune times. For example, the program may check on an SSH process every 100 milliseconds, concentrating its efforts only when it sees an `su` fork. Thus, it will be

able to recover the ESP trace more fully when a password is being entered. The second technique Zhang and Wang propose is to use a known attack from 2007 by Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson [34]. Since Unix-like systems count CPU usage only at certain intervals, at the end of every "tick", the attacking program gives up control just before the end of each tick. Thus, it can use the processor without ever being counted.

Once the ESP/EIP trace is recovered, the attacker must compare the recorded sequence with the known sequences that indicate keystroke events. Two problems make this process more difficult. First, if a system uses address space layout randomization (ASLR), the ESP values will vary between different runs of the program. This means that recorded ESP and EIP values belonging to keystrokes will be different than the ones identified when first analyzing the program, but the solution is simple. The address of the bottom of a process's stack may be found in the `stat` file (and in the `/proc/PID/maps` file on some systems). By taking the difference between this address and the individual ESP values, we can find the relative ESP values which will be more consistent between runs. While the address of the vDSO (where the EIP points during system calls) may not always be available, it can be identified since the majority of EIP values belong to system calls anyway. The second major problem is that, due to the `stat` file being monitored through system calls, the attacking program's ESP/EIP trace may be incomplete. Therefore, when a keystroke event occurs, only part of the characteristic sequence may be recovered. To solve this, they use partial comparisons, treating each comparison like an LCS problem. When looking at an application trace, they use a sliding window approach as in the false positive test. They borrow the length of the LCS identified in the false positive test to determine a threshold for identifying new keystrokes. As the comparison window slides along the ESP trace, they find the LCS between each snapshot and the ESP sequence characterizing keystroke events. If the LCS is greater than the threshold from the false positive test, and if the observed sequence does not have non-keystroke elements to it, they identify a match. Sometimes,

the attacking program may not be able to recover enough ESP values to spot a keystroke with certainty. In this case, the threshold is lowered to make the keystrokes more identifiable, and some other means must be used to filter false positives. One method is to assume that users will type more than one key at a time; isolated keystrokes may then be omitted. Another method is to only start searching after a certain amount of time has elapsed past the application's start time. In this way, an attacker can avoid searching all the start-up system calls unrelated to keyboard input. Factoring in all these elements, the attacking program identifies matches based on the LCS comparison and records the time differences between them, omitting latencies that fall above a certain threshold.

Once the timings have been identified, the timing analysis (phase two of the attack) must begin; Zhang and Wang describe their approach in Section 4. Zhang and Wang use the $n$-Viterbi algorithm for HMMs described by Song et al. [28], adding a handful of improvements. Two main attack categories are addressed: passwords and English text. When attacking passwords, it may be that a long-term attack can observe multiple password entries. Process names and their owners can be found in procfs, making it easy to see when the same application is being re-used. If multiple password observations are available, the data can be combined to enhance the password attack documented by Song et al. The first way of doing this is to perform a simple average between each inter-keystroke timing, creating a new word sequence based on the averages. The second way is to make a further modification to the Viterbi algorithm, which the authors call the $m$-$n$-Viterbi algorithm. This modified algorithm takes advantage of the extra data in a slightly different way, and requires a large number of timing sequences to function well. However, when it does, it performs better than the averaging method when two keypairs have very close timing distributions.

The second attack category, against English text, involves several improvements or new developments. Zhang and Wang start by pointing out that attacks against English text cannot be tested in the way password attacks can. In a password-cracking attack,

the attacker tests guessed passwords until the right one is found. When snooping on written text, however, no such testing mechanism exists. The only test is to see if the resulting text makes sense, which can be difficult if the number of candidates is large. Furthermore, the trick they gave for attacking passwords (combining multiple samples) may not be feasible, since it can be much harder to identify matching samples of English text. It is easy to identify repeated passwords: every time the `su` program is run, for example, you can expect a password to be typed, but no such trick exists when cracking an email's contents. On the other hand, English text is much less random than password text; the low entropy of English can be leveraged to enhance the HMM approach. The first point they note is that keypairs going from a space to a letter tend to have higher timings than letter-letter pairs. The difference is significant enough (at least in their data) that the spaces can be used as anchor points to identify individual words; any timing higher than a certain threshold is assumed to be a space key. The exact value of this threshold will depend on the user. Taking advantage of the low entropy in English text, Zhang and Wang also use the probabilities of character transitions to modify the HMM used in their analysis. Some characters ('e', example) will be more common in written text than others, and some character pairs (such as 'th') will also be more common. Zhang and Wang use these probabilities to adjust the transition probabilities within their HMM. The original model used when attacking random passwords assumes that the probability of moving from one character to another is always equal, and that the timing difference provides the only hint. The new model, however, assumes that the chance of moving from one character to another depends on the probabilities of written English, and also factors in the timing differences. They also point out that the statistics could be modified to fit the user's writing style if training data exists.

To evaluate the effectiveness of their methods, they ran the attack on three applications: vim, SSH, and Gedit. In Section 5, they summarize the results, discussing the recovery of timing sequences (phase one) before addressing the actual timing analysis

(phase two). The attack used a 2.40GHz Core 2 Duo machine with 3GB of memory, and was run on three different versions of Linux (Red Hat, Debian, Ubuntu). Vim is a deterministic program in that each run will always produce the same set of system calls for the same input, so the differential analysis technique was used. The ESP pattern for keystrokes involved 17 system calls, but only five were identified when attacking. When analyzing the trace, all keystrokes were successfully identified. Furthermore, the timings were accurate to within one millisecond, a very promising result. The SSH application was attacked by looking for a spawned `su` process and tracing SSH keystrokes as soon as the `su` process was identified. Song et al. attacked SSH from the network in their paper [28], but Zhang and Wang's attack is from the local machine. The ESP pattern for keystrokes involved 17 system calls, just like vim, though the system calls themselves were different (seven to ten of the system calls were seen in the attack trace). The experiment was successful, again with a deviation of up to one millisecond. The final test application was Gedit, an X Window text editor. Unlike vim and SSH, Gedit is a non-deterministic application in that different runs with the same input may result in different system calls and different ESP sequences. Instruction-level analysis was used, and the only system call characteristic of keystroke sequences was `gettimeofday()`. While this seems like a generic function, its ESP value actually proved to be a very effective indicator. This system call was caught consistently by the attacking program, making the attack successful. This time, the timings were accurate down to two milliseconds, still pretty good. Zhang and Wang also tested these attacks on servers under different loads, revealing their limitations. The vim and SSH attacks were both 100% successful up to 10% system load, but the Gedit attack started losing keystrokes at 2%. However, Zhang and Wang give three examples of real-world servers and show that it is not unrealistic for servers to operate under these thresholds. They also tried the CPU usage-hiding methods described by Tsafrir et al. [34]. While the CPU-hiding was successful, it brought keystroke detection rates to about 50%, a significant degradation.

While this greatly hinders the attack, the authors assert that it could still be useful in the case when multiple password samples are gathered and combined together. Outside of this situation, an attacker will need to deal with the potentially revealing CPU usage.

After demonstrating the ability to gather timing data through their attack, Zhang and Wang move on to timing analysis, the actual inference of key sequences from timing data. The two main attack categories, passwords and English text, are treated separately. When attacking passwords, Zhang and Wang tested two improvements to the $n$-Viterbi algorithm used by Song et al. [28]. The experiment was performed in the space of 15 keys: 13 letters and two numbers. The researchers took 45 inter-keystroke timing samples from each of the 225 possible keypairs, and verified that the individual timings did indeed form a Gaussian-like distribution. They tested three 8-byte passwords randomly selected from the 15 characters. For each password, 50 samples were collected and the algorithm was run on each one. They only gathered the top 4500 sequences from each run of the algorithm (based on two HMMs each handling four characters), and the password was identified 75% of the time. Averaging the positions of the actual password in the guessed list gave a typical position of 7.8%, 6.6%, and 6.8% for the three passwords. If a password was at 10%, for example, that means it was the 450th password in the list. After running this control (mirroring Song et al.'s experiment), Zhang and Wang tested the averaging approach (in which all 50 passwords are averaged together to get a mean timing sequence). The resulting positions were 0.38%, 0.34%, and 0.05%; every password was a significant improvement. The second enhancement, the $m$-$n$-Viterbi algorithm, resulted in positions of 0.39%, 0.34%, and 0.05%, little change compared to the averaging approach. It is worth noting that 0.05% of 4500 translates to less than three password guesses. As Zhang and Wang point out, the effectiveness of the attack varies with the key combinations involved in the password itself; some letter pairs are more distinguishable than others. Since typing characteristics can vary between users, the attack's effectiveness will depend not only on the password but also

on the user typing it.

To prepare for the English-attack experiment, the authors prepared a list of 2103 random words of three, four, or five letters each. In the experiment, words were randomly taken from the list, with more common words being taken first. Spaces were used to differentiate words from the overall timing sequence. For each word, the $n$-Viterbi algorithm was used to give a ranked list of potential words. From those, a spelling check was used to remove non-words. The test was comprised of 14 three-letter words, 11 four-letter words, and 14 five-letter words. The results were optimistic; 40% of the words appeared in the top ten guesses, and 86% appeared in the top 50. However, the number of potential words is not listed, so it is not clear how much information was gained.

In Section 6, the authors describe future research directions, variations of the attack on other systems, and defensive measures. They point out that their attack only attempts to track the key press event, and that the attack may be enhanced by observing the key up event as well. Only three applications were tested, so the attack's impact on other applications is a potential research direction. It is known, however, that the attack will fail if no system calls are immediately associated with keystroke events (`su` is an example of such an application). They also note that other events, such as mouse moves, may be revealed through the ESP trace. They acknowledge that more can be done to improve their timing analysis techniques. Specifically, the English attack only addressed short words, due to the computational cost when solving HMMs. While the word could be broken down into two or more HMMs, the attack would suffer since it would miss any relations across the HMM divisions. Furthermore, they point out that the HMM approach is fundamentally flawed in this problem space, since it assumes that the chance of going from one letter to the next depends only on the previous letter — not true in written English. The high-order HMM is suggested as a possible alternative approach.

Zhang and Wang list a few related attack vectors that come from the process file system. They suggest interrupt statistics (from `/proc/interrupts/`), network status information (`/proc/net`), and system time as potential attack vectors. Two systems, FreeBSD and OpenSolaris, do not reveal ESP/EIP information, but do reveal applications' system times. Zhang and Wang perform a brief analysis of vim demonstrating that system time (the amount of time the operating system (OS) spends handling calls from the process) can be used to detect keystrokes. One disadvantage is that special keystrokes, such as cursor moves, cannot be identified and filtered as they were in the first attack. It may be that the actual differences in system time indicate what kind of keystroke event occurred, but this question is left to future research.

An immediate means of defense would be to hide the `stat` file by changing its permissions whenever a program starts, but this operation is both tedious and error-prone. Furthermore, hiding the `stat` file will also interfere with tools like `ps` and `top`. Another solution is to modify the Linux kernel to remove ESP and EIP information from the `stat` file, placing them in a separate, more private file. However, other attack vectors (such as the system time) may be available, so there is no guarantee that these patches will plug all side-channels. The authors assert that, given the growing pervasiveness of multi-core systems, the security implications of these files (and public user information in general) should be reevaluated.

Section 7 contains a summary on related work. They indicate that keystroke timing characteristics as a means of authentication has already been widely studied, while relatively little effort has gone into keystroke timings as an attack vector. They cite the previous work by Song et al. [28] and Trostle [33] as the main precursors, noting that timing analysis against cryptosystems has also been investigated. The section ends with a comparison against keyboard acoustic emanation attacks. While these attacks have very high success rates, their use case is different than that of Zhang and Wang's attack, and the authors describe some situations in which their attack is much more applicable.

In closing, Zhang and Wang credit Angelos Stavrou for guidance, some anonymous reviewers for comments on their draft, Rui Wang for experimental assistance, and the National Science Foundation for funding.

### 2.1.5 Keystroke Analysis for Authentication

While there are few publications on keystroke timing analysis as an attack vector, there is an abundant body of research into authentication through inter-keystroke timings [15] [6] [29] [30] [24] [7] [14] [13] [21] [10] [9] [17] [23] [36] [4] [5] [20] [32] [31] [35] [25], including at least two doctoral dissertations [16] [2]. The basic idea is that, since different users type differently, we can use typing patterns as a sort of signature for identification or authentication. While keystroke timing characteristics have been examined as an alternative primary authentication method, the most promising potential seems to be in an intrusion detection system (IDS) [15] [2]. Keystroke timing-based IDSs have the particular requirement that they must handle free text, that is, text entered by users in the course of normal activities. This fills an interesting gap in the prior research — both Song et al. [28] and Zhang and Wang [38] used very controlled typing samples. For their training data, keystrokes were typed in complete isolation from others; for attacks, specific passwords and words were used. Even though the authentication problem is very different from the attack problem, some of the authentication-related research provides useful insights and metrics relevant to key sequence-inference.

For example, Terence Sim and Rajkumar Janakiraman concluded in a 2007 publication [27] that (1) keystroke durations (the length a key is held down) do indeed provide some information above keystroke latencies and (2) the context of a digraph can have a noticeable impact on its latency. Another 2007 paper by Janakiraman and Sim used entire words as identifying features [17]. However, they found that some of the most discriminating features were non-word digraphs and trigraphs such as “an”, “in”, and “ing”. In a 2012 thesis [2], Eesa Alsolami observed several potential features for

authentication/identification, concluding that the most useful features were the most frequently typed $n$-graphs in a sample. Second were $n$-graphs with the greatest time stability. Alsolami followed Kathryn Hempstalk [16] in using a 500 ms threshold to filter the experimental data set — that is, inter-keystroke timings greater than 500 ms were discarded (this may be interpreted as the user stopping to think or take a break while typing). A major caveat when considering all this is that that which is good for authentication/identification may not be good for sequence inference.

Besides all these insights, a major benefit of this research has been the gathering of inter-keystroke timing data [23] [15] [35] [27] [16] [4], allowing for further research without the need to collect new data [2].

While this body of literature is not all examined in detail, one paper is summarized as an example in the following section.

### 2.1.6 Free Text in Keystroke Timing Authentication

In 2005, Daniele Gunetti and Claudia Picardi made a significant publication, "Keystroke Analysis of Free Text." They start out by making the case for authentication based on free text rather than fixed text (fixed text is constrained by the system (e.g., copy typing), while free text is dictated by the user). Briefly, the argument is that a short typing sample will provide insufficient data, while a large typing sample will be obtrusive to users trying to enter a system. The solution is to utilize a large typing sample after the user has logged in. In their summary, Gunetti and Picardi found only three related works dealing specifically with free text [21] [10] [9].

They present two measures used as the basis of their comparisons. The first is the R measure. In the R measure, $n$-graphs (digraphs, trigraphs, four-graphs, etc.) shared between two samples are ordered by latency (or duration, as they call it), and the orders from each sample are compared against each other. This measurement is resistant to changes in overall typing speed — if relative latencies stay the same. In

their experiments, it was found that $n$-graphs were only useful with lengths up to four, though it is expected that longer $n$-graphs will become useful with more data.

The downside of the R measure is that it fails to differentiate users who type similarly but at different speeds. For this reason, the authors also describe the A measure. The A measure is determined by the number of "similar" $n$-graphs, that is, the number of $n$-graphs with latency differences less than a certain threshold. For both R and A measures (domain $[0, 1]$), the smaller the number, the closer the typing samples.

In their experiment, Gunetti and Picardi gathered 15 typing samples (700–900 characters on average) from 40 volunteers The users were encouraged to type whatever they felt in the text box with no hard requirements for length; the typing samples include typos (and sometimes fixes), potentially long pauses, and other sorts of noise expected in a real system. In addition, 165 standalone samples were gathered to simulate unknown attackers.

In their experiments, Gunetti and Picardi compared the effectiveness of R and A measures on digraphs, trigraphs, four-graphs, and combinations. They observed that (1) R measures were seen to function more effectively in differentiating users than A measures; (2) measurements using more than one length of $n$-graph were generally more effective; and (3) measurements using shorter $n$-graphs performed better than those using only longer ones. In the end, the best approaches take into account both R and A measures with multiple $n$-graph lengths.

As one would anticipate, more samples can increase accuracy, but longer samples (both training and testing samples) can have an even bigger impact. Multiple small samples can also be combined into bigger ones without a drastic loss of accuracy. Interestingly, Gunetti and Picardi found that the average distance between two users was larger for skilled typists than it was for unskilled typists. That is, two fast typists will be more distinct than two slow typists (though the difference between a fast and slow typist is even greater).

After many experiments, they ended up with a false alarm rate (FAR) of less than 5% and an impostor pass rate (IPR) of 0.00489% (though the system can be tuned to adjust the FAR/IPR tradeoff). While they acknowledge that the FAR might be too high for an IDS, they suggest using it as a complementary test in combination with other approaches.

Even though the authentication problem is very different from the attack problem, and many of the results therefore do not apply, this paper does raise some interesting questions for key sequence-inference:

- R measures, which are less dependent on absolute inter-keystroke timings, are very helpful in authentication. Can they also be used for character-inference?

- Trigraphs and longer $n$-graphs do provide some information for user-identification; it seems reasonable to infer that they could also provide information for sequence analysis.

- Preliminary research by Song et al. [28] suggested that the analysis attack could work without specific users' information. However, Gunetti and Picardi demonstrate thoroughly that the differences between users can be used to distinguish them. This raises the question: To what extent can training data be shared between users?

# Chapter 3: Expanding Timing Analysis

The two papers by Song, Wagner, and Tian [28] and Zhang and Wang [38] form the basis for this thesis. In both papers, the authors recover inter-keystroke timings and perform some kind of analysis on the timing data to learn about the actual sequences. The results are promising, but with the major caveat that each experiment was performed on a limited alphabet (about 15 keys), and with data collected in a scripted laboratory setting.

The following section summarizes the state of the art of keystroke timing analysis as an attack vector given these two works.

## 3.1 State of the Art

Both papers, while using different means of attack, focus on analysis of the latency between key down events. Key up events are not taken into account. Both papers use training data collected in a very controlled environment and perform the attack on passwords and/or regular dictionary words. The alphabet was limited in order to keep the number of character pairs small. Song et al. calculated a theoretical maximum information gain of about 1.2 bits per character pair. They assert that this would hold with a full alphabet, and question whether or not it would hold if training data were collected from more natural text.

The fundamental means of analysis in both situations is a Hidden Markov Model (HMM) with the $n$-Viterbi algorithm. Song et al. achieve an information gain of 5.7 bits in an eight character password — about 0.81 bits per character pair. Zhang and Wang are able to improve this significantly by using multiple password samples and averaging the timings. In some cases, the accuracy is enhanced 2000 times, but this attack requires more information than the basic attack, which requires only one sample.

Zhang and Wang also expanded the attack into the realm of English text. However,

far from using free text, the targets typed random words selected from those that include only the character pairs for which training data was available. Importantly, they used spaces as anchor points to identify individual words; space to letter transitions averaged much higher than other transitions. The HMM was weighted with the probabilities of going from one letter to another in English, improving the word guesses. Finally, guesses were checked against a dictionary to make sure that the solution pool consisted only of actual words. All this led to significantly improved guessing.

Song et al. ran the attack with multiple users and demonstrated that it can still be effective without a specific user's training data, at least in some cases.

## 3.2 Research Questions

This section provides a list of potential research questions given the current state of the literature:

Both Song et al. and Zhang and Wang performed their experiments on a very limited data set. The training data was collected in a very controlled fashion, and the target data (that is, the timing data actually analyzed) was collected in a somewhat controlled manner as well. In an attack scenario, a victim cannot be expected to provide controlled typing samples; it is likely that free text will be the only text available. This leads to one major question:

- Does the attack become less effective when applied to free text training and target data?

A more specific question is:

- Do character pair timing characteristics become less distinct when working with free text?

Song et al. also address the question of whether target-specific training data is necessary. In a brief experiment, they demonstrate that it is possible to use one user's training

data to infer the typing sequences of another. However, it is not clear if the users were of similar skill level and typing behavior or not. This leads to several questions about the extent to which training data can be shared between users:

- Does shared training data fail for users who type differently?

If it does, we might expect users to fit into different categories of timing behavior. If this is the case, an attacker would benefit from knowing which class the user fit into, leading to another question:

- Can an individual be classified based only on their timing characteristics, without plaintext?

If training data can indeed be shared, as Song et al. have found, it could be useful to combine the training data from multiple users into one aggregate profile. This would also make it easy to compile a sufficient list of digraphs by taking small samples from a large number of people.

- Can training data be combined between users to strengthen the attack?

Terence Sim and Rajkumar Janakiraman [27, p. 4] concluded that, for the purpose of identification and authentication, it is useful to consider not only the time between keystrokes, but the length of time a key is held down. That is, keystroke durations (the length of time a key is held down) provide information that inter-keystroke latencies alone do not. However, keystroke durations have not yet been utilized in an attack in the literature, so the extent of their helpfulness is unclear. Furthermore, if both key up and key down events can be recorded in an attack, it may not be immediately obvious which events are up and which are down. This leaves us with two questions:

- How useful are keystroke durations in key sequence-inference?
- In a sequence of key down and key up events, is it possible to determine which are up and which are down?

Zhang and Wang use spaces to separate words in their fixed text experiment. They found in their data set that space to letter latencies were higher than normal, making them easy to identify with relative confidence. However, their experiment involved strings of unrelated words, so it remains to be demonstrated that spaces are as easily distinguishable in free text. Likewise, it may be that sentence-level analysis could yield results in free text (e.g., by taking advantage of the fact that some words are more common in English than others). This leads to two more questions:

- Are word breaks in free text distinct enough to be easily identified?
- Can an attack against free text be improved by looking at words and sentences as a whole (e.g., the commonality of words in typical sentences)?

Song et al. demonstrated that a set of latency measurements between two keystrokes forms a normal distribution. This understanding was a basic assumption in their simulated attack. Therefore, a natural question, and one with implications for most of the other questions in this section, is:

- Do inter-keystroke timing distributions from a free text data set form a normal distribution?

Hidden Markov models require two major assumptions that do not necessarily fit keystroke timing analysis. Song et al. and Zhang and Wang both suggest the use of high-order HMMs to improve analysis, but neither attempt this implementation. Sim and Janakiraman [27] found that the average latency of a digraph can vary significantly depending on its context, calling into question an attack based purely on digraphs.

- Are high-order HMMs feasible and effective in key sequence-inference?
- Are there other methods that could improve on HMMs by taking into account the context of digraphs, or longer sequences as a whole?

## 3.3 Available Data Sets

Several researchers have already performed the task of gathering inter-keystroke timing data. Typically, the researchers are concerned with identification and authentication, but the raw timing data can still be of use for sequence-inference. The data sets available in the literature are summarized and compared here.

Data sets which were not able to be recovered for this thesis are not covered.

### 3.3.1 Nisenson Data Set

In their 2003 paper "Towards Behaviometric Security Systems: Learning to Identify a Typist," Mordechai Nisenson, Ido Yariv, Ran El-Yaniv, and Ron Meir, gathered what may be the first free text data set in the literature [23]. Gunetti and Picardi define free text as text that is entered without constraints, that is, where the user is free to type what they want (as opposed to copying a predetermined sentence). To put it another way, free text is composed by the user as the user types. There were five "users" and 30 "attackers" in the experiment. The users have a larger set of data for identification/authentication, while the attackers represent unknown attackers to a system and therefore require less data. The researchers used a customized interrupt service routine in Linux to record precise keystroke timings.

The users' input was composed of an answer to an open ended question, a copying of a short sentence, and an entirely free-form input box. On average, the users typed 2551 $\pm$ 1866 keystrokes. The attackers were given two open ended questions and asked to copy the same sentence as the users. Attackers typed an average of 660 $\pm$ 597 keystrokes.

All users apparently typed their data in one session, which Hempstalk notes as a downside [16]. Compared to the Gunetti Picardi data set, this one is relatively small in size. However, it also records key up events (unlike the Gunetti set), giving it one more potentially useful degree of freedom [16].

### 3.3.2 Gunetti Picardi (GP) Data Set

Daniele Gunetti and Claudia Picardi gathered a significant amount of data for their 2005 paper, "Keystroke Analysis of Free Text" [15]. Alsolami referred to this as the "GP" data set [2]. While their experiment involved 40 users and 165 "impostors," the redistributed set has nine less regular users due to privacy-related constraints. [1]

The users were given a free-form input field and told to write until they felt that the box was full. All users typed in Italian. Topic suggestions were given, and the users were more or less performing a creative writing exercise. Altogether, this data set contains 15 samples from each of 31 users, with average lengths varying between 700 and 900 characters. The 165 impostors performed the same exercise only once.

Based on their summary, this data set contains at least $700 \cdot 15 = 10,500$ keystrokes for most users, a significant increase over the Nisenson data set. Likewise, the GP set contains 31 users versus the five in the Nisenson data set. One downside, noted by Kathryn Hempstalk [16], is that this set lacks key up event information, which is a requirement for some identification/authentication algorithms and may also have use for key sequence-inference. Both Song et al. [28] and Zhang and Wang [38] suggested that key up information (which can be used to calculate keystroke duration) may be used to enhance the attack, but neither pursued this hypothesis in detail.

### 3.3.3 Hempstalk sm-150 Data Set

After comparing these two data sets, Kathryn Hempstalk decided to collect a new data set [16]. This data set was made to combine the advantages of the Nisenson and GP sets; it has a size more comparable to the GP data set and records both key up and key down events. Furthermore, it was recorded using a specialized email system, so that the data reflects emails typed in the course of regular usage. On the other hand, previous

[1] Gunetti and Picardi, personal correspondence, 20 March 2013.

data sets were recorded with input made specifically for the application; whether or not this makes a difference is not entirely certain.

After dealing with privacy and technical issues, the redistributable data set, dubbed `sm-150`, contains 15 email samples from each of ten users. Each email has at least 500 events. Since a single key press has both an up and down event, each email contains at least 250 characters. To protect participant privacy, certain identifying information was removed and replaced with the total time duration of the removed text.

### 3.3.4 Si6 Data Set

In a 2010 paper from Si6 Labs, "Collection and Publication of a Fixed Text Keystroke Dynamics Dataset," Luciano Bello, Maximiliano Bertacchini, Carlos Benitez, Juan Carlos Pizzoni, and Marcelo Cipriano document the collection of a data set specifically for redistribution and use by other researchers [4]. The set consists of 66 users, each of whom copy-typed 15 specific Spanish sentences. The data was used experimentally in the same conference by Bertacchini et al. [5].

While this data set consists of fixed text rather than free — that is, users are given sentences which they must copy — the text consists of natural language sentences and may still be useful for analysis. Overall, 282,020 events were recorded, meaning that there were about 141,010 keystrokes (however, some special keys only had their up events recorded due to technical issues). Several digraphs had a high rate of repetition, and the authors were able to verify the consistency of digraphs between sentences.

Sentences were rejected if too many or too few keystrokes were identified, which may cause bias due to the typer behaving more carefully on subsequent attempts. While this data set contains a relatively small amount of data per user, it covers more users than any of the previous data sets. The fact that the typing samples are all fixed may be a downside depending on the application, but it does not necessarily make the data without use.

### 3.3.5 Other Fixed-Text Data Sets

This section summarizes a number of publicly available fixed-text data sets; the list comes from a 2012 survey by Banerjee and Woodward [3].

One 2006 database, prepared by Jugurta R. Montalvão Filho and Eduardo O. Freire [22], contains data from a total of 47 participants. Each of about four select words was typed ten times between two sessions. Only down-down intervals were recorded.

In 2009, Kevin S. Killourhy and Roy A. Maxion developed and published a fixed-test data set for the purpose of comparing various algorithms [18]. The paper was entitled, "Comparing Anomaly-Detection Algorithms for Keystroke Dynamics." The data set consists of 51 users, each of whom typed the same exact password, ".tie5Roanl", 400 times. While this is of very limited use when dealing with free text, it is a sizable and usable data set for its purposes. Both key down and key up events were recorded.

Another 2009 data set comes from "GREYC keystroke: A benchmark for keystroke dynamics biometric systems" by Romain Giot, Mohamad El-Abed, and Christophe Rosenberger [12]. This data set contains 133 individuals with an average of 51 password samples per user. The password was "greyc laboratory." Data was collected for both key presses and key releases.

Jeffrey D. Allen, in conjunction with his 2010 thesis, "An Analysis of Pressure-Based Keystroke Dynamics Algorithms," published a database of password samples documenting over 100 users [1]. There were seven regular users who each had between 89 and 504 entries, and 96 attackers with three to five entries each. Three different passwords were used — one "strong" password, "pr7q1z", one with the author's name, and one simple dictionary word. One unique aspect of this data set is that, besides up and down events, it also contains pressure information as the data was collected with a pressure-sensitive keyboard.

## 3.4 Hypotheses

This thesis will address some of the questions listed in Section 3.2:

1. Do inter-keystroke timing distributions from a free text data set form a normal distribution?

   Hypothesis: Yes. Inter-keystroke timing distributions from free text form a normal distribution.

In order to test this hypothesis, I will plot a histogram with an overlaid Gaussian curve and perform a visual analysis. The results of this experiment will inform experiments for the rest of the hypotheses:

2. Are word breaks in free text distinct enough to be easily identified?

   Hypothesis: Word breaks in free text are not distinct enough to be consistently identified.

In order to test this hypothesis, I will utilize the Hempstalk data set [16]. I will compile averages (and standard deviations) of inter-keystroke timings for each digraph, and will compare space-to-letter and letter-to-space combinations with letter-letter digraphs. In Zhang and Wang's work [38], the average for space-to-letter transitions was significantly separated from the other digraphs. I expect that this difference, while still visible, will be much less pronounced when analyzing the Hempstalk data set. The reason for this expectation is that Zhang and Wang had individuals enter a series of unrelated words, which I think may disrupt the flow of thought and typing. When typing one's own thoughts, an individual may be better prepared to transition between words. The answer to this question should help resolve the final one:

3. Do character pair timing characteristics become less distinct when working with free text?

Hypothesis: Yes. The larger alphabet and less controlled data collection in a free text environment make character pairs harder to distinguish.

In order to test this hypothesis, I will perform an analysis on a limited alphabet. That is, I will filter out all but a limited subset of keystroke pairs from the `sm-150` data set before analysis. Then, I will repeat the same analysis with all keypairs included. I anticipate that the increased alphabet size will make letters harder to distinguish.

All of these questions will give insight into the overarching question:

- How effective is the attack against English text (or other languages, for that matter)?

# Chapter 4: Experimental Details

Section 4.1 in this chapter documents a brief data comparison. Sections 4.2, 4.3, and 4.4 each test one of the hypotheses from Section 3.4.

## 4.1 Data Comparison

This section will explain the choice of data set and compare it with the data used by Song et al. [28] and Zhang and Wang [38].

The Hempstalk `sm-150` data set was chosen to test the hypotheses given in Section 3.4. It meets the most important requirement — that the data comes from free text input. The data was collected from volunteers through a special key

The `sm-150` data set comes from all real-world emails from 10 different users. Each user has 15 emails, and each email has 250 characters or more. Song et al. report collecting 30–40 samples for each of 142 character pairs for two users. Zhang and Wang's data set contains 45 samples for each of 255 character pairs. Both data sets involve numbers and letters only. Since these sets were collected in a more controlled setting, minimum sample sizes for character pairs were deliberately met. In a free text data source, these minimums can only be met incidentally as the overall data set grows. A brief analysis of the `sm-150` data set shows that all but one of the users have between 61 and 93 character pairs with at least 40 samples. This is a significantly smaller amount of character pairs for the given minimum sample size, but I do not expect this to be an issue for the sake of this thesis. One benefit of the `sm-150` data set, on the other hand, is that it has a greater number of character pairs with smaller sample sizes, and several character pairs with much larger sample sizes.

## 4.2 Normality of Inter-Keystroke Timing Distributions

This section will document the experiment designed to test the first hypothesis: "Inter-keystroke timing distributions from free text form a normal distribution." The normal distribution is an important assumption in the analysis in Song et al. [28] and Zhang and Wang's [38] papers. Both papers verify this assumption from the data (in Sections 3.1 and 5.2, respectively), but it has not been explicitly tested for the `sm-150` data set. Hempstalk did, however, base experiments on this assumption with a degree of success [16], indicating that the distributions are at least similar to a Gaussian distribution.

### 4.2.1 Unexpected Anomalies

The `sm-150` data set has data from ten participants, labeled A through J. Participant G was chosen arbitrarily for the initial observation. The following character pairs were chosen for the test:

- H, E (two-hand keypair)
- E, R (same hand, different finger)
- L, L (same finger, same key)
- T, Space (end of word)
- Space, T (start of word)
- Backspace, Backspace

The first three were chosen because they represent two-hand (H, E), same-hand (E, R), and same finger (L, L) digraphs. The next two represent the starts and ends of words, while the last pair represents special characters. Interestingly, the Backspace pair was the most common digraph for eight of the ten users (and second most common for the other two).

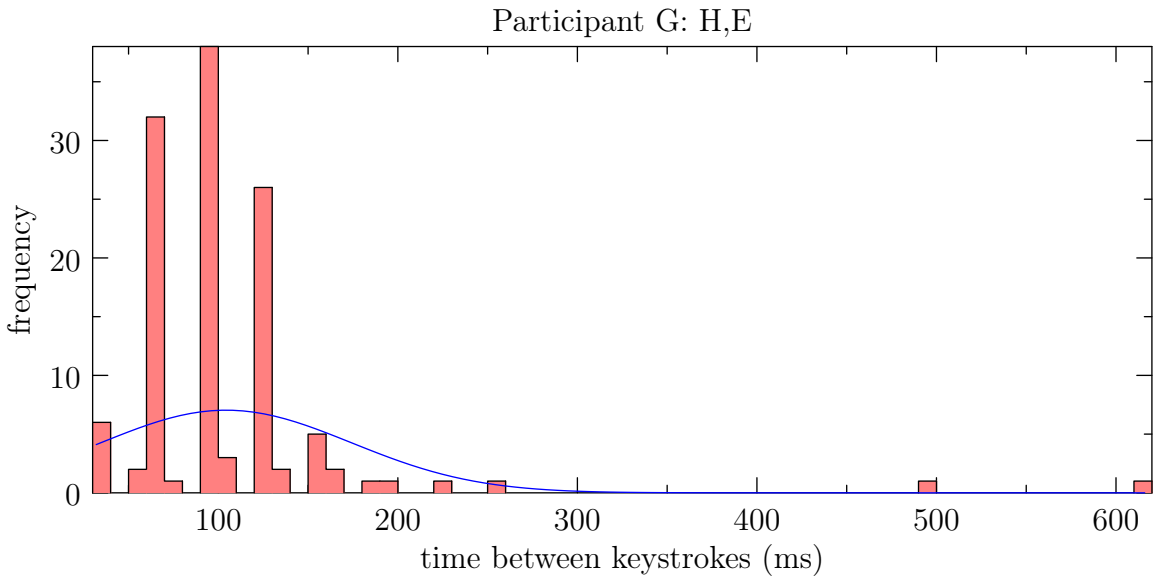



Figure 4.1: Gaussian test, Participant G: H,E

In order to test the normality of an inter-keystroke timing distribution, I followed Song et al.'s example and plotted a histogram overlaid with a normal distribution. I started with the H-E keypair (see Figure 4.1).

While the histogram seems to approximate a Gaussian function (with two noticeable outliers), it also seems to show anomalous groupings, concentrated and isolated from others. Figures 4.2 and 4.3 show similar results for the E-R and L-L keypairs: punctuated groupings that together form a roughly normal distribution (outliers aside).

Figure 4.4 shows the H-E keypair for participant A. At first glimpse, it seems that this punctuation does not occur with all users. However, if the histogram is drawn with smaller bins, each representing a 1ms range instead of 10ms, we see that the punctuation is still observable (Figure 4.5). Further experiments with other character pairs from participant A confirm this pattern; there is still consistent punctuation, just on a smaller time scale.

I hypothesize that these punctuated groupings are a result of the way the data was collected. Gunetti and Picardi [15] indicated in Section 4 of their paper that a client-side JavaScript was used to collect their data, and that their timer had a clock tick of 10ms.

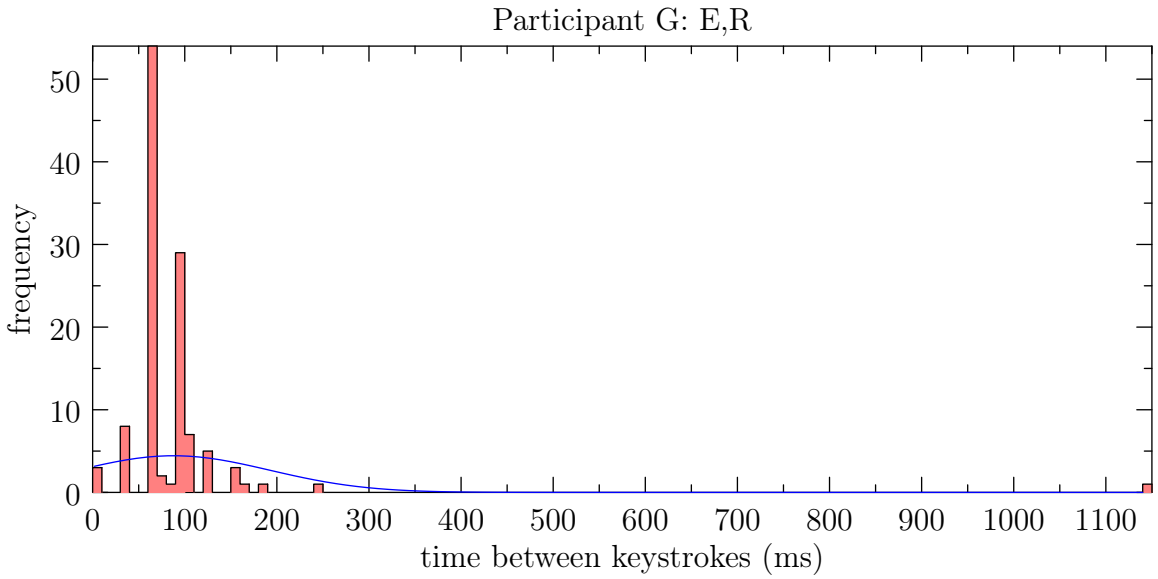



Figure 4.2: Gaussian test, Participant G: E,R

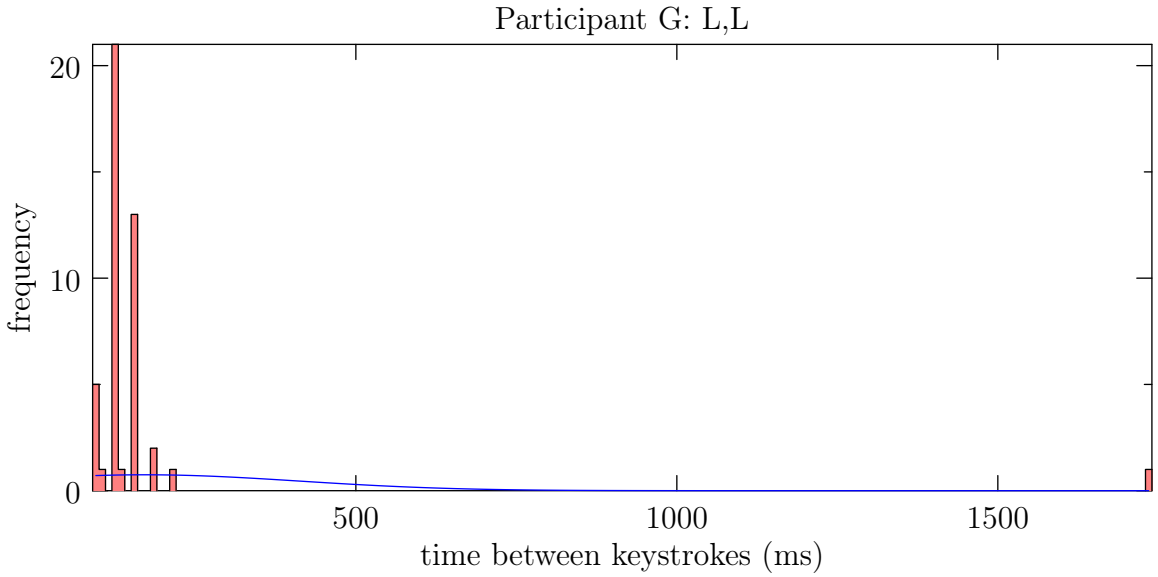



Figure 4.3: Gaussian test, Participant G: L,L

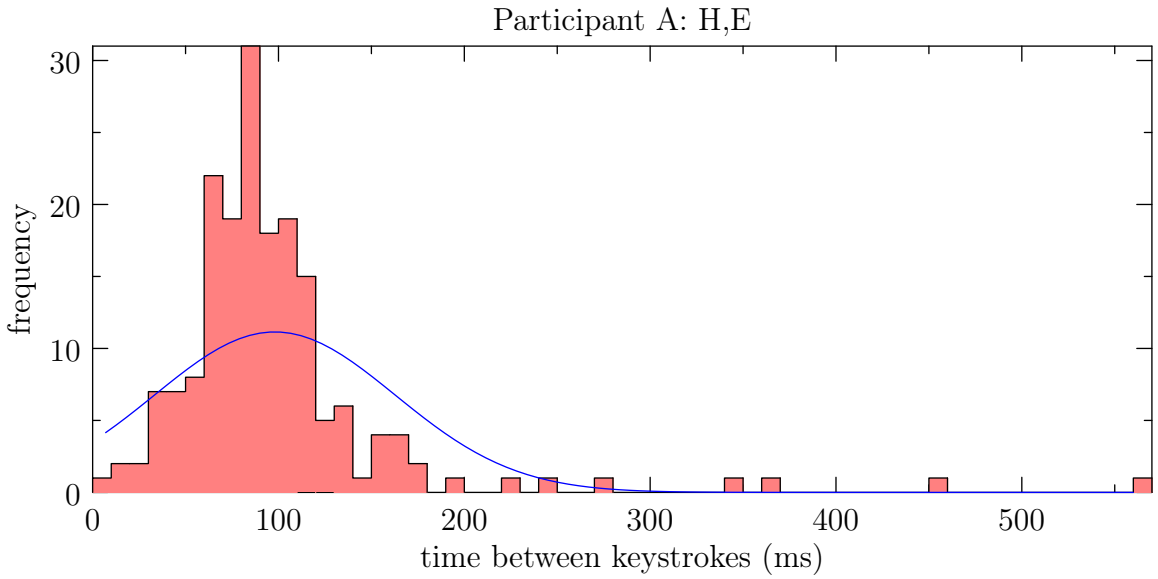



Figure 4.4: Gaussian test, Participant A: H,E (10ms bins)

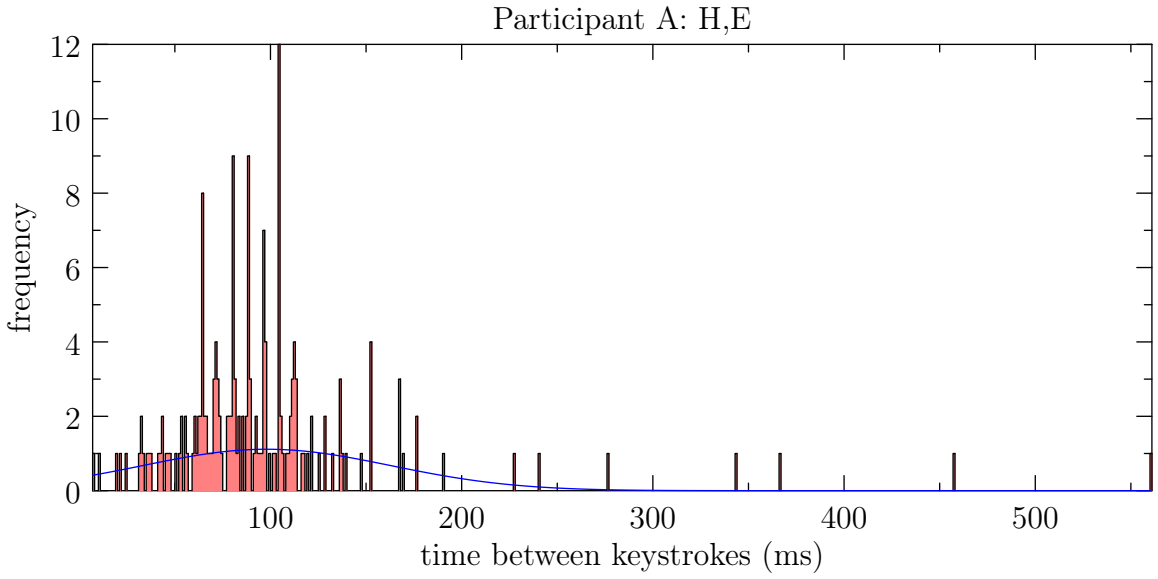



Figure 4.5: Gaussian test, Participant A: H,E (1ms bins)

While Hempstalk did not give as much detail about data collection, the experiment was performed in a browser. Therefore, it seems likely (but cannot be confirmed at this point) that the groupings are the result of some timing artifact from the data collection process. I will from now on assume that these groupings are a result of an essentially low-resolution timing system. The actual timings behind them are assumed to have a more or less regular spread between spikes. This should have little impact on the results of any experiments, except to decrease the precision of any known timing-related quantities. Indeed, this precision loss may be significant in some cases; participant G's spikes are each about 30ms apart.

Future researchers seeking to build similar data sets should be aware of this potential limitation of web-based data collection.

### 4.2.2 Outliers

The keypairs observed in Section 4.2.1 all appear to match normal distributions with the exception of a few outliers. The distribution for participant G's H-E keypair, for example, has a mean time of 104ms and a standard deviation of 69.7ms, with 123 total values. There are only four values above 200ms, but two of them are very distant outliers — 496ms and 616ms. Their separation suggests that these two values may be anomalous, perhaps representing interruptions in thought or focus as the user was typing. Participant G's R-E and L-L keypairs have more significant outliers, each above one second, while all other values are under 300ms. Participant A's H-E pair (Figure 4.4, however, does not show such severe outliers; rather, the values are scattered more smoothly up to above 500ms. This may be due in part to the larger sample size, 181 instead of 123 values.

These anomalies suggest that, in free-text, a set of inter-keystroke timings is not characterized by a true normal distribution. Furthermore, the impossibility of negative values disqualifies the Gaussian function from being a perfect model. However, this does

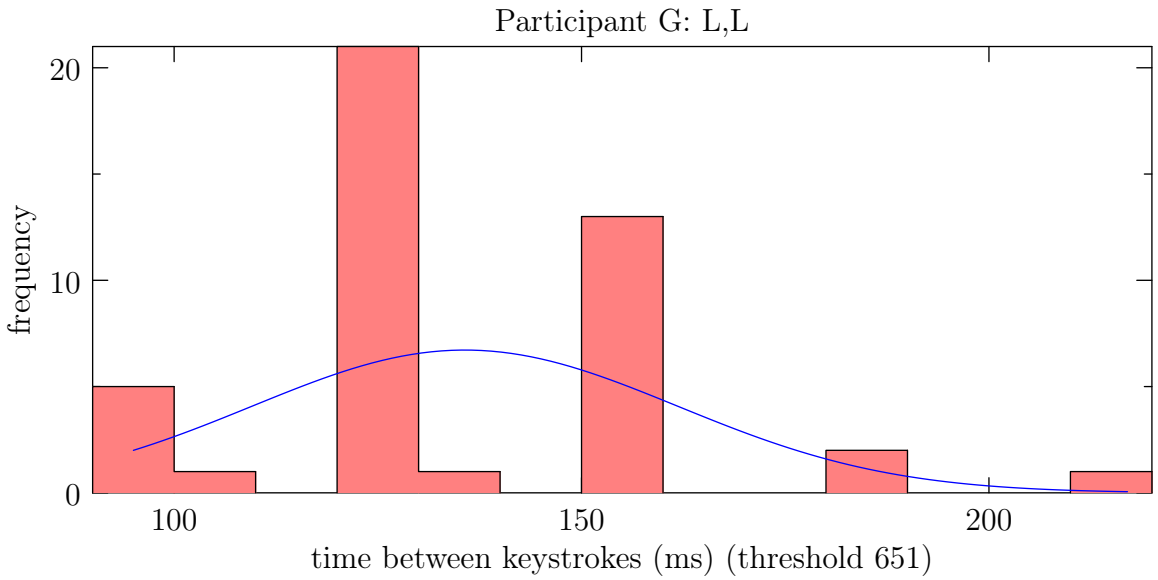



Figure 4.6: Gaussian test, Participant G: L,L ($3\sigma$ threshold)

not necessarily mean that the model is not useful. The timings still approximate the Gaussian function, outliers aside, and removing the outliers may improve the fit.

Figure 4.6 shows participant G's L-L keypair, this time with the one outlying value removed (compare Figure 4.3). The threshold was set at 890ms (three standard deviations above the mean), and all values greater than the threshold (only one in this case) were removed. Figure 4.6 shows a tighter fit than Figure 4.3, and a tight-fitting distribution is what will allow keystroke pairs to be distinguished. In an analysis system, it may be best to remove outliers and then handle larger values differently. For example, if an attacker sees an inter-keystroke timing of 900ms, they will not be able to say with much confidence that it is more likely to be one letter-letter pair than another — that is, unless the higher values apply to some keypairs more than others. This, however, is a difficult question to test, since the data set is limited and these anomalies occur very rarely (about once in 50 samples, offhandedly).

I conclude that, for letter-letter pairs, it is best (or at least good enough for current purposes) to model the timings as a normal distribution after excluding outliers. Here, outliers are defined as any values more than three standard deviations above or below

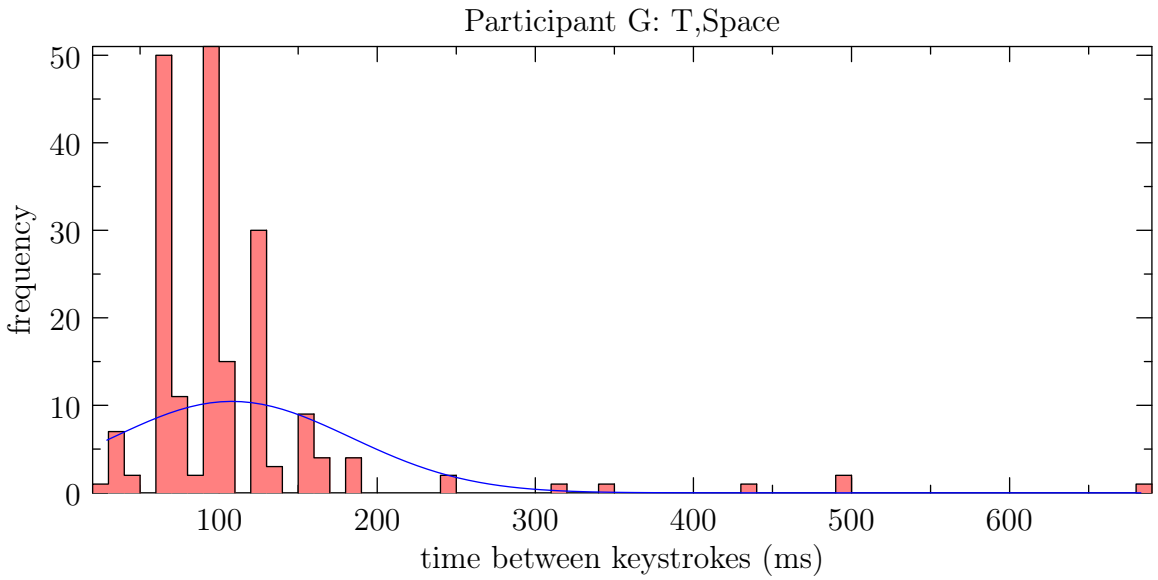



Figure 4.7: Gaussian test, Participant G: T,Space

the mean. By the empirical rule, this is expected to exclude only up to about 0.3% of actual values (probably less, since there are no negative values to exclude, and since the outliers will shift the mean slightly to the right).

### 4.2.3 Spaces and Backspaces

Figure 4.7 shows a histogram plot (with outliers) of user G's T-Space (end of word) keypair. The average time is 108ms with a standard deviation of 75.2ms. After removing outliers, the recalculated mean is 98ms with a standard deviation of 39.6ms. Five of 197 values were identified and removed as outliers, compared to two out of 121 for the H-E keypair and one out of 115 for the E-R keypair. As with the letter-letter pairs, most values seem to fit a normal distribution. However, the T-Space keypair has a heavier right tail; that is, there are more values farther above the mean than expected than with the letter-letter pairs previously observed. This difference will be considered in greater detail in Section 4.3, but for now it is sufficient to conclude that most timings fall in a normal distribution.

Figure 4.8 shows user G's Space-T (start of word) keypair, with the three-sigma

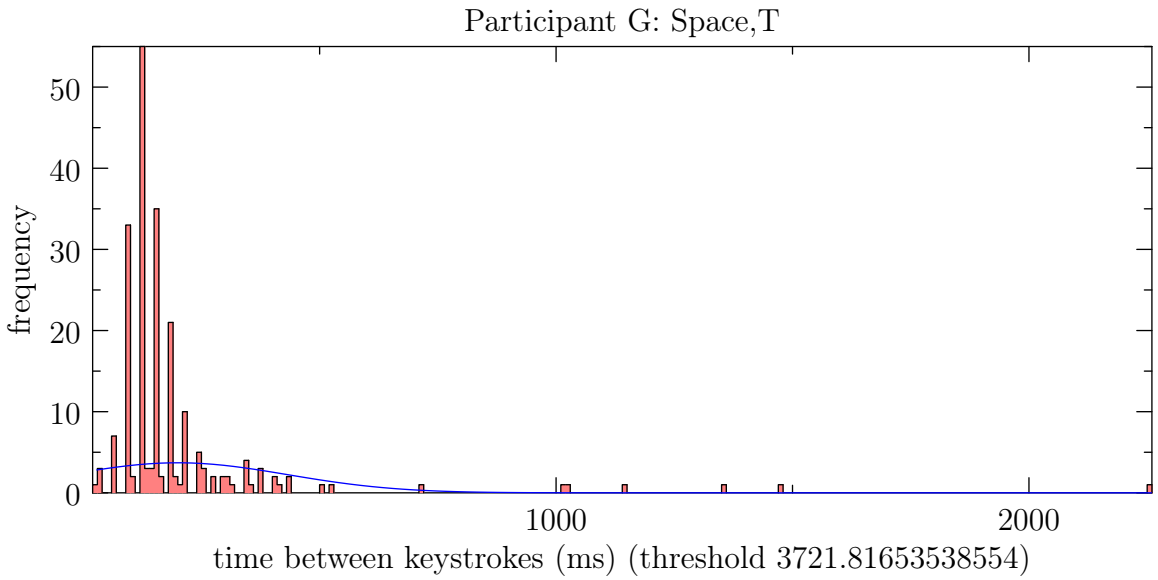



Figure 4.8: Gaussian test, Participant G: Space,T ($3\sigma$ threshold)

threshold applied. There were only two values above the threshold, one of which was over 16 seconds. Even with these two values removed, the graph is still heavily weighted to the right with a large standard deviation (1143ms). Figure 4.9 shows the same keypair with an artificial threshold of 1000ms. The fit is better in this case, but still imperfect. For this keypair and others like it, it may be better to use an alternative distribution, such as a Rayleigh distribution. If not, another outlier test will need to be used and the removed values still accounted for.

Figure 4.10 shows user G's backspace-backspace keypair. The heavy right tail in this histogram is even more significant than in the space-T keypair. Interestingly, rather than a uniform long tail, this histogram seems to show a fairly normal distribution below 500ms, and another, flatter distribution between about 750 and 1250ms. This demonstrates more definitively that a simple normal distribution will not be sufficient for all keypairs.

Figure 4.10 has outliers beyond three standard deviations removed already, yet does not at all fit a normal distribution. In order to deal with this sort of keypair, it may be best to take a split approach to analysis.

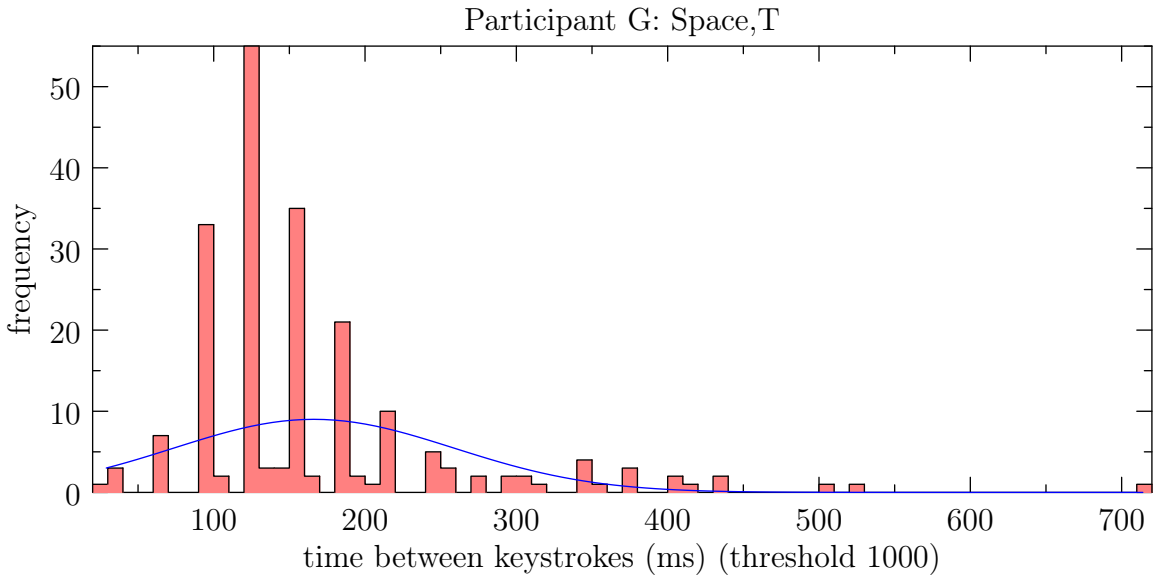



Figure 4.9: Gaussian test, Participant G: Space,T (1000ms threshold)

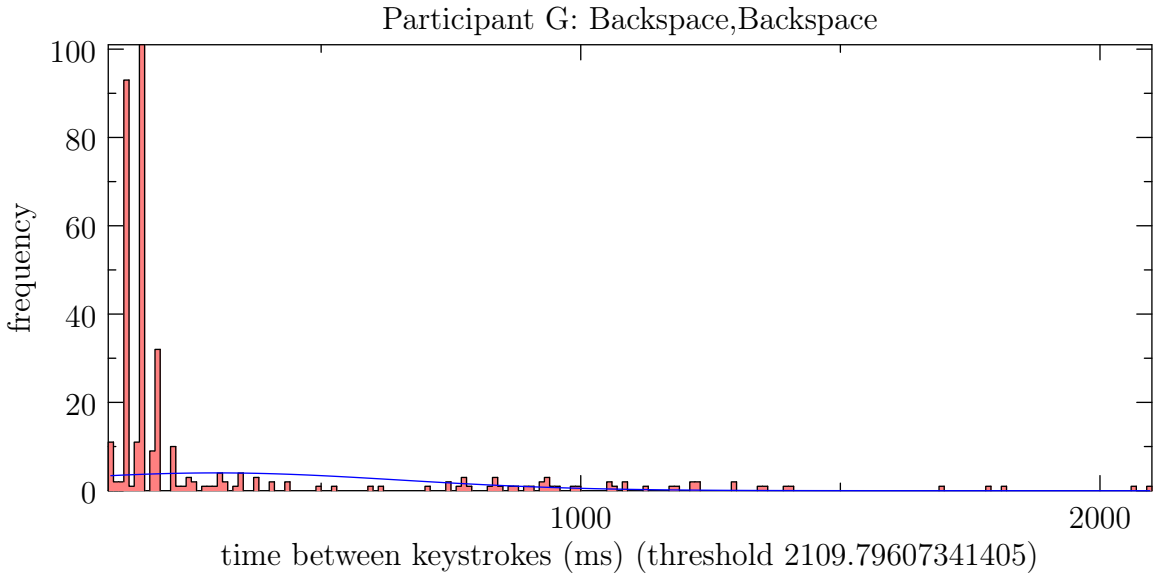



Figure 4.10: Gaussian test, Participant G: Backspace,Backspace (3 std. dev. threshold)

### 4.2.4 A Split-Analysis Approach

In order to deal with certain keypair distributions, I propose a split-analysis approach, in which each keypair is treated as having two separate distributions. The first is the typical distribution, treated as a normal distribution. This first distribution is comprised of all values below a certain threshold, about 300ms for user G. The second "distribution" is comprised of all values above that threshold; it may be considered as one or broken up into discrete bins for analysis. The threshold is chosen such that, for all relevant keypairs, data below that time approximates a normal distribution. This approach is by no means ideal, but will allow for testing of the hypotheses in this thesis.

### 4.2.5 Summary

This section demonstrated that, in a free-text setting, inter-keystroke latencies form a normal distribution with a heavy right tail. Certain keypair distributions have a very heavy right tail (such as backspace-backspace pairs), while others have only a few apparently anomalous values. Thus, a Gaussian distribution alone is insufficient to fully model inter-keystroke timing distributions. We must either find a better distribution, or else consider these higher than usual values separately in a split-analysis approach.

## 4.3 Identifying Word Breaks

This section will document the experiment designed to test the first hypothesis: "Word breaks in free text are not distinct enough to be consistently identified."

For the sake of testing this hypothesis, I will only consider letter and space keypairs. This is an extremely limiting assumption, but if word breaks are shown to be indistinct, this will still be the case for the general problem. If, however, word breaks are shown to be distinct, further work will be required to determine whether or not they stay distinct in the general case. Removing special characters will also create an experiment more

comparable to those performed by Zhang and Wang and Song et. al.

Three classes of character pairs will be considered for analysis: letter-letter, letter-space, and space-letter pairs (a space-letter pair, for example, is any pair in which the first key is a space and the second key is a letter, typically the start of a word). Figure 6 in Zhang and Wang's 2009 paper shows a significant separation between space-letter and letter-letter keypairs, with the following conclusion: "typically the key pair involving SPACE incurs longer inter-keystroke latency than other pairs." [38, p. 8] In the illustration (see Figure 5.1), the vast majority of space-to-letter transitions were above 400ms, while extremely few letter-to-letter transitions were above 400ms. The first step will be to reproduce this experiment with free text, using Hempstalk's `sm-150` data set.

### 4.3.1 Reproducing Zhang and Wang's Experiment

Zhang and Wang collapsed all space-letter pairs and considered them as one. Figure 4.11 tests the validity of this approach for space-letter pairs from `sm-150`'s participant G. The lines represent each type of space-letter keypair (space-A, space-B, etc.) with over 30 samples, after removing values above 300ms. The red line represents all those space-letter keypairs considered as one. This provides a visual verification of the reasonableness of combining space- letter pairs. Rather than considering space-A and space-B as separate keypairs, we can think of them both as simply space-letter pairs. If these pairs can be distinguished from others, word breaks can be positively identified. A similar pattern was observed for letter-space pairs, but not for letter-letter pairs.

Figure 4.12 shows an overlay of letter-letter pairs and the combined space-letter and letter-space graphs for participant G. Contrary to the results in Zhang and Wang's paper, the space-letter graph has a significant area of intersection with other pairs. In fact, there appears to be little to no statistical difference between space-letter, letter-space, and letter-letter pairs when considering only values below 300ms (for this user). This result stands in stark contrast to Zhang and Wang's result. I hypothesize that the

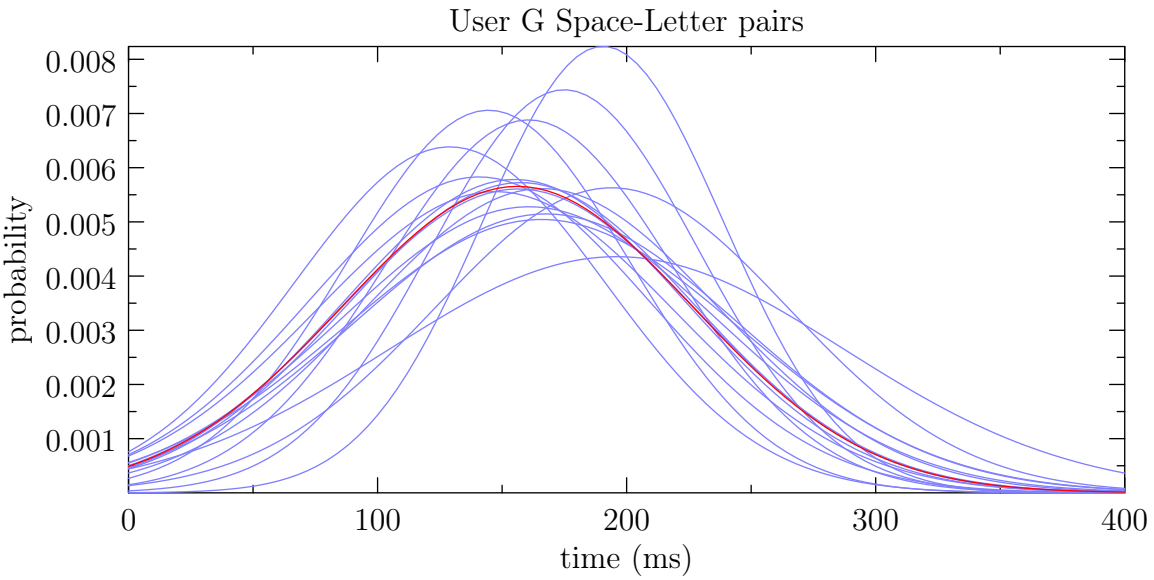



Figure 4.11: Combining space-letter pairs (participant G)

difference is due primarily to the nature of the data sets involved. It may be that, in the scripted laboratory data used by Zhang and Wang, users typing an arbitrary string of words take a slightly longer pause between each word to consider the next one. Whether this is truly the case is difficult to verify.

In any case, Zhang and Wang's result that space-letter pairs consistently incur a longer than average latency, it seems, does not hold for free text. Similar experiments were carried out for the other nine participants in `sm-150`. While the details of the graphs differed, none of them demonstrated the kind of separation between space-letter and letter-letter pairs seen in Zhang and Wang's 2009 paper.

### 4.3.2 Anomaly Analysis

While space-letter pairs may not be easily distinguishable in general, it may be worthwhile to observe the higher latency values. In order to facilitate the use of Gaussian functions in the section above, it was necessary to truncate the data and consider timings only below a certain threshold. This was done precisely because data above that threshold does not fit a normal distribution. Far from being a mere inconvenience, this

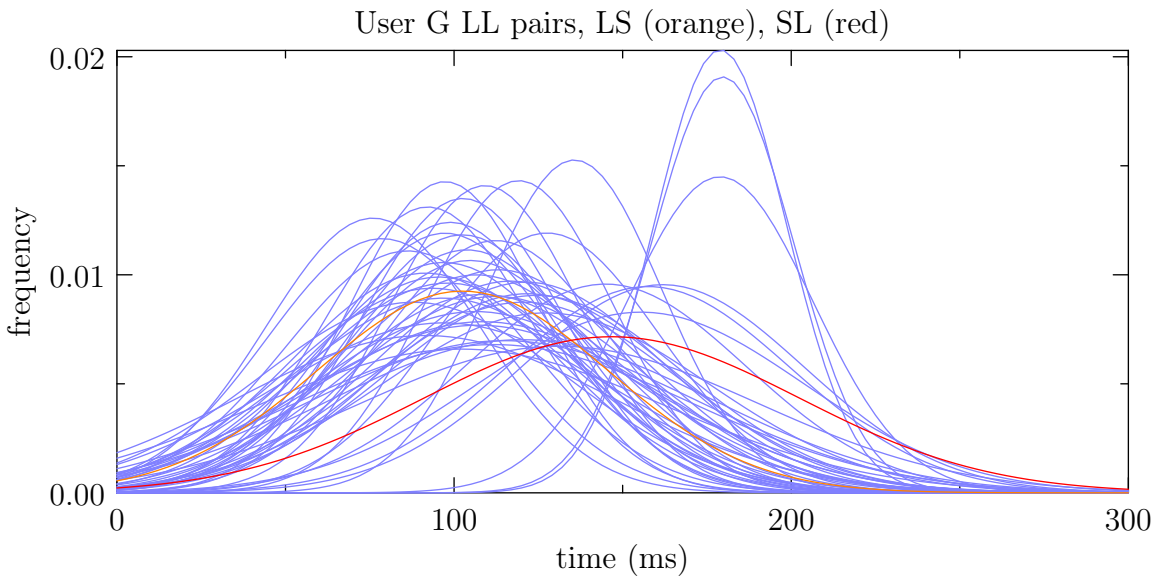



Figure 4.12: Letter-Letter (LL), Letter-Space (LS), and Space-Letter Pairs (SL)

| LL Pairs | LS Pairs | SL Pairs |
|---|---|---|
| 66.13% (±2.88) | 17.01% (±1.80) | 16.86% (±1.79) |

Table 4.1: Average frequency of letter-letter (LL), letter-space (LS), and space-letter (SL) pairs for all users

abnormality may be the key to identifying certain inter-keystroke timings.

In considering these higher-than-normal values, we will break them into discrete bins for analysis. Then, a simple percentage calculation will be used to determine the distinctness of space-letter or letter-space pairs.

Before moving on to this test, we must first establish a baseline probability of any given letter and/or space pair being a letter-letter, letter-space, or space-letter pair. Table 4.1 gives the average frequencies of these character classes. The first column indicates that, among character pairs consisting only of letters and one or zero spaces, 66 ± 2% of them are letter-letter pairs. Likewise, the second and third columns indicate that letter-space and space-letter pairs split the remainder with about 17% each. Interestingly, there is very little variation among users. These numbers are a mere reflection of the frequency of spaces in English emails.

Table 4.2 shows the probabilities for participant G when only certain subsets of

| Bin (ms) | LL | LS | SL |
|---|---|---|---|
| 0 – 300 | 67.06% | 17.12% | 15.82% |
| 300 – 400 | 54.44% | 8.89% | 36.67% |
| 400 – 500 | 51.49% | 12.87% | 35.64% |
| 500 – 600 | 42.86% | 5.71% | 51.43% |
| 600 – 700 | 47.83% | 8.70% | 43.48% |
| 700 – 1000 | 44.44% | 8.33% | 47.22% |
| 1000 – 2000 | 20.45% | 4.55% | 75.00% |
| 2000+ | 34.48% | 3.45% | 62.07% |

Table 4.2: Participant G: Frequency of LL, LS, SL pairs by timing range

the data are considered. The 0–300ms bin reflects the "typical" values studied in Section 4.3.1; word breaks are indistinct. The first row indicates that, among participant G's space and/or letter digraphs with an inter-keystroke latency of 300ms or less, 67% were letter-letter pairs, 17% were letter-space pairs, and 16% were space-letter pairs. However, in the very next bin, 300–400ms, the frequency of space-letter pairs jumps from 15.8% to 36.7%. The 500–600ms bin has a 51.4% space-letter frequency. The probability reaches 75% in the 1000–2000ms range, before decreasing in the 2000+ range.

While the space-letter pair becomes more prominent in the higher timing ranges, the letter-letter and letter-space pairs decrease at about the same speed. In both the 0–300ms and the 1000–2000ms bins, letter-letter pairs make up about four-fifths of the remainder after taking away space-letters, and letter-space pairs make up the remaining one-fifth.

Table 4.3 lists the frequencies of the three categories considering only values over 2000ms for each user in the `sm-150` data set. User D has the lowest space-letter frequency, 44%. Users C, G, and J have space-letter frequencies of about 60%. The remaining six users have space-letter frequencies of 70% or more. The letter-space pairs appear to be even less consistent. With most users, letter-space pairs remain infrequent compared to letter-letters, even in the 2000+ range. With others (A, C, and H), letter-space pairs are just as common or even more common than letter-letters. In either case, they remain in the minority compared to space-letters.

| User | SL | LS | LL |
|---|---|---|---|
| A | 91.11% | 5.56% | 3.33% |
| B | 70.59% | 11.76% | 17.65% |
| C | 60.71% | 28.57% | 10.71% |
| D | 44.29% | 8.57% | 47.14% |
| E | 80.17% | 5.17% | 14.66% |
| F | 76.47% | 7.35% | 16.18% |
| G | 62.07% | 3.45% | 34.48% |
| H | 91.43% | 4.29% | 4.29% |
| I | 70.59% | 13.73% | 15.69% |
| J | 60.12% | 13.50% | 26.38% |

Table 4.3: Frequencies for keypairs in the 2000ms+ range.

This is a very interesting result from an analysis standpoint; a small subset of space-letter pairs can be identified with a high degree of confidence (70% or more for six of ten users).

Therefore, if an analyst has access to plaintext examples, they could discern these frequencies with precision and make educated guesses about certain keypairs in a timing stream. However, if an analyst has no known plaintext, the exact space-letter frequencies will be unknown, though they could say that, above some threshold, space-letter pairs are likely to have at least a certain frequency. Based on the current data, they could say that, for nine out of ten users, each space and/or letter keypair with an inter-keystroke latency of more than 2000ms has a 60% chance of being a space-letter pair.

It is worth mentioning again a major assumption from the beginning of this section: non-letter and non-space keypairs were omitted from these calculations. This assumption, and the implications of its removal, will be addressed in Section 4.4.

### 4.3.3 Predictive Experiment

In this section, we will evaluate the effectiveness of an attack based on the simple guessing rule: keypairs above 2000ms are space-letter pairs.

Table 4.4 shows the results of this experiment. Type I (false positive) and Type II

| User | True Positives | True Negatives | False Positives | False Negatives |
|---|---|---|---|---|
| A | 82 | 8829 | 8 | 1770 |
| B | 36 | 6640 | 15 | 1210 |
| C | 17 | 8309 | 11 | 1777 |
| D | 31 | 6248 | 39 | 1084 |
| E | 93 | 5174 | 23 | 885 |
| F | 52 | 8187 | 16 | 1702 |
| G | 18 | 6418 | 11 | 1340 |
| H | 64 | 7880 | 6 | 1687 |
| I | 36 | 4111 | 15 | 864 |
| J | 196 | 5623 | 130 | 875 |

Table 4.4: Predictive experiment for all participants.

(false negative) errors are both shown. For example, the attack identified 90 space-letter keypairs for participant A — 90 positives. Of those 90, 82 were correct (true positives), and eight were incorrect (false positives). Also for participant A, 10599 keypairs were *not* identified as space-letter keypairs (negatives). Of those 10599, 8829 were correct (true negatives), but 1770 actually were space-letter keypairs (false negatives). All users have a significant quantity of false negatives, but relatively few false positives.

In order to test the significance, or insignificance, of these results, we will use a contingency table chi-squared test.

The null hypothesis is: "A keypair with an inter-keystroke latency of more than 2000ms is no more or less likely to be a space-letter pair than any other keypair."

If the data in Table 4.4 contradicts the null hypothesis, we will know that the test is meaningful.

The calculations will be described in detail for participant A and summarized for others.

Assuming the null hypothesis, all keypairs should have the same probability of being a space-letter or non-space-letter pair, regardless of whether or not the timing is greater than 2000ms. Participant A's test results from Table 4.4 are given again in Table 4.5. The table shows that, if a keypair timing is greater than 2000ms (first column), it is more likely to be a space-letter pair than not. However, if the timing is not greater than

| | >2000ms | <=2000ms |
|---|---|---|
| SL | 82 | 1770 |
| Not SL | 8 | 8829 |

Table 4.5: Contingency table for participant A

| | >2000ms | <=2000ms |
|---|---|---|
| SL | 15.597 | 1836.8 |
| Not SL | 74.403 | 8762.2 |

Table 4.6: Expected contingency table for participant A

2000ms, it is more likely to be something other than a space-letter pair. In the chi-squared test, we essentially test how improbable these results are, if they only happened by chance.

Based on the data in Table 4.5, participant A's sample is made up of 1852 space-letter pairs and 8837 other pairs; 17.33% space-letter, 82.67% other. There are a total of 90 greater-than-2000ms keypairs. Therefore, the expected number of greater-than-2000ms keypairs which are also space-letter pairs is $0.1733 \times 90 = 15.597$. Likewise, the expected number of non-space-letter pairs in the greater than 2000ms range is $0.8267 \times 90 = 74.403$. In this way, Table 4.6 is filled up with the expected values.

To calculate the chi-squared statistic, we add the variances between each cell of Tables 4.5 and 4.6. The variance is: $(expected - actual)^2/expected$. So our chi-squared statistic is given by:

$$\frac{(15.597-82)^2}{15.597} + \frac{(1836.8-1770)^2}{1836.8} + \frac{(74.403-8)^2}{74.403} + \frac{(8762.2-8829)^2}{8762.2} = 344.9$$

For any contingency table, the degrees of freedom is given by $(r-1)(c-1)$. This table is two rows by two columns, so there is one degree of freedom. Using a chi-squared distribution table, the minimum value for a probability of .005 is 7.879. Our value of nearly 345 far exceeds this threshold, so there is a very small chance that these values came about by chance. Indeed, the probability associated with a statistic of 345 is a

| User | Chi-Squared Statistic | Chance null hypothesis is true |
|---|---|---|
| A | 344.98 | 5.29E-77 |
| B | 116.13 | 4.46E-27 |
| C | 35.54 | 2.50E-09 |
| D | 47.17 | 6.52E-12 |
| E | 367.08 | 8.09E-82 |
| F | 163.42 | 2.03E-37 |
| G | 40.29 | 2.21E-10 |
| H | 254.52 | 2.68E-57 |
| I | 97.27 | 6.04E-23 |
| J | 510.72 | 4.41E-113 |

Table 4.7: Contingency table chi-squared test results for all participants

negligible 5.5E-77.

The chance of getting these results after assuming the null hypothesis is insignificant. Therefore, we can reject the null hypothesis with certainty, for participant A. Table 4.7 shows the results of similar calculations for all users. In each case, the chi-squared statistic is well above 7.879.

Therefore, it is clear that there is an association between space-letter pairs and extreme inter-keystroke latencies for all users in this data set.

### 4.3.4 Summary

Recall the hypothesis for this section (4.3): "Word breaks in free text are not distinct enough to be consistently identified." Our hypothesis from Section 4.3 was: Section 4.3.1 challenged the results from Zhang and Wang's 2009 paper [38]. In their paper, they asserted that word breaks were easily and consistently identifiable, that the vast majority of word breaks could be identified with a very high degree of confidence. We demonstrated that this is not the case for free-text data (at least not for the `sm-150` data set).

This suggests that our hypothesis holds true.

However, Section 4.3.3 demonstrates that there is at least some association between

space-letter pairs (word breaks) and inter-keystroke timings, and that this association could be exploited by an attacker. Keypairs with extreme inter-keystroke timings, for most users, tend to be space-letter pairs more often than not (participant D is the exception). However, the test has a very high false-negative rate, restricting its usefulness. The positive indications are statistically helpful, while the negative indications are almost meaningless in comparison.

In short, not all word breaks in free text are distinct enough to be identified, but some may indeed be identified with a degree of certainty (for most users).

## 4.4 Impact of Increased Noise in Free-Text Data

This section will document the experiment designed to test the second hypothesis: "The larger alphabet and less controlled data collection in a free text environment make character pairs harder to distinguish."

Section 4.3.3 already demonstrated that some character pairs (space letter pairs, in particular) are harder to distinguish in a free text data set, probably due to the less controlled environment. This section will demonstrate the impact of an increased alphabet size on keystroke analysis.

Recall that, in Section 4.3.3, non-space/letter key pairs were removed from the data set. This amounted to an artificial experiment, but one that more closely resembled the experiment considered by Zhang and Wang's [38]. In Table 4.3, it was observed that, for most users, keystroke pairs separated by more than 2000ms were much more likely to be space-letter pairs than anything else. That is to say, for most users, the longer the inter-keystroke latency, the more likely it is that the keypair is a word break.

Table 4.8 repeats this experiment with all types of keypairs included. The last column, "Other," includes all keystroke pairs that involve something besides a space or letter. These include numbers, punctuation, backspaces, and everything else.

While space-letter keypairs in the table are still more common than letter-letter

| User | SL | LS | LL | Other |
|---|---|---|---|---|
| A | 40.80% | 1.49% | 2.49% | 55.22% |
| B | 11.65% | 2.91% | 1.94% | 83.50% |
| C | 6.12% | 1.08% | 2.88% | 89.93% |
| D | 7.31% | 7.78% | 1.42% | 83.49% |
| E | 21.93% | 4.01% | 1.42% | 72.64% |
| F | 23.21% | 4.91% | 2.23% | 69.64% |
| G | 12.41% | 6.90% | 0.69% | 80.00% |
| H | 15.96% | 0.75% | 0.75% | 82.54% |
| I | 29.27% | 6.50% | 5.69% | 58.54% |
| J | 26.49% | 11.62% | 5.95% | 55.95% |

Table 4.8: Frequencies for keypairs in the 2000ms+ range, including all keypair types.

pairs (for most users), the Other category is much, much more common. For five out of ten users, the Other category takes up 80% or more of all keypairs in the table. The otherwise distinct space-letter pairs are drowned out by the noise. If an attacker is looking for word breaks, they will not be able to make any reliable guesses based only on each digraph's inter-keystroke latency.

It is therefore clear that the increased alphabet size and less controlled data collection in a free text data set can make a predictive attack much more difficult.

# Chapter 5: Conclusion

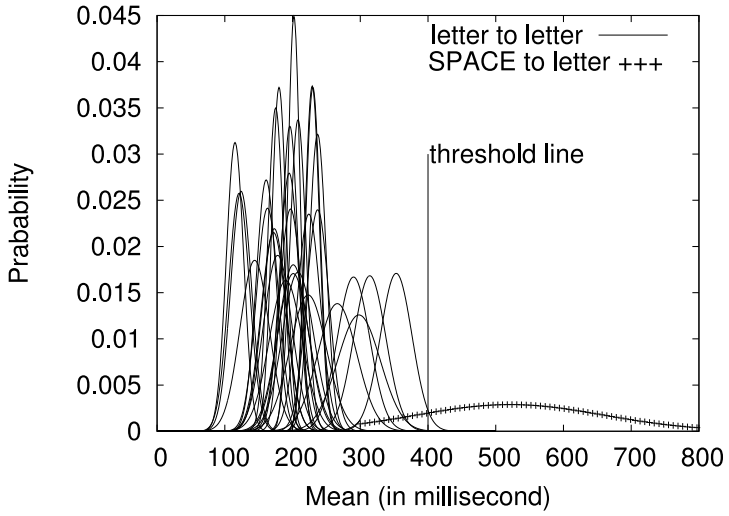
## 5.1 Summary

Zhang and Wang [38] postulated that inter-keystroke timings could be used to distinguish between letter-letter pairs and space-letter pairs. That is, they postulated that the timing between a space key and the start of a word could be distinguished from the timings within a word. This postulation was backed up by data from their experiment.

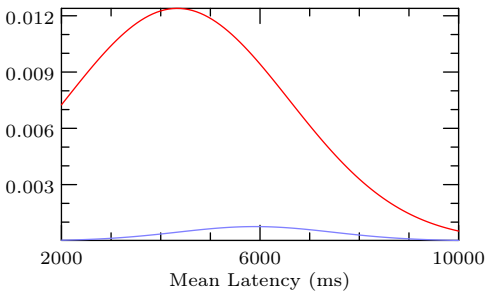
Our research shows that this holds in a free-text environment, but only when the space-letter timing characteristic forms a distribution with an anomalous mode or shape. In Zhang and Wang's paper, the space-letter timing distribution was normal with a very distinct mode compared to the distributions of other character pairs. In the free text `sm-150` data set, the space-letter mode is not separated enough to make a strong distinction. However, the shape of the distribution is unusual, with a large number of outliers. These outliers give the space-letter timing distribution a bimodal shape, making a subset of space-letter pairs distinct and easily separable from other pairs.

Figure 5.1 illustrates this principle, comparing word breaks in the Zhang and Wang data set to those in the Hempstalk data set. In Figure 5.1a, from Figure 6 in Zhang and Wang's 2009 paper, the space-to-letter distribution is significantly separated from all letter-to-letter distributions. Therefore, as recorded in their paper, space-to-letter transitions are easy to find. Similarly, Figure 5.1b shows word breaks in the 2000–10000 ms range for participant A, using the `sm-150` data set. In this range, there are significantly more space-letter transitions than letter-letters, making them easy to spot. In contrast, Figure 5.1c shows that participant D's space-letter distribution is not significantly different than the letter-letter distribution. While the shapes are not identical, the significant overlap makes word breaks identifiable with only about a 50% accuracy.

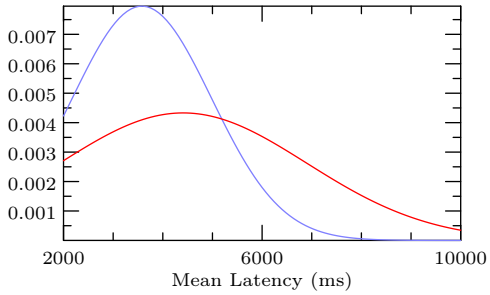
To summarize, space-letter pairs are only distinct from other pairs if the timing




(a) Figure 6 from Zhang and Wang's 2009 paper [38]




(b) `sm-150` Participant A space-letters (red), letter-letters (blue)




(c) `sm-150` Participant D space-letters (red), letter-letters (blue)

Figure 5.1: Comparison of word breaks in fixed and free-text data sets.

distributions themselves are distinct. How distinct the distribution is depends both on the way in which data was collected and on the user. In Zhang and Wang's scripted laboratory setting, the distributions were very distinct. In Hempstalk's free text data set, `sm-150`, the distinctness of the space-letter distribution varies from user to user. This suggests that, in general, the attack will be less effective on free-text data, and that its effectiveness will vary from user to user.

## 5.2 Hypotheses Revisited

This section reviews the hypotheses surrounding the three questions first posed in Chapter 1 of this thesis.

Hypothesis #1:

Inter-keystroke timing distributions from free text form a normal distribution.

Resolution: Yes, inter-keystroke timing distributions from free text form a normal distribution, for the most part.

This hypothesis was tested in Section 4.2. While it was verified that most inter-keystroke timing values fit into a normal distribution, it was also found that there is a small portion of values that fall very far outside the normal distribution. This results in a relatively normal distribution with a heavy right tail. Furthermore, the number of these outliers depends in part on the keypair being observed. A split-analysis approach was also proposed to deal with these outliers. Below some threshold, keypairs could be treated as having a normal distribution. Above the threshold, another approach may be used.

While this split-analysis approach was sufficient for the purposes of this thesis, there may be other non-Gaussian distributions which prove more useful in future research.

Hypothesis #2:

Word breaks in free text are not distinct enough to be consistently identified.

Resolution: In a free text environment, the majority of space-letter pairs blend in with

letter-letter pairs. However, a subset of those space-letter pairs stand out, and are, for most users, identifiable.

Zhang and Wang demonstrated a separation, in their data set, between space-letter pairs and other types of keypairs [38]. Specifically, inter-keystroke latencies between space and letter keys were typically above 400ms, while other pairs' latencies were typically below 400ms. As a result, they concluded that space-letter pairs could be used to separate different words, making analysis essentially a one-word-at-a-time problem. I hypothesized that this would not hold for free text.

In accordance with this hypothesis, there was no strong boundary between space-letter and letter-letter pairs in Hempstalk's `sm-150` data set. This left the majority of word breaks unidentifiable, at least using the approach described in Zhang and Wang's paper. However, it was found that space-letter pairs tended to have a higher number of outliers than letter-letter pairs. Section 4.3.3 demonstrated that, for at least some users, a subset of word breaks can be identified with a measure of confidence. This resulted in the clarification on Zhang and Wang's result: In a free text environment, the majority of space-letter pairs blend in with letter-letter pairs. However, a subset of those space-letter pairs stand out, and are, for most users, identifiable.

Hypothesis #3:

The larger alphabet and less controlled data collection in a free text environment make character pairs harder to distinguish.

Resolution: Yes, the larger alphabet and less controlled data collection in a free text environment both make character pairs harder to distinguish.

In addressing the second hypothesis, Section 4.3 also verified that space-letter pairs are less distinct in a free-text environment, due probably to less-controlled data collection. The experiment in Section 4.4 sought to verify that the increased alphabet size, including backspaces, punctuation, arrow keys, etc., also impacts an attacker's ability to identify character pairs. Section 4.3.3's experiment was performed with all non-space

and non-letter pairs filtered out (this allowed for a situation comparable to Zhang and Wang's, where only spaces, letters, and a few numbers were used). To more closely consider an attacker's situation, and to test this hypothesis, Section 4.3.3's experiment was repeated with no characters filtered out. While space-letter pairs still stood out compared to letter-letter pairs, they were drowned out by the noise. The confidence with which they could be identified was drastically decreased, and their predictive value was almost entirely removed. This confirmed, along with Section 4.3, that increased alphabet size and less controlled environment both serve to make character pairs harder to distinguish in a free text data set.

## 5.3 Questions Revisited

Several questions were posed in Section 3.2 that have yet to be addressed.

Question #1: Does shared training data fail for users who type differently?

While not directly addressed, the predictive experiment on word breaks showed varying levels of success for different users. This suggests that training data for some users may fail to apply for others.

Question #2: Can training data be combined between users to strengthen the attack?

This question was not directly addressed by any experiments in this thesis, but the experiments performed do demonstrate that keypair frequencies can vary a great deal between users. Something that is a strong indicator for one user may not apply to another user at all. Consider users A and D in Table 4.3. For user A, space-letter keypairs are much more common than letter-letter pairs. With user D, however, there is virtually no difference.

This suggests that there will be challenges involved in combining data, if it can be done at all. Attempts at combining keystroke data are left to future research.

Question #3: Can an individual be classified based only on their timing characteristics, without plaintext?

Much of the research on keystroke timing analysis is aimed toward user-authentication and identification. This suggests that users can indeed be classified and distinguished. As far as I know, however, no literature documents an attempt to classify unknown users for the sake of a keystroke timing attack.

Question #4: How useful are keystroke durations in key sequence-inference?

Terence Sim and Rajkumar Janakiraman [27, p. 4] found that durations do indeed have distinguishing value for user identification and authentication. Therefore, it seems likely, but has yet to be demonstrated, that keystroke durations hold some value for an attacker. Exactly how much value they hold, if any, is unclear.

Question #5: In a sequence of key down and key up events, is it possible to determine which are up and which are down?

I hypothesize that it is possible, though not trivial, to determine the difference between most up and down key events based purely on timing. This question has not been addressed at all, to my knowledge. If it is not possible, it may upset keystroke timing attacks in certain situations (recall that Song et al. [28] attacked SSH, an application in which only key down events are sent over the network).

Question #6: Can an attack against free text be improved by looking at words and sentences as a whole (e.g., the commonality of words in typical sentences)?

Terence Sim and Rajkumar Janakiraman have already demonstrated that the context of a digraph does affect its timing [27] [17]. Sim and Janakiraman did not attempt a keystroke timing attack, but there is little reason to believe that an attack would not be helped if it could take context into account.

Question #7: Are high-order HMMs feasible and effective in key sequence-inference?

Zhang and Wang [38] were confident that a high-order HMM could improve their attack. I see no reason to disagree, though it has yet to be attempted in the literature. Whether or not high-order HMMs are computationally feasible or worthwhile in this application, however, is another question — I make no conjecture here.

Question #8: Are there other methods that could improve on HMMs by taking into account the context of digraphs, or longer sequences as a whole?

I suspect that there are other methods besides high-order HMMs that could take advantage of the larger patterns in English text, but which existing tools apply here, if any, is an open question.

## 5.4 Future Research

In light of the results summarized in Sections 5.1 and 5.2 and the few unanswered questions listed in Section 5.3, several clear research directions can be seen.

Section 4.2 demonstrated that a Gaussian distribution alone is insufficient to fully model keypair timing distributions. The split-analysis approach proposed in Section 4.2.4 was sufficient for the experiments in this thesis, but it may not be ideal. Better results may be found using non-Gaussian models or entirely non-parametric approaches. In particular, a k-nearest-neighbor algorithm or classification tree may yield interesting results.

The predictive experiment from Section 4.3 demonstrates that certain information is present in free text data that is not present in more controlled data gained from a laboratory experiment: Certain space-letter pairs exhibit unusual and useful timing behavior. It may be worthwhile to investigate similar patterns in other types of keypairs.

Beyond this, it would be worthwhile to investigate the broader context of digraphs. It seems clear that there is data in the larger context of a keypair that may help an attacker. However, the best way to utilize this data in an attack is not clear. High-order HMMs are one direction, but there may be many more. Considering the wider context of inter-keystroke timings may be the key to increasing this attack's effectiveness.

The usefulness of keystroke durations may be investigated as part of the larger context question, or as a separate problem, but I don't expect it to be as fruitful of a pursuit in itself.

The question of classifying and combining users' keystroke timing data may also be an important component of a successful attack. If users do tend to fall into distinct categories of behavior, it may be possible to attack users without any plaintext samples. This would greatly increase the attack's potential usefulness.

The distinguishing of up key events from down key events is a separate problem. It may be necessary for some attack scenarios, and is a very open problem in the literature.

The concept of a keystroke timing attack is attractive for its potential effectiveness and breadth of application, but it is by no means a trivial problem. Another approach would be to prove, perhaps from a more theoretical standpoint, how limited the attack is. If an upper bound can be placed on its usefulness, it may be concluded that this timing side-channel is not a source of significant information loss.

If, however, it is found that the attack can reliably decipher free text, it would leave countless systems open to dangerous and potentially costly attacks.

# Bibliography

[1] Jeffrey D. Allen. An analysis of pressure-based keystroke dynamics algorithms. Master's thesis, Southern Methodist University, Dallas, TX, May 2010.

[2] Eesa Alsolami. An examination of keystroke dynamics for continuous user authentication. PhD dissertation, Queensland University of Technology, 2012.

[3] Salil P. Banerjee and Damon L. Woodard. Biometric authentication and identification using keystroke dynamics: A survey. *Journal of Pattern Recognition Research*, 7:116–139, 2012.

[4] Luciano Bello, Maximiliano Bertacchini, Carlos Benitez, Juan Carlos Pizzoni, and Marcelo Cipriano. Collection and publication of a fixed text keystroke dynamics dataset. In *XVI Congreso Argentino de Ciencias de la Computación*, 2010.

[5] Maximiliano Bertacchini, Carlos Benitez E., and Pablo I. Fierens. User clustering based on keystroke dynamics. In *XVI Congreso Argentino de Ciencias de la Computación*, 2010.

[6] Tai-Hoon Cho. Pattern classification methods for keystroke analysis. In *SICE-ICASE, 2006. International Joint Conference*, pages 3812–3815, Oct. 2006.

[7] Sérgio T. de Magalhães, Kenneth Revett, and Henrique M. D. Santos. Password secured sites - stepping forward with keystroke dynamics. In *International Conference on Next Generation Web Services Practices. NWeSP 2005.*, pages 6–11, Aug. 2005.

[8] Solar Designer and Dug Song. Passive analysis of ssh (secure shell) traffic. *Openwall advisory OW-003*, 2001.

[9] Paul Dowland, Steven Furnell, and Maria Papadaki. Keystroke analysis as a method of advanced user authentication and response. In *Proceedings of the IFIP TC11*

*17th International Conference on Information Security: Visions and Perspectives*, volume 214, pages 215–226, 2002.

[10] Paul Dowland, Harjit Singh, and Steven Furnell. A preliminary investigation of user authentication using continuous keystroke analysis. In *Proceedings of the Workshop Conference on Information Security Management and Small Systems Security*, 2001.

[11] Denis Foo Kune and Yongdae Kim. Timing attacks on pin input devices. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS 2010, pages 678–680, New York, NY, USA, 2010. ACM.

[12] Romain Giot, Mohamad El-Abed, and Christophe Rosenberger. Greyc keystroke: A benchmark for keystroke dynamics biometric systems. In *IEEE 3rd International Conference on Biometrics: Theory, Applications, and Systems. BTAS 2009.*, pages 1–6, 2009.

[13] Shallen Giroux, Renata Wachowiak-Smolikova, and Mark P. Wachowiak. Keypress interval timing ratios as behavioral biometrics for authentication in computer security. In *First International Conference on Networked Digital Technologies. NDT 2009.*, pages 195–200, July 2009.

[14] Shallen Giroux, Renata Wachowiak-Smolikova, and Mark P. Wachowiak. Keystroke-based authentication by key press intervals as a complementary behavioral biometric. In *IEEE International Conference on Systems, Man and Cybernetics. SMC 2009.*, pages 80–85, Oct. 2009.

[15] Daniele Gunetti and Claudia Picardi. Keystroke analysis of free text. *ACM Trans. Inf. Syst. Secur.*, 8(3):312–347, Aug. 2005.

[16] Kathryn Hempstalk. Continuous typist verification using machine learning. PhD dissertation, The University of Waikato, 2009.

[17] Rajkumar Janakiraman and Terence Sim. Keystroke dynamics in a general setting. In Seong-Whan Lee and StanZ. Li, editors, *Advances in Biometrics*, volume 4642 of *Lecture Notes in Computer Science*, pages 584–593. Springer Berlin Heidelberg, 2007.

[18] Kevin S. Killourhy and Roy A. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *IEEE/IFIP International Conference on Dependable Systems Networks. DSN 2009.*, pages 125–134, July 2009.

[19] Paul C. Kocher. *Cryptanalysis of Diffie-Hellman, RSA, DSS, and other systems using timing attacks (Extended Abstract)*, pages 27–31. Springer-Verlag, 1995.

[20] Jae-Wook Lee, Sung-Soon Choi, and Byung-Ro Moon. An evolutionary keystroke authentication based on ellipsoidal hypothesis space. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO 2007, pages 2090–2097, New York, NY, USA, 2007. ACM.

[21] Fabian Monrose and Aviel Rubin. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM conference on Computer and communications security*, CCS 1997, pages 48–56, New York, NY, USA, 1997. ACM.

[22] Jugurta R. Montalvão Filho and Eduardo O. Freire. On the equalization of keystroke timing histograms. *Pattern Recognition Letters*, 27(13):1440–1446, 2006.

[23] Mordechai Nisenson, Ido Yariv, Ran El-Yaniv, and Ron Meir. Towards behaviometric security systems: Learning to identify a typist. In Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel, editors, *Knowledge Discovery in Databases: PKDD 2003*, volume 2838 of *Lecture Notes in Computer Science*, pages 363–374. Springer Berlin Heidelberg, 2003.

[24] Alen Peacock, Xian Ke, and Matthew Wilkerson. Typing patterns: a key to user identification. *Security Privacy, IEEE*, 2(5):40–47, Sep.-Oct. 2004.

[25] Toshiharu Samura and Haruhiko Nishimura. Keystroke timing analysis for individual identification in japanese free text typing. In *ICCAS-SICE, 2009*, pages 3166–3170, Aug. 2009.

[26] Claude E. Shannon. Prediction and entropy of printed english. *Bell System Technical Journal*, 30(1), Jan. 1951.

[27] Terence Sim and Rajkumar Janakiraman. Are digraphs good for free-text keystroke dynamics? In *IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2007.*, pages 1–6, June 2007.

[28] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of the 10th conference on USENIX Security Symposium*, volume 10 of *SSYM 2001*, pages 25–25, Berkeley, CA, USA, 2001. USENIX Association.

[29] Deian Stefan, Xiaokui Shu, and Danfeng (Daphne) Yao. Robustness of keystroke-dynamics based biometrics against synthetic forgeries. *Computers & Security*, 31(1):109–121, 2012.

[30] Deian Stefan and Danfeng Yao. Keystroke-dynamics authentication against synthetic forgeries. In *6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 1–8, Oct. 2010.

[31] Charles C. Tappert, Sung-Hyuk Cha, Mary Villani, and Robert S. Zack. A keystroke biometric system for long-text input. *International Journal of Information Security and Privacy (IJISP)*, 4(1):32–60, 2010.

[32] Charles C. Tappert, Mary Villani, and Sung-Hyuk Cha. Keystroke biometric identification and authentication on long-text input. In Liang Wang and Xin Geng, editors, *Behavioral Biometrics for Human Identification: Intelligent Applications*. Medical Information Science Reference, 1 edition, 2009.

[33] Jonathan T. Trostle. Timing attacks against trusted path. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 125–134, May 1998.

[34] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. Secretly monopolizing the cpu without superuser privileges. In *Proceedings of 16th USENIX Security Symposium*, pages 1–18, 2007.

[35] Mary Villani, Charles Tappert, Giang Ngo, Justin Simone, Huguens St. Fort, and Sung-Hyuk Cha. Keystroke biometric recognition studies on long-text input under ideal and application-oriented conditions. In *Computer Vision and Pattern Recognition Workshop. CVPRW 2006.*, pages 39–39, June.

[36] Kai Xi, Yan Tang, and Jiankun Hu. Correlation keystroke verification scheme for user access control in cloud computing environment. *The Computer Journal*, 54(10):1632–1644, 2011.

[37] Ge Zhang and Simone Fischer-Hübner. Timing attacks on pin input in voip networks (short paper). In Thorsten Holz and Herbert Bos, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 6739 of *Lecture Notes in Computer Science*, pages 75–84. Springer Berlin Heidelberg, 2011.

[38] Kehuan Zhang and XiaoFeng Wang. Peeping tom in the neighborhood: keystroke eavesdropping on multi-user systems. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM 2009, pages 17–32, Berkeley, CA, USA, 2009. USENIX Association.