Wireless Training-free Keystroke Inference Attack and Defense

Edwin Yang, Song Fang, Ian Markwood, Yao Liu, Shangqing Zhao, Zhuo Lu, and Haojin Zhu

Abstract—Existing research work has identified a new class of attacks that can eavesdrop on the keystrokes in a non-invasive way without infecting the target computer to install malware. The common idea is that pressing a key of a keyboard can cause a unique and subtle environmental change, which can be captured and analyzed by the eavesdropper to learn the keystrokes. For these attacks, however, a training phase must be accomplished to establish the relationship between an observed environmental change and the action of pressing a specific key. This significantly limits the impact and practicality of these attacks. In this paper, we discover that it is possible to design keystroke eavesdropping attacks without requiring the training phase. We create this attack based on the channel state information extracted from the wireless signal. To eavesdrop on keystrokes, we establish a mapping between typing each letter and its respective environmental change by exploiting the correlation among observed changes and known structures of dictionary words. To defend against this attack, we propose a reactive jamming mechanism that launches the jamming only during the typing period. Experimental results on software-defined radio platforms validate the impact of the attack and the performance of the defense.

Index Terms—Keystroke eavesdropping, correlation, reactive jamming.

1 Introduction

Sensitive information such as classified documents, trade secrets, or private emails are typeset and input into a computer for storage or transmission almost exclusively via a keyboard. Emerging research work has identified a new class of attacks that can eavesdrop on the keystrokes in a non-invasive way [2]–[13]. These new attacks eliminate the requirement to infect the target computer with a keylogger or other malware to violate the user's privacy. Their common underlying principle is that pressing a key on a keyboard causes subtle environmental impacts unique to that key, which can be observed and correlated for all keys. For example, an eavesdropper can set up a malicious WiFi router to receive the wireless signal emitted by a target laptop. A user pressing a key causes a unique disturbance on the received signal, and the eavesdropper can analyze these disturbances to learn which key is pressed. In general, these non-invasive keystroke eavesdropping attacks can be classified into three categories, vibration based [5], [6], acoustic signal based [7]–[9], [13], and wireless signal based [2]–[4], [14].

These attacks also share a common weakness, i.e., requiring a training phase. This establishes the relationships between observed environmental disturbances and specific key presses. During the attack phase, unknown disturbances are compared with those recorded in the training phase to determine which key was most likely pressed. However, the training significantly limits the impact of these attacks. Most existing works [2]–[7], [9]–[12], [14] assume the attacker has some way to perform the training in a practical situation, but none have provided technical details justifying their logistical feasibility. [13] proposes a practical way to collect keystrokes for training by Voice-over-IP (VoIP) software (e.g., Skype), while this technique targets the scenario when the attacker is able to talk with the target user via VoIP calls.

Requiring training imposes a large practical hurdle for the attacker - most users are in full physical control of their keyboards, whether they are part of a laptop set in arbitrary locations or on a roll-out keyboard tray (a common feature of desks). Anytime a laptop is moved or a keyboard tray is pushed in or pulled out slightly, any previous training efforts are invalidated. A user may also change typing behaviors (heaviness of hand, etc.) during use of the computer. Hence, training must be conducted frequently to cope with all these changes. Because training requires knowledge of what key is pressed to construct a mapping, and therefore requires access to the system for some time, it is impossible to retrain once the user has control of the system, and it is highly difficult to train on systems controlled physically by the user (which are most).

In this paper, we make non-invasive keystroke eavesdropping practical by removing the training requirement entirely. Not only does this make these attacks actually possible, but it also makes them far less invasive still, because physical access to the system is never required.

Intuitively, statistical methods provide a way to remove the training phase. Frequency analysis [15] is a typical unsupervised learning method based on the statistical observation that certain letters normally occur with varying frequencies in a given language. In English, the letter 'e' is the most often used. An input text of sufficiently large

- *E. Yang and S. Fang are with the School of Computer Science, University of Oklahoma, Norman, OK, 73019. E-mail: {edwiny, songf}@ou.edu.*
- *I. Markwood is with Cyber Development, BlackHorse Solutions, Herndon, Virginia, 20171. E-mail: imarkwood@mail.usf.edu.*
- *S. Zhao is with the School of Computer Science, University of Oklahoma, Tulsa, OK, 74135. E-mail: shangqing@ou.edu.*
- *Y. Liu is with the Department of Computer Science and Engineering, and Z. Lu is with the Department of Electrical Engineering, University of South Florida, Tampa, FL, 33620. E-mail: {yliu@cse., zhuolu@}usf.edu.*
- *H. Zhu is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. E-mail: zhu-hj@cs.sjtu.edu.cn.*
- *Corresponding author: Song Fang.*

An earlier version of the work [1] was presented in ACM CCS'18.

size will have a distribution of letter frequencies close to the typical distribution of English letters [16]. Since an environmental disturbance is associated with a key, by analyzing the frequencies of observed disturbances, the attacker can possibly determine the associated keys. Intuitively, the most frequently observed disturbance is likely to be caused by typing the letter 'e'.

However, statistical methods determine the typical distribution of English letters by ingesting a large amount of text, while the distribution within a small sample text may not be quite the same. The discrepancy between the sample and typical distributions is unpredictable, so correlating environmental disturbances and keystrokes requires collecting statistics over a long period, during which the environmental disturbances (e.g., wireless signal properties) for different keystrokes must remain static as well as distinct from one another. In practice, these disturbances (especially wireless signals) may change over time due to environmental changes and mobility, preventing the attacker from collecting sufficient reliable statistics for keystroke inference.

The challenges with using statistical methods motivate us to develop an effective approach for non-invasive keystroke eavesdropping within a shorter time window. We analyze the self-contained structures of words, which can be immediately observed by typing a single word, rather than probabilistic statistics among words, which require many words to establish. In particular, we notice that the repetition or uniqueness of characters in a word shows through the structure of repeated or unique environmental disturbances collected in the process of eavesdropping. For example, assume that a user types "sense", and accordingly the attacker observes five environmental disturbances. The first and fourth observed disturbances are similar to each other, because they are caused by the action of pressing the same key "s". Similarly, the second and last disturbances appear alike, because they are caused by pressing the same key "e". This structural information enables the attacker to quickly identify the typed word, as only one word "sense" from the 1,500 most frequently used words [17] matches this structure, achieving a much faster establishment of a mapping between disturbances and characters typed. This observation also requires no prior interaction with the user's system and thus facilitates fast and accurate training-agnostic keystroke eavesdropping.

To exploit this observation, we must compare the correlations among letters of words with those among observed disturbances. This requires a self-contained feature that can quantify such correlations and be compared against others. We identify and describe herein such a feature, having three necessary characteristics. First, it achieves high uniqueness to provide fast distinction among differently structured words. Second, it can be extracted both from words and sets of observed disturbances, so the two can be compared. Lastly, as more words are typed, their corresponding structures can be captured and integrated with previous information to refine and shrink the search space.

Using this feature, we create approaches to compare sets of observed disturbances to possible candidate words. Our technique has mechanisms to adapt to and retain high accuracy in the presence of natural noise and sudden environmental changes, which may cause similar disturbances to appear different or vice versa. It is similarly able to continue inferring letters in the presence of non-alphabetical characters such as punctuation, navigation arrows, delete and backspace keys, etc.

Our attack analyzes disturbances in a wireless signal, which can penetrate through obstacles, so it does not require line-of-sight between the attacker and the victim. External wireless devices controlled by the attacker are used to collect the signal disturbances, so there is no need for exploits to install malware on the target computer. The attack is especially suitable for the wireless scenario, since the wireless channel is time-varying and it can quickly determine the disturbance-key relationship. Within a short time window, the attacker can apply this relationship to infer the remaining keystrokes, including typed words not in the dictionary. Except for English, our attack can also be utilized to infer other languages, such as source code. To defend against the attack, we propose a reactive jamming scheme called *TypeGuard* which prevents the eavesdropper from obtaining enough CSI information for keystroke inference.

We implement the proposed attack and defense on Universal Software Radio Peripherals (USRPs) X300 platform. Experimental results show that for a sample input of 150 words, the proposed attack can recognize an average of 95.3% of these words, whereas frequency analysis can only recognize less than 2.4%. We also note that the attacker only needs 1-2 minutes to collect 50 words to identify the disturbance-key relationship that allows a word recovery rate of 94.3%. Besides, we verify the feasibility of inferring Linux kernel source code with the proposed attack. Furthermore, we show that the attacker can effectively decrease the entropy of a 9-character password from 54.8 bits to as low as 5.4 bits, vastly reducing the maximum brute-force attempts required for breaking the key from 31.08 quadrillion to just 42. On the other hand, *TypeGuard* can jam 97% of the user's typing duration on average so that the attacker is unable to infer keystrokes without insufficient CSI waveforms.

In summary, our contributions are as follows:

- We propose a novel wireless keystroke eavesdropping attack, which requires no training.
- We develop a dictionary-assisted demodulation algorithm to establish the mapping between typing each letter and its respective environmental change.
- We design a defense technique to defend against the proposed attack.
- We implement real-world prototypes of both the proposed attack and defense techniques using USRPs, evaluating the performance of the attack and the effectiveness of the defense.

2 PRELIMINARIES

As wireless signals can penetrate through obstacles [18]–[21], we monitor this environment for our training-agnostic attack to remove the line-of-sight requirement.

2.1 Channel State Information

Wireless signal disturbances can be quantified by the CSI measurement, which describes how the wireless channel impacts the radio signal that propagates through the channel (e.g., amplitude attenuation and phase shift) [22].

The orthogonal frequency-division multiplexing (OFDM) technique is widely used in modern wireless communication systems (e.g., 802.11a/g/n/ac/ad). OFDM utilizes multiple subcarrier frequencies to encode a packet, and the *channel frequency responses* measured from the subcarriers form the CSI of OFDM. The channel frequency response at time t is denoted by $H(f,t)$, where f represents a particular subcarrier frequency, and it is usually estimated by using a pseudo-noise sequence that is publicly known [23]. Specifically, a transmitter sends a pseudo-noise sequence over the wireless channel, and the receiver estimates the channel frequency response from the received, distorted copy and the publicly known original sequence. Let $X(f,t)$ denote the transmitted pseudo-noise sequence. Based on the received signal $Y(f,t)$, $H(f,t)$ can be calculated by $H(f,t) = \frac{Y(f,t)}{X(f,t)}$. Existing work utilizes the amplitude of CSI to extract keystroke waveforms [3], [4]. In this paper, we also explore the amplitude of CSI and refer to this as just "CSI" in the following.

2.2 Existing Work on CSI-based Keystroke Inference

Researchers have proposed to utilize CSI to recognize subtle human activities, including mouth movements [24] and keystrokes [3], [4]. Existing techniques [3], [4] on CSI-based keystroke inference assume that the attacker typically sets up a wireless transmitter and receiver in close proximity of the target keyboard. If the keyboard is part of a computer like a laptop that can connect to wireless networks, the computer itself transmits the wireless signal whenever it needs to exchange information with the WiFi router, and thus it can play the role of the transmitter for the attacker. The receiver can then be a malicious 802.11 access point that provides free WiFi service to attract victim computers to connect to it. In a general case, the attacker can also create a custom transmitter and receiver using software-defined radio platforms such as USRPs. The transmitter transmits the wireless signal to create a radio environment, and the receiver receives the signal and computes the CSI.

These techniques normally use three steps to infer keystrokes, namely, pre-processing, training, and testing. Pre-processing removes noise from the CSI, reduces computational complexity, and segments the time series of the CSI into individual samples that correspond to keystrokes. The training phase records each keystroke and the corresponding CSI so that a training model for classification can be built. In the testing phase, an observed CSI for an unknown keystroke is matched within the training model to determine which keystroke it corresponds to. Our attack uses the same pre-processing step as these existing techniques.

3 Attack Design

3.1 System Overview

We consider a general attack scenario, where the attacker uses a customized transmitter and receiver pair to launch this attack. The attacker constantly transmits the wireless signal, or just whenever typing activity is detected. In the latter case, a WiFi packet analyzer can detect when a user starts to type [4]. We assume the typed content is in English, though the attack can target other languages just as easily.

The receiver needs to collect the CSI, so the attacker implements a channel estimation algorithm such as the one mentioned in Section 2.1. The estimated CSI stream is divided by the *pre-processing* step into individual segments that correspond to the actions of pressing a key. In this paper, we refer to a segment as a *CSI sample*. After pre-processing, unlike the existing methods, the training-agnostic attack takes three different important steps to infer keystrokes, namely CSI word group generation, dictionary demodulation, and alphabet matching.

CSI word group generation partitions the CSI samples into groups corresponding to each typed word. The attacker will explore the correlation among and order of unique letters in each word to infer keystrokes, and thus needs to separate the stream into words. This step performs this task by identifying the CSI samples caused by pressing the space key, since words are almost always separated by a space. Dictionary demodulation aligns the correlation of CSI samples to that of letters in a word, so as to find the corresponding word for a CSI word group. Based on the demodulation result, potential mappings are formed between CSI samples and keystrokes, with which the attacker can infer the remaining typed words, including those not appearing in the dictionary.

3.2 CSI word group generation

CSI word group generation involves classification, sorting, and word segmentation.

3.2.1 Classification

Dynamic Time Warping is a classical technique to measure the similarity between two temporal sequences [25], and it has been widely used to identify the spatial similarity between the signal profiles of two wireless links [3], [4], [26], [27]. Thus, to quantify the similarity between two CSI samples, we utilize the Dynamic Time Warping technique to calculate the distance between them. A small distance indicates that both CSI samples are similar and accordingly that they originate from the same key. Conversely, a large distance indicates that they deviate from each other, and that they are caused by two different keys. We assume that the victim user presses a single key at a time, since this is the common typing behavior for most keyboard users.

3.2.2 Sorting

Since the space character is used to connect consecutive words, it normally appears more frequently than any other character in a long text. We thus expect that the CSI sample caused by the space key also appears more frequently than other CSI samples. The classification outcome includes multiple sets, each consisting of similar CSI samples. We sort the sets according to size and associate the space key with the largest set, so that all CSI samples in this set are assumed to be caused by pressing the spacebar. If this association is incorrect, we will ultimately not be able to recover meaningful English words. In that case, we continue on, associating the space key to the second-largest set and reattempting the same recovery process. We try these sets from largest to smallest cardinality until we successfully recover meaningful English words or exhaust all sets.

3.2.3 Word Segmentation

Once the set of CSI samples associated with the space key is identified, we can start the word segmentation process to find the CSI samples comprising each word of the typed content. Everything between two successive CSI samples from the space-associated set is grouped together. In the following, we refer to such a group as a *CSI word group*, and this does not include the spaces at either end. CSI word groups will be used as the input of the dictionary demodulation method to eventually establish the complete mapping between the CSI samples and keystrokes.

3.3 Dictionary Demodulation

Dictionary demodulation converts CSI word groups to corresponding English words. We begin by developing a feature to apply to these CSI word groups suitable for narrowing down the search space of possible candidates.

3.3.1 Feature Selection

Ideally, a feature extracted from each CSI word group would enable us to uniquely determine the corresponding word. Our strategy is thus to find a feature that can divide all words in the dictionary into as many sets as possible, to achieve high distinguishability.

Without knowing the exact letters in a word, but having a CSI sample for each letter, we can determine the number of constituent letters and whether or not any letters in the word are repeated. These two pieces of information yield two features to partition words, and we utilize a top 1,500 most frequently used word list [17] as the dictionary to calculate the number of sets divided by each. To quantify the distinguishability of a feature, we define a new metric, called the *uniqueness rate*, as the ratio T_p/T, where T is the number of considered words, and T_p represents the number of sets obtained by dividing T words according to the selected feature. The uniqueness rate should be maximized for the best partitioning of the words. We next evaluate the uniqueness rates for our two features:

Length: We empirically find that all words in this dictionary are 1-14 characters long. If we choose length as the only feature, we can divide all words into 14 sets, the members of each set having the same length. On average, each set has $1,500/14 \approx 107$ words. This means that an input CSI word group will have an average of 107 possible candidate words based on the length feature. The uniqueness rate is then $14/1,500 \approx 0.009$.

CSI Sample Repetition: We also count the number of distinct letters that repeat. We denote the repetition information of a word as S_r, and we set $S_r = 0$ if no repetition is found. Otherwise, we denote S_r by $(t_1, \cdots, t_r)$, where r is the number of distinct letters that repeat, and t_i ($i \in \{1, \cdots, r\}$) denotes how many times the corresponding letter repeats. For example, the repetition information for the word "level" should be (2, 2), because 2 different letters ('l' and 'e') repeat, and both letters repeat twice respectively. Considering a word of length L, we can quantify the repetition information using (L, S_r). Using this repetition information, we can then divide all 1,500 words into a total of 63 sets, such that members of each set share the same value of (L, S_r). On average, each set has $1,500/63 \approx 24$ words, so an input CSI word group will be mapped to one of 24 words based on this feature. The uniqueness rate is then $63/1,500 \approx 0.042$.

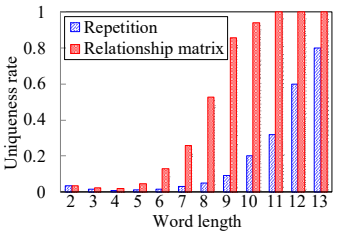


Fig. 1. Uniqueness rate for words of different length.

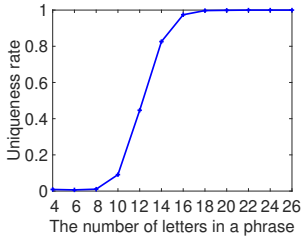


Fig. 2. Uniqueness rate for joint words.

The repetition feature has better distinguishability than the length feature, because its larger uniqueness rate yields a reduced search space to map an input CSI word group to a word. The repetition feature only provides the result of repeated letters in a word, however, and does not consider the position information of these letters. We expect that the uniqueness rate can be further increased if we construct a feature that not only employs the word length and repetition information, but also distinguishes the positions of repeated letters from non-repeated letters.

3.3.2 Inter-Element Relationship Matrix

We define a new data structure to represent the structure of every word/CSI word group. Specifically, we denote a word or a CSI word group by a vector $[x_1, \ldots, x_n]$ of n elements, each of which is a letter (CSI sample). We then define its *inter-element relationship matrix* as

$$M : [x_1, \ldots, x_n] \mapsto \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & r_{2,3} & \cdots & r_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{n,1} & r_{n,2} & r_{n,3} & \cdots & r_{n,n} \end{bmatrix}.$$

For a CSI word group, we set $r_{i,j} = 1$ if x_i and x_j are similar CSI samples as classified in the CSI word group generation step (Section 3.2). Otherwise, we set $r_{i,j} = 0$. The diagonal elements are always 1 and the matrix is symmetric.

We build the inter-element relationship matrix for each word and ultimately partition the 1,500 most commonly used words into 337 sets. The words in a particular set have the same inter-element relationship matrix. On average, each set has about $1,500/337 \approx 4$ words which are possible candidates for the CSI word group having that inter-element relationship matrix. The corresponding uniqueness rate is $337/1,500 \approx 0.225$, which is much larger than those of the previously discussed features.

Empirically, we find the uniqueness rates for words of different lengths are not evenly distributed, and this fact actually enables our scheme. Figure 1 presents the uniqueness rates for the inter-element relationship matrix as well as the repetition feature for comparison, respective to word length. The relationship matrix clearly performs much better than the repetition feature in all cases, but very evident also is as words become larger, they become more uniquely structured, leading to high uniqueness rates for the relationship matrix. For example, the uniqueness rate for a 3-letter word is 0.025, while that for a 10-letter word is 0.940.

Indeed, a phrase comprised of multiple words can be considered as one "long word" for the purpose of gen-

erating an inter-element relationship matrix, though the dictionary must also expand to contain these combinations. Assuming a phrase formed by N words, the new dictionary will include $T_1T_2\cdots T_N$ phrases, where T_i ($1 \leq i \leq N$) is the size of the set of candidate words having a length equal to the i-th CSI word group. Figure 2 illustrates how the uniqueness rate benefits from the combination of each pair of two words from the dictionary of 1,500 most used words. The words in each pair range from 2 to 13 characters in length, for a possible total of 4-26 characters. The uniqueness rate jumps as the length of these word pairs increases, and after 18 total characters, the pair of words has a fully unique structure. This indicates within a few words it should always be possible to narrow down to the specific content the victim types, giving rise to our joint demodulation method.

3.3.3 General Joint Demodulation Method

After CSI word group generation, assume that the attacker obtains from the eavesdropped typing m CSI word groups denoted by $\mathbf{S} = \{S_1, S_2, \ldots, S_m\}$. We further use $W_1, W_2, \ldots, W_q$ to denote the q words in the dictionary $\mathbf{W}$. Our goal is to find a phrase of m words that correspond to the m CSI word groups. Clearly, while each individual CSI word group could have several candidate dictionary words with matching structure, each candidate will impose a mapping of some CSI samples and letters on some successive words, and several of these possible mappings will result in successive words that are not real, so the below technique works to rule out these impossible mappings. The full method includes two steps: 1) demodulation of each single CSI word group; and 2) joint demodulation of multiple CSI word groups.

Step 1: This step finds initial candidate words for each CSI word group or determines if a word cannot be immediately demodulated and must be returned to later. We first create the inter-element relationship matrices for $W_1, W_2, \ldots, W_q$ in our dictionary $\mathbf{W}$. We next iterate over each $S_i \in \mathbf{S}$, creating its inter-element relationship matrix and considering the subset $\mathbf{W}'$ of $\mathbf{W}$ whose entries are of the same length as S_i. We compare the relationship matrix of S_i to that of each $W_j \in \mathbf{W}'$ and mark that W_j as a candidate if the two matrices are equal. If no candidates match, the word must not appear in the collection of English words comprising our dictionary, so we add S_i to the "undemodulated set" $\mathbf{U}$.

Step 2: This step works to build up a mapping between CSI samples and letters that works for multiple CSI word groups simultaneously, successively ruling out the many candidates established by the first step, until (ideally) only one candidate remains for each word and the message is uncovered. Conceptually, we iterate over the word groups not in the undemodulated set $\mathbf{U}$,

(a) concatenating each with all those previous,
(b) applying each possible mapping thus far constructed,
(c) ruling out all candidates that cannot coexist with any mappings,
(d) and adding any new CSI sample/character mapping information from the remaining candidates.

Specifically, we name T_i the concatenation of the first $i-1$ CSI word groups $\{S_1, \ldots, S_{i-1}\}, 1 < i \leq m$, excluding any $S_k \in \mathbf{U}$. In other words, while considering S_i, we concatenate all the previous CSI word groups which have candidates into T_i. Candidates for T_i, or groups of valid words satisfying the structures of the CSI samples comprising T_i, are denoted by $\mathbf{T}_{i_C} = \{T_{i_1}, T_{i_2}, \ldots, T_{i_p}\}$. Further, candidates for S_i, as determined by Step 1, are denoted by $\mathbf{S}_{i_C} = \{S_{i_1}, S_{i_2}, \ldots, S_{i_q}\}$. With $T_i||S_i$ signifying the concatenation of T_i and S_i, we calculate the inter-element relationship matrix for $T_i||S_i$, as well as that for every $T_{i_j}||S_{i_k}, T_{i_j} \in \mathbf{T}_{i_C}, S_{i_k} \in \mathbf{S}_{i_C}$. We note that this is $p \times q$ matrices to be compared and that this series of comparisons happens at each iteration; we analyze the time complexity in Section 5.3, and our experiments show the number of comparisons converges quickly over successive iterations. Then, if the relationship matrix for one such $T_{i_j}||S_{i_k}$ matches that for $T_i||S_i$, we know that the CSI sample/character mapping of the candidate S_{i_k} will work in concordance with the mapping established for T_{i_j} while maintaining the structure stipulated by $T_i||S_i$. Each such $T_{i_j}||S_{i_k}$ is therefore a new candidate for T_{i+1}.

In the event that no $T_{i_j}||S_{i_k}$ has a relationship matrix matching that for $T_i||S_i$, this means that no CSI sample/character mappings satisfying the structure of T_i result in valid words within our dictionary when applied to S_i. Such S_i are placed in $\mathbf{U}$ and execution skips to S_{i+1}. Pseudocode for this step is shown in Algorithm 1. In this manner, we iterate over i and gradually build up T_i until all distinct CSI samples are mapped to characters in the alphabet. At this time, the mapping can be applied to the remaining word groups, including those in $\mathbf{U}$, for which no matches were found in the dictionary used. An example of this final alphabet matching is visible in Figure 3.

Algorithm 1 Joint Demodulation

```
1:  procedure JOINT_DEMOD(S_i, T_{i_C}, S_{i_C}, U)
2:      T_{(i+1)_C} ← ∅ (i > 0)
3:      for T_{i_j} in T_{i_C} do
4:          for S_{i_k} in S_{i_C} do
5:              if M(T_i||S_i)=M(T_{i_j}||S_{i_k}) then
6:                  T_{(i+1)_C} ← T_{(i+1)_C} ∪ T_{i_j}||S_{i_k}
7:              end if
8:          end for
9:      end for
10:     if T_{(i+1)_C} = ∅ then          ▷ no candidates, skip S_i
11:         U ← U ∪ S_i
12:         T_{(i+1)_C} ← T_{i_C}
13:     end if
14:     return T_{(i+1)_C}, U
15: end procedure
```

3.3.4 Error tolerance

Wireless channel noise may cause CSI classification errors, such that a recorded CSI sample for a character might not appear like others for that character or may appear like a different character. Otherwise, CSI samples may be classified correctly but a typo by the user may mean a word is misspelled and will not appear in the dictionary. This can cause a concatenated set of CSI word groups to have an incorrect inter-element relationship matrix, which may match with invalid words or have no candidates at

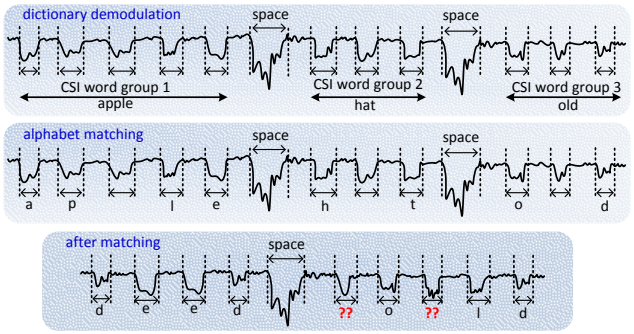


Fig. 3. Assume a simple dictionary of three words "apple", "hat", and "old", typed in that order by the user. The alphabet of this dictionary consists of 8 letters "a","p","l","e","h","t","o", and "d". Dictionary demodulation maps each letter in this alphabet to the corresponding CSI sample, and any further CSI word groups may simply have this mapping applied to them. After matching, suppose the user then types the word "deed", the attacker can directly demodulate the observed CSI word group, which did not appear in the dictionary. Next, assume instead the second typed word is "would". Since "w" and "u" do not appear in the alphabet of this simple dictionary, the attacker cannot decode them but can continue decoding the other letters "o", "l", and "d".

all. The latter is the ideal case as the word group having the CSI sample in question will simply be added to the undemodulated set and skipped. However, if invalid words are incorporated into the candidates for joint demodulation, incorrect relationship matrices will continue to be used as the joint demodulation progresses, and the content recovery will fail. We have observed in experiments that even if a wrong matrix matches other word sequences, cascading discovery failures inevitably happen for successive words.

The attacker may employ this observation to work around the presence of typos or CSI classification errors. If a CSI word group is successfully demodulated but continuous recovery failures occur thereafter, this word can be added to the undemodulated set and skipped in favor of proceeding with the next word. Further word groups are thus less likely to be processed with an incorrect portion of the relationship matrix, and a correct mapping is more probable. Algorithm 2 shows how to check for cascading errors at each step i based on the demodulation result for S_i. Finally, when the mapping is complete and applied to the CSI word groups in the undemodulated set, any errors in CSI classification or typos will persist, but not further damage the results. The attacker can use some common knowledge to work out these errors and any other ambiguities.

In the event the cascading errors do not seem to be avoidable, this is evidence that the wireless channel has changed, because as previously mentioned the channel is time-varying. In this case, the dictionary demodulation may be begun anew, so that the attack can adapt to the changes.

3.3.5 Impact of Non-Alphabetical Characters

Users mostly type alphabetical characters and spaces, but also occasionally use numbers and punctuation, which obviously cannot be matched by examining word structures. If these appear during alphabet mapping construction, they will cause cascading demodulation errors, be added to the undemodulated set, and be skipped, similar to typos or CSI classification errors as just discussed. If the mapping has already been constructed, the CSI samples for these numbers or punctuation will not appear in the mapping

Algorithm 2 Error Handling

$[\mathbf{T}_{(i+1)_C}, \mathbf{U}] = \text{JOINT\_DEMOD}(S_i, \mathbf{T}_{i_C}, S_{i_C}, \mathbf{U})$
if $\mathbf{T}_{(i+1)_C} \neq \mathbf{T}_{i_C}$ **then** ▷ demodulation success
  $F \leftarrow$ allowable failure threshold
  $flag \leftarrow$ true
  for $j \in \{i+1, \cdots, i+F\}$ **do**
    $[\mathbf{T}_{(j+1)_C}, \mathbf{U}]$=JOINT_DEMOD$(S_j, \mathbf{T}_{j_C}, S_{j_C}, \mathbf{U})$
    if $\mathbf{T}_{(j+1)_C} \neq \mathbf{T}_{j_C}$ **then** ▷ demodulation success
      $flag \leftarrow$ false; **break** ▷ reset failure count
    end if
  end for
  if $flag$ **then** ▷ reached failure threshold
    $\mathbf{U} \leftarrow \mathbf{U} \cup S_i$ ▷ skip S_i
    $\mathbf{T}_{(i+1)_C} \leftarrow \mathbf{T}_{i_C}$
  end if
end if

and will be left as unknown. In both cases, the attacker can use some common knowledge to infer or narrow down candidates for these characters.

For example, users press the backspace key to remove multiple characters before the cursor and then continue typing. For a CSI word group that is recovered as "abab××out", the attacker may recognize that the unidentified character "×" corresponds to the backspace key and that the word should be "about". In another case, a user may press the left arrow key to move the cursor backward, insert some text, and then press the right arrow key to return the cursor to the original position. Hence, the left and right arrow keys often appear in pairs and are each pressed multiple times. In this way, an attacker may infer the word "about" from a CSI word group recovered as "aut◁◁bo▷▷", with unidentified samples "◁" and "▷" corresponding to left and right arrow keys, respectively.

4 COUNTERMEASURES

The proposed keystroke inference attack explores the inter-element relationship matrix to eavesdrop typing content via intercepted wireless signals. Intuitively, to defend against such an attack, we should disrupt the attacker from obtaining the correct relationship. The user may manually encrypt the words to be typed by using some traditional substitution and permutation ciphers. However, this approach is impractical, because it requires the user to calculate and type in the ciphertext, an unintelligible string of random appearance which would take much more time to type and incur numerous input errors. The encryption also brings an extreme computational burden to the user.

Instead, we investigate two privacy preservation directions to protect typing content, i.e., hardware based and fake input based defenses. We begin by developing *TypeGuard*, a hardware based technique that introduces a selective jamming mechanism to obfuscate the received signals at the eavesdropper. Then we discuss how to construct fake input to fail the keystroke inference.

4.1 TypeGuard

Ideally, a constant jammer is able to make the attacker fail to obtain accurate CSI, which is required for all wireless-based

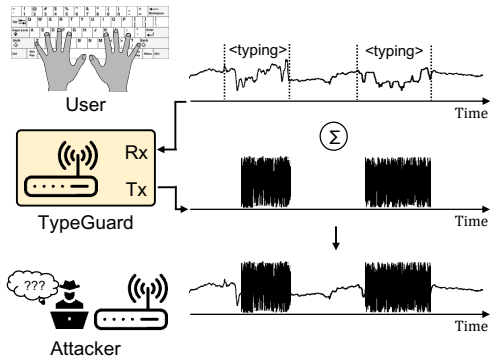


Fig. 4. *TypeGuard* acts as a reactive jamming device, which first determines starting point of jamming based on the first keystroke event of each typing event and then emits jamming signals until the channel becomes stable (i.e., the user stops typing).

keystroke eavesdropping attacks, including the proposed one. However, it is quite inefficient and expensive to utilize jamming which never stops. Compared to constant jamming, reactive jamming is not only cost effective, but also hard to track and compensate against [28]. With *TypeGuard*, the legitimate user deploys a wireless reactive jamming device, which listens to signals from the wireless channel and also transmits noise signals to the wireless channel to interfere with the attacker's transmissions once the typing is detected. As a result, the attacker will not be able to collect enough accurate CSI to analyze self-contained structures of words, leading to the failure of launching the proposed wireless-based keystroke eavesdropping attack. Figure 4 illustrates the defense mechanism of *TypeGuard*.

Determination of Jamming Starting Point: A reactive jammer (i.e., defender) needs to take a reactive time to detect the typing and initialize the jamming. To detect the typing (i.e., the event of at least one keystroke), the defender needs to collect the waveform of the first keystroke in the typing session. Thus, the ending point of the first keystroke waveform would trigger the jamming. Each keystroke normally corresponds to a sharp fall and rise pattern in the CSI waveform, which in turn facilitates the detection of each keystroke duration, as described in Section 5.1.

Stop Jamming: To obfuscate received signals at the attacker, *TypeGuard* transmits high-power noise signals, which can become dominant at the attacker side. *TypeGuard* then needs to return to the inactive mode once it identifies the end of the typing session.

Due to the existence of the reaction time, the attacker is still able to obtain the first keystroke waveform for each typing session when *TypeGuard* is launched. However, with only one keystroke waveform, the attacker can only guess the first typed character and is unable to infer the whole typing content without knowing the inner structure of the typing content. Therefore, the proposed attack fails. *TypeGuard* has hardware demands, however, the jammer does not need to be a sophisticated high-end device, and it can be any low-cost wireless device (e.g., BladeRF [29] or nRF24L01+ [30]) that can perform basic wireless communication function (e.g., transmitting jamming signals).

4.2 Fake Input based Defense

In a more expedient fashion, based on the target input, the user may first construct a set of characters that are uncommonly used, and then disrupt the inter-element relationship among letters by randomly inserting a large number of characters in such a set while typing. For common English sentences, characters such as $\backslash$, $<$, $>$, and $\&$ are rarely used. While if the target input is source code, such characters, as the basic components of the code, become common. Thus, inserting them may disrupt the integrity of the input. In that case, the user may have to find other uncommon characters based on the category of the source code and the content of input. Specifically, if the user inserts uncommon characters before the first word, the matrix of the first observed CSI word group will either match an incorrect word or not match with any word in the dictionary, so the demodulation algorithm will return incorrect or no candidates. In the former case, the attacker can still correctly demodulate the following word if it shares no letters with the previous. If no candidates are returned, the attacker will discard the first observed CSI word group and start the demodulation algorithm at the second observed CSI word group. Clearly in both cases, to confuse subsequent words, the user must continue inserting uncommon characters in each word.

To further mislead the attacker, the user can also construct sequences of uncommon characters with the same inter-element relationship matrices as various words in the dictionary. The user can type several of these "fake words" before inputting the meaningful content, and continue typing fake words periodically. The fake words can not only feed the attacker with wrong mappings but also mislead the attacker with incorrect eavesdropping results. To prevent the fake words from interfering with the meaningful content, the user may employ a computer program that automatically searches for and removes the uncommon characters or fake words from the input text.

5 Experiment Results

We implement the training-agnostic keystroke eavesdropping attack using USRPs. The prototype attack system includes a wireless transmitter and a receiver. Each node is a USRP X300 with 40 MHz bandwidth CBX daughterboards [31]. The channel estimation algorithm runs at the receiver to extract the CSI for key inference.

The target user operates a desktop computer with a Dell SK-8115 USB wired standard keyboard. The transmitter and the receiver are placed at opposite positions relative to the keyboard. We place the transmitter at a distance of 3 meters away from the keyboard, and the receiver under the 2 cm-thick desk, at a distance of 50 cm away from the keyboard. Also, there is a 4 cm-thick wooden barrier between the transmitter and the keyboard. Thus, both the transmitter and the receiver are not within line-of-sight of the target user. We also form a dictionary using the top 1,500 most frequently used English words [17].

5.1 Example Recovery Process

In this section, we will demonstrate the process of recovering a sample user's typed text.

CSI Sample Extraction: To extract the CSI samples from the CSI time series, we utilize the same pre-processing step as these existing techniques [3], [4]. Correspondingly,

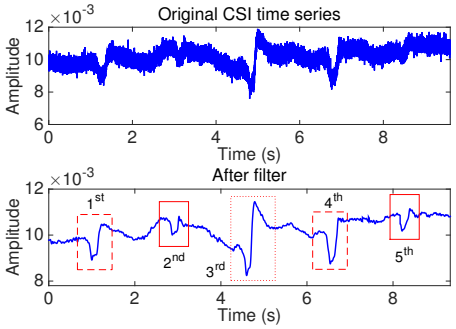


Fig. 5. The CSI word group for the word "sense".

this step has three phases, i.e., noise removal, Principle Component Analysis (PCA) [32], and segmentation.

We observe the frequency of the CSI influenced by keystrokes always lies within a low-frequency range of 2 to 30 Hz. We thus utilize a *Butterworth* low-pass filter [33] to mitigate the impact of high-frequency noise. Initially, the receiver obtains CSI from all subcarriers. We then apply the PCA technique to decrease computational complexity by converting the received CSI into a set of orthogonal components, called principle components [32], which most represent the effects of the keystrokes. The segmentation phase separates the full CSI time series into the individual CSI samples corresponding to single unknown keystrokes. Looking at the CSI waveform, we can observe a sharp fall and rise whenever a key is pressed and released. Therefore, we search over the data for shapes having sharp fall-and-rise features. We utilize A_i to denote the amplitude of the i^{th} extremum of the CSI time series. Suppose the local minima and maxima appear alternately, and the local minima appear first. We use the following steps for segmentation.

- Find all local minima A_{2i-1} and local maxima A_{2i} within the CSI time series, where $i \in \{1,\ldots,N\}$, and N denotes the number of local minima or maxima.
- Calculate the fluctuation $\Delta_j = |A_{j+1} - A_j|$ ($j \in \{1,\ldots,2N-1\}$) and the mean value of the fluctuation $\bar{\Delta} = \sum_{j=2}^{2N} \Delta_j/(2N-1)$. If $\Delta_j > \bar{\Delta}$, we consider this a noteworthy fluctuation caused by a keystroke. Otherwise, it is likely an inconsequential fluctuation caused by noise.
- When we observe n continuous fluctuations (i.e., Δ_j, $\cdots, \Delta_{j+n}, n \geq 2$), and they are all larger than $\bar{\Delta}$, we mark A_j and A_{j+n+1} as the beginning and end of a keystroke, respectively, and the CSI values between A_j and A_{j+n+1} are grouped as a CSI sample.

After the receiver assigns the space character to the most frequently appearing CSI sample group, the remaining samples are grouped into CSI word groups. Figure 5 shows the CSI word group for the word "sense". The full data contains five CSI samples caused by pressing the keys 's', 'e', 'n', 's', and 'e' as visible on the figure. With Dynamic Time Warping, we classify the five samples into three sets, including the pair of the first and fourth samples, the pair of the second and fifth samples, and the third sample alone.

Next, we illustrate how the collected CSI word groups can be narrowed down to the typed content. We choose the Harvard sentences [34] to be typed in for our experiments; these are phonetically balanced sentences commonly used

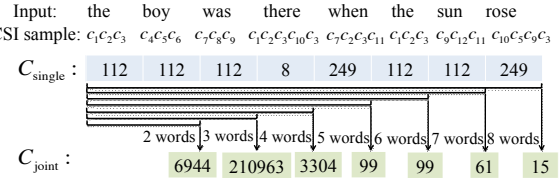


Fig. 6. The evolution of the amount of candidates returned.

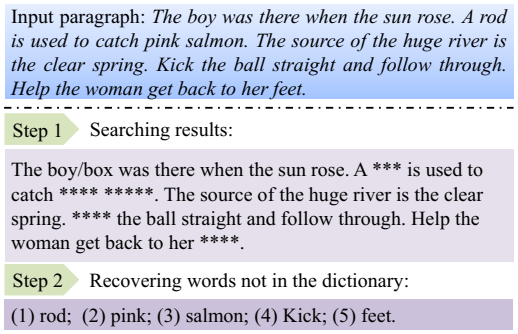


Fig. 7. Example paragraph recovery.

for testing speech recognition techniques. For this example recovery, we randomly select five sentences from these representative English sentences, with a total of 41 words. We record C_{single}, which is the number of words that have the same inter-element relationship matrix as the current CSI word group under processing, and C_{joint}, which is the number of candidates returned by the joint demodulation algorithm for each CSI word group.

Figure 6 shows C_{single} and C_{joint} during the processing of this sentence. To facilitate understanding, we also mark the CSI sample sets on this figure. For example, f_1, f_2, and f_3 represent the CSI sample sets caused by typing the letters 't', 'h', 'e', respectively. We can see that C_{single} is 112 for three-letter words, and consequently C_{joint} increases dramatically from 112 to 6,944 and then to 210,963 as the second and third CSI word groups are added, as these word groups share no common CSI samples. However, as more CSI word groups are added, the joint demodulation algorithm finds more common CSI samples, which shrinks the search space. C_{joint} drops sharply from 210,963 to 3,304 after the fourth CSI word group is processed, and further reduces to 15 as the remaining CSI word groups are processed.

The demodulation phase returns two candidates, as shown in Figure 7. They differ by only one word; the second word is either "boy" or "box". Even for the wrong candidate, 97.6% of the words are successfully recovered, and all characters except one. The example paragraph also contains five words ("rod", "pink", "salmon", "kick", and "feet") that are not in the dictionary. These are still successfully inferred, however, because their constituent CSI samples also appear in other words, and their sample/letter mappings have already been determined by the matching phase.

5.2 Eavesdropping Accuracy

We define the word recovery ratio as the ratio of successfully recovered words to the total number of input words. We employ this metric to ascertain the accuracy of our attack using 100 online articles randomly selected from CNN, New York Times, and Voice of America. For comparison, we also apply the traditional frequency analysis technique to the segmented CSI samples.

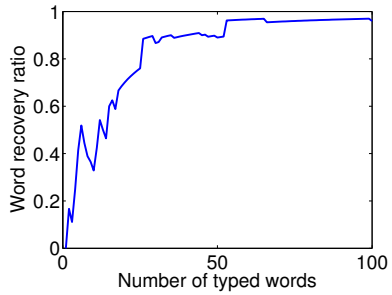


Fig. 8. WRR vs. word count

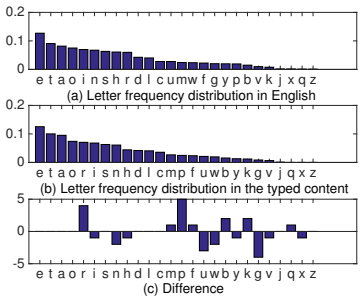


Fig. 9. Comparing distributions.

Typed content: *police indicated they knew the identity of the assailant who fatally stabbed unarmed police officer*

✧ (1) Recovery with frequency analysis

molnce nhdncated trep bheg tre ndehtntp ow tre assanlaht gro watallp stayyed fhaiued molnce owwncei

✧ (2) Recovery with the proposed attack

police indicated they knew the identity of the assailant who fatally stabbed unarmed police officer

Fig. 10. Recovered words.

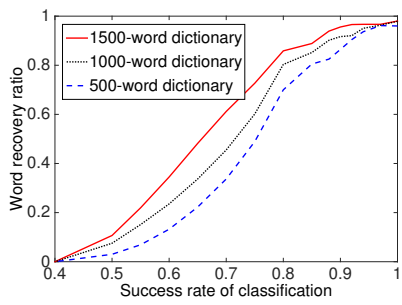


Fig. 11. WRR vs. classification errors.

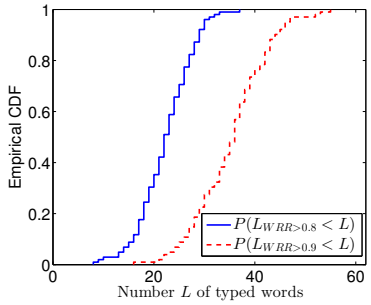


Fig. 12. CDFs of $L_{WRR>0.8}$ and $L_{WRR>0.9}$.

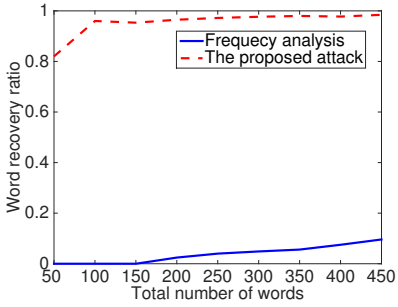


Fig. 13. Comparison with Frequency analysis.

5.2.1 *Single Article Recovery*

We first type a piece of CNN news [35] into a computer, and collect the CSI while typing. Suppose the demodulation algorithm returns N candidates for the typed content. We use WRR_i ($i \in \{1, \cdots, N\}$) to denote the word recovery ratio for the i^{th} candidate. We consider the overall word recovery ratio WRR of the proposed attack to be calculated as the average of these word recovery ratios for each candidate: $WRR = \sum_{i=1}^{N} \frac{WRR_i}{N}$.

Figure 8 shows the overall word recovery ratio as a function of the number of typed words. We can observe for the first couple of typed words, the ratio is less than 0.17, because these words are not in the dictionary or the joint demodulation algorithm returns wrong candidates. As more words are typed in, the ratio increases significantly and fluctuates, since newly typed words may or may not be identified correctly in the various candidates. After a sufficient number of words are typed, the mapping between CSI samples and the letters converges to only one candidate. As a result, the word recovery ratio stabilizes at a high value. As shown in Figure 8, when more than 52 words have been typed, the overall word recovery ratio remains above 0.96.

For meaningful results, we apply the frequency analysis recovery technique to compare with our method. Figure 9(a) shows the typical distribution of frequencies of English letters [15], while Figure 9(b) shows the distribution of letters in the typed text. Because the typed text is short and not representative of the whole English language, the sample distribution is not perfectly equal to the typical distribution. This difference is highlighted in Figure 9(c) and causes the word recovery ratio for the frequency analysis to be as low as 0.07. Figure 10 shows parts of the recovery results using the frequency analysis and our method. The content recovered using the frequency analysis is meaningless, whereas our new attack successfully recovers the typed words.

Impact of CSI sample classification errors and dictionary size: As discussed in Section 3.3.4, errors in grouping CSI samples during pre-processing may occasionally lead to a failure in demodulating a CSI word group when the word's pattern is not correctly detected. To test the impact of this on the overall word recovery ratio, we artificially introduce errors into the groupings and attempt the demodulation algorithm using the intentionally incorrect data. Specifically, we vary the number of correctly grouped CSI samples from 40% to 100% in intervals of 5%, and measure the resulting overall word recovery ratio. We also examine the effects of dictionaries of three different sizes, including the 500, 1000, and 1500 most frequently used words.

We repeat this experiment 10 times and present the average results in Figure 11. Intuitively, more correctly classified CSI samples result in higher word recovery ratios, as do larger dictionaries. Nonetheless, we also note that only 80% of CSI samples need to be correctly classified for the overall word recovery ratios to achieve 0.86, 0.81, and 0.7 for the various dictionary sizes.

5.2.2 *Average Article Recovery*

We repeat the above experiment for 100 online articles. Intuitively based on the discussed observations, the proposed attack should achieve a high word recovery ratio for a long text. Considering a desired overall word recovery ratio of 0.8 or 0.9, let $L_{WRR>0.8}$ and $L_{WRR>0.9}$ denote the required number of typed words from each article to satisfy those ratios, respectively. Figure 12 shows the empirical cumulative distribution functions (CDFs) of $L_{WRR>0.8}$ and $L_{WRR>0.9}$, indicating conclusively longer input text results in higher word recovery ratios. Specifically, for more than 82.4% of articles, the achieved word recovery ratio is greater than 0.8 and 0.9 when the number of these words is greater than 27 and 42, respectively.

Figure 13 compares the efficacy of our attack and the frequency analysis technique. Our attack can achieve a 0.82

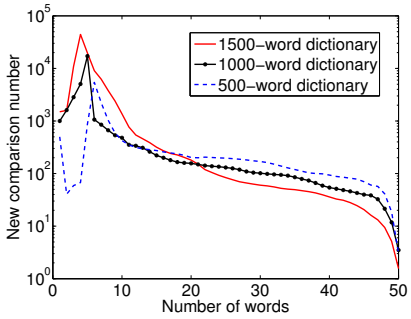


Fig. 14. New comparisons vs. word count.

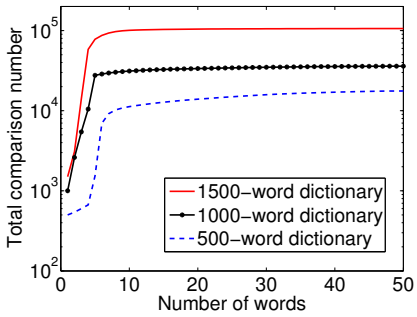


Fig. 15. Total comparisons vs. word count.

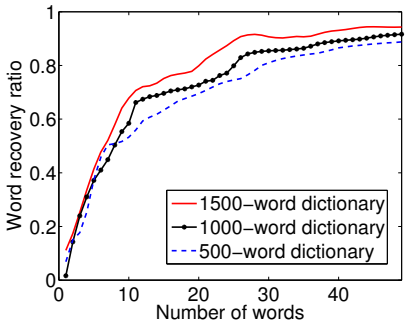


Fig. 16. Recovery of "secrets".

word recovery ratio after 50 typed words, whereas the frequency analysis requires typing 150 words before any can be successfully recovered. Indeed, the highest ratio achieved by the frequency analysis in these online articles is around 0.1, after 450 words, while in stark contrast our attack stabilizes around 0.95 after 150 words.

5.3 Time Complexity Analysis

The comparison of inter-element relationship matrices is the dominant part of the dictionary demodulation phase, so we count the required comparisons to calculate complexity. We use three different dictionaries, which contain the top 500, 1000, and 1500 most frequently used words [17].

During the 100 experiments in Section 5.2, we count the comparisons to generate candidates for the typed content each time a new CSI word group is added to the dictionary demodulation process. Figure 14 shows the average comparison number for each newly typed word, on a log scale. This number greatly increases for the first few typed words but promptly decreases to a low value below 10 as more words are typed. This was seen for a single sentence in Figure 6 and holds for these 100 trials as well. The addition of more unique letters results in a vastly enlarged search space, while the later inclusion of more repeated letters imposes a structure to the words and quickly reduces the search space.

Interestingly, the search space for a larger dictionary shrinks faster than that of a smaller dictionary as more words are typed, despite being larger after the first few words. For example, for the 45^{th} word, the average numbers of required comparisons for the 1500-, 1000-, and 500-word dictionaries are 20.6, 39.7, and 70.1, respectively. At first, a larger dictionary will find more matches for the word structures searched, but this quickly narrows down as repeated letters are added. Conversely, a smaller dictionary has a lower probability of finding candidate words for a particular structure, leading to skipped words, and therefore requiring more typed words before repeated letters can appear and reduce the search space.

Figure 15 shows the cumulative average comparison numbers as more words are typed. The time is clearly spent mostly on inferring the first couple of words, after which the total time complexity stabilizes. This trend is the same for all dictionaries, though larger dictionaries see distinctly more total comparisons and consequently higher time complexity. Larger dictionaries also stabilize faster, however; the 1500-, 1000-, and 500-word dictionaries stabilize at 8, 11, and 15 typed words, respectively.

5.4 An Example of the Attack

We recruited 10 volunteers and asked each to type a paragraph of "secret" content for us to attempt to infer. For ethical reasons, we did not ask them to type actual secrets that they would wish to keep private, but simply to type comprehensible English content which we did not provide them. While each volunteer typed, the receiver continuously collected CSI data and processed them. The eavesdropping result was presented to the volunteer, who compared the recovered content with their typed content to quantify the word recovery rate. Figure 16 shows the resulting average word recovery ratios as each word is typed and with the three different dictionary sizes. Our attack achieves a word recovery ratio of more than 0.8 after 28 words are typed, regardless of dictionary size. Additionally, a larger dictionary yields a higher word recovery ratio. With more than 40 typed words and a dictionary of 1,500 words, the ratio exceeds 0.94. This demonstrates our attack can recover typed secrets effectively and efficiently in a real-world setting.

5.5 Steal Source Code

Except for English, the proposed attack can also target other languages, e.g., source code. In this section, we demonstrate how the attack is applied to steal source code. Note that programs are written exclusively via a keyboard and are of interest to corporate espionage, etc. Without loss of generality, we utilize Linux kernel source code as an example.

To launch the proposed attack, we first explore the specific properties that Linux kernel source code has for inferring coding. A programming token (e.g., constant, identifier, operator, reserved words, separator) is the basic component of the source code. When typing codes, we often use the space character to separate tokens in a line, and use the semicolon character to close an expression and a line. To begin a new line, the "Enter" key is pressed. Besides, the parentheses characters are often used to indicate function calls and function parameters. Based on those properties, we first try to identify the space, semicolon, "Enter", and parentheses characters, and use them as token dividers.

First, we build a token dictionary that includes C language keywords in ANSI C, extended keywords which do not exist in ANSI C, and function names of kernel modules (e.g., printk, init_module, cleanup_module) [36]. Note that each language should have its specific dictionary.

We select a piece of source code, as shown in Figure 17, and let the user type it. The receiver continually collects

Typed code:

```
int init_module(void) {
    printk(KERN_INFO "Hello World! \n");
    return 0;
}

void cleanup_module(void) {
    printk(KERN_INFO "Goodbye world! \n");
}
```

Recovered code:

```
int init_module(void) {
    printk(KERN_INFO **ello *orld* *n*);
    return *;
}

void cleanup_module(void) {
    printk(KERN_INFO **ood**e *orld* *n*);
}
```

Fig. 17. Source code inference example ('*' is an unidentified character).

the CSI samples and processes them. Based on the aforementioned rule of source code, we first identify the CSI samples corresponding with the space, semicolon, "Enter", and parentheses characters from the observed CSI samples. With such token divider mapped CSI samples, we divide the CSI sample stream into separate CSI token groups. Next, the dictionary demodulation is launched. The recovered code is presented in Figure 17. We see that our attack can successfully recover all tokens that are in the pre-established dictionary. Meanwhile, for tokens that are not in the dictionary, the proposed may still recover the characters in them as long as the characters also appear in the identified tokens.

5.6 Password Entropy Reduction

Modern passwords include letters, numbers, and special characters. The password strength lies in its resistance to brute-force attacks. Our attack focuses on letters, but it can still greatly decrease password strength. As users normally type both passwords and English content during computer usage, we can apply the alphabet matching afforded by the latter to infer significant portions of the former.

Typical users usually pick fewer non-letter characters in their password to make it easy to remember, leaving the password more vulnerable to these attacks. We did preliminary experiments to evaluate the entropy reduction impact using the password list, which contains 342,508 passwords leaked from Yahoo! Voices [37]. Figure 18 shows the average ratios of letter characters in passwords with different lengths. We observe that the ratio of letters in a password with a length ranging from 6 to 12 lies between 0.65 to 0.73, and also with the key length increasing, the ratio of letters slightly increases. We find that 98.42% of the leaked passwords are 12 characters or fewer, and people utilize an average of 8.72 letters for a 12-character password. This means that the difficulty for guessing a 12-character random password is reduced to that for guessing an extremely weak password of 3-4 characters. Furthermore, the attacker knows these 3-4 characters are not English letters.

We quantify the damage our attack can inflict on password entropy, the typical measure of password strength. The entropy of a password X is defined as $H(X) = -\sum_{i=1}^{n} P(x_i)\cdot\log_2 P(x_i)$, where x_i ($i \in \{1, 2, \cdots, n\}$) is one of n possible values of X and $P(x_i)$ represents the probability that $X = x_i$ holds. Considering a keyboard housing N characters, a password with length l selected at random has

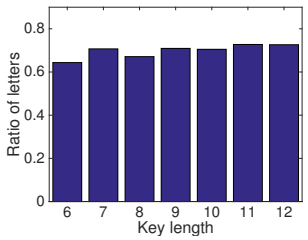


Fig. 18. Ratio of letters vs. key length.

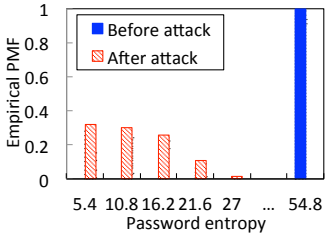


Fig. 19. The PMFs of password entropies.

N^l possible values and $l\cdot\log_2 N$ bits of entropy. Suppose this password has l' letter characters and $l - l'$ non-letter characters. The keyboard with N characters necessarily contains 26 letters and $N-26$ non-alphabetical characters. Having successfully established a full CSI sample/letter mapping and applying this mapping to the CSI samples comprising the password, its entropy becomes $(l-l')\log_2(N-26)$ bits.

In our experiment, we randomly select 1000 9-character passwords from the Yahoo! Voices dataset. 32 non-alphanumerical characters are allowed in passwords, yielding 42 non-alphabetical characters when factoring in numbers. However, we find that an average of 6.38 of 9 characters were letters, meaning their discovery will vastly reduce entropy. Each of these 1000 passwords was added to the end of the text typed by volunteers in the previous experiment, and the resulting CSI sample/letter map was applied to each. We compare the inferred password information to the original password, to identify the correctly recovered characters and calculate the difference in password entropy.

Figure 19 plots the empirical probability mass functions (PMFs) of the password entropies before and after the proposed attack is applied. A randomly selected 9-character password with the assumed keyboard layout provides 54.8 bits of entropy and requires a maximum of 31.08 quadrillion brute force attempts. With our attack, the password entropy can be decreased to within a range of 5.4 to 27.0 bits, such that breaking a 9-character password is reduced to guessing 1-5 non-letter characters. The maximum number of brute-force attack attempts targeting a password with an entropy of 5.4 bits is just 42. In fact, 89.0% of the randomly selected passwords have less than 16.2 bits of entropy after our attack, meaning at most 74,000 brute-force attack attempts are required for the vast majority of these passwords. Evidently, the security of these passwords is decreased by several orders of magnitude courtesy of the proposed attack.

5.7 Evaluation of TypeGuard

To evaluate the effectiveness of *TypeGuard* against our attack, we add a third USRP X300 as the jammer, which starts to transmit noise signals when it detects the type event, and stops when it detects that the typing session ends. We consider two scenarios: (1) when the user types without other moving objects around; and (2) when the user types with other users moving around from time to time.

Figure 20 presents an example of the pre-processed CSI waveforms with and without *TypeGuard*, where the user types a word "apple". We can see that at the time of 1.8 seconds, *TypeGuard* initiates, and the jamming signals successfully obscure the keystroke associated patterns in the CSI waveform, demonstrating the effectiveness of *TypeGuard*.

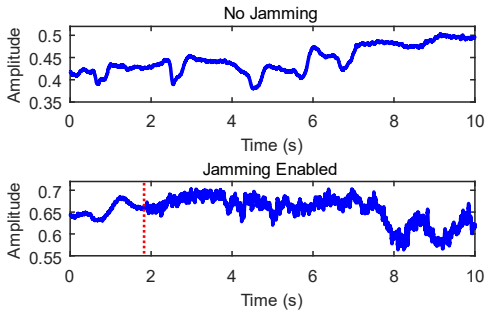


Fig. 20. An example of observed CSI waveforms after pre-processing at the eavesdropper with and without *TypeGuard*.

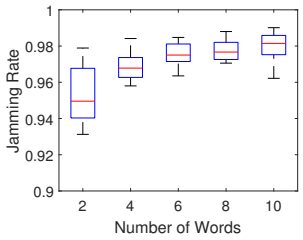


Fig. 21. Jamming rate δ vs. N (without nearby interference).

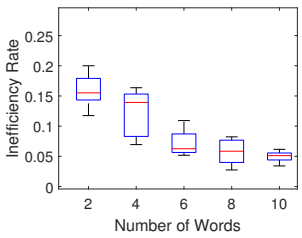


Fig. 22. Inefficiency rate β vs. N (without nearby interference).

Besides, to further evaluate effectiveness and efficiency, we utilize the following two metrics respectively:

- *Jamming rate* δ: We define $\delta = \frac{J_e}{T}$, where J_e denotes the duration of the jammed portion of a typing session, and T represents the entire duration of the typing session. A higher δ indicates that the eavesdropper would observe less useful CSI information.
- *Inefficiency rate* β: We define $\beta = \frac{J_{ie}}{J}$, where J_{ie} is the duration of the jammed portion of the non-typing period, and J denotes the whole jamming duration (i.e., $J = J_{ie} + J_e$). Since no jamming is needed for non-typing periods, a lower β then implies that the jamming scheme is more efficient.

We let the user type an English sentence with N words for each typing session. N varies from 2 to 10 with increments of 2. The user types each sentence 10 times. With recorded durations T, J_e, and J_{ie} for each typing session, we compute corresponding jamming rate δ and inefficiency rate β.

Figures 21 and 22 present δ and β across different word count N in the environment without inference from nearby movement. We can see that the jamming rate is always above 0.93, and the average jamming rate is 0.97 for all typing sessions. Also, the median jamming rate slightly increases with the word count. This is because *TypeGuard* normally only leaves the first keystroke waveform unjammed and thus a longer sentence would lead to a larger jamming rate. Besides, we can see that the median inefficiency rate across different word count lies in the range of 0.05 to 0.14. We observe that hand movement not for typing may also trigger the jamming by accident. Such incidents would increase the value of β. Overall, the average inefficiency rate decreases with the word count. These results show that *TypeGuard* can effectively and efficiently disrupt the signal reception at the eavesdropper, and thus successfully defend against the proposed wireless keystroke inference attack.

Figures 23 and 24 depict measured δ and β over different N in the environment with interference from nearby

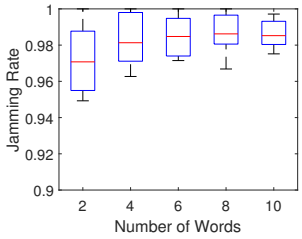


Fig. 23. Jamming rate δ vs. N (with nearby interference).

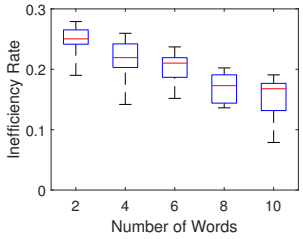


Fig. 24. Inefficiency rate β vs. N (with nearby interference).

movement. We can see that the jamming rate is at least 0.95, and the average jamming rate is 0.98 for all typing sessions, slightly higher than the case without the inference. This appears because the interference introduced by nearby movement may trigger *TypeGuard* to even jam the first keystroke waveform. Meanwhile, compared with the case without the interference, the median inefficiency rates for all N are consistently increased, ranging from 0.16 to 0.25. These increases are caused by false triggering of *TypeGuard* brought by the nearby movement-induced interference.

6 Limitations

Environmental Movement: Usually, a user is more focused with less body movement during typing. The movement of the typist or other movements in the environment may bring the variation of CSI, and thus introduce interference for CSI associated with keystrokes. This is a general issue to all wireless-based keystroke inference attacks. There are several tolerance methods to reduce the impact of human movements. For example, [3] applies noise reduction algorithms to improve keystroke recognition accuracy. Also, unlike omni-directional antennas which have a uniform gain in each direction, directional antennas have a different antenna gain in each direction. Thus, [4] adopts directional antennas to eliminate CSI noises introduced by environmental non-keystroke-induced movement. Besides, the eavesdropping device can be placed close to the target keyboard, e.g., under the victim's desk, to reduce surrounding impact.

Auto-correction and Auto-complete: Auto-correction uses a dictionary to spellcheck typed words and correct misspelled ones; auto-complete predicts the rest of a word that a user types. In both cases, due to inefficient CSI information, the attacker is often unable to directly demodulate the incomplete CSI word group via comparing inter-element relationship matrices. However, as no candidates match, the formed CSI word group would be added to the undermodulated set. When the sample/letter mappings are built, they can be applied to the CSI word groups in the undemodulated set. With the recovered misspelled word or partial letters of the word, the attacker can further utilize the public auto-correction and auto-complete applications to infer the exact word that the user intends to input.

7 Related Work

Existing non-invasive attacks to infer keystrokes fall into the following categories:

Vibration based attacks: Typing on a keyboard can cause vibrations on the surface where the keyboard rests, with subtle differences depending on keys typed [5], [6]. The

accelerometer of a nearby phone or tablet on the same surface can capture the vibrations. With training, an attacker can establish the relationship between the keystroke and the acceleration disturbance caused by the vibration. In the detection phase, the attacker can then recover the typed content by applying this relationship.

Acoustic signal based attacks: It has been observed typing on a keyboard can produce sounds unique to each key. Researchers extract features from these sounds and then train a classifier to reconstruct the keystrokes [7]–[9], [13]. The requirement for training is relaxed in [8], which uses a statistical unsupervised training method to design a supervised classifier. However, the proposed method is faster than the method based on the Hidden Markov Model (HMM) in [8]. The HMM method requires collecting 10 minutes worth of keystrokes (around 340 words) for a word recovery rate of 87.6%. This minimized training method may not function for wireless based attacks, as due to the time-varying nature of the wireless channel, a training time of 10 minutes may be too long to generate a useful mapping between observed CSI samples and letters. Unlike [8], frequency analysis, and all other statistical methods, the proposed method explores the self-contained structures of words, which can be observed for each word immediately as it is typed, rather than probabilistic statistics among words, which require many words to establish. Thus, the proposed attack only needs 50 words within 1-2 minutes for a word recovery rate of 94.3%.

Zhou *et al.* discovers that the recorded audio signals can be used to infer the victim's finger movement and thus crack Android pattern locks [38]. Such an attack, however, is invasive as it requires installing malware on the victim's smartphone. An adversary may use a triangulation localization technique to localize the sound source and accordingly infer keystrokes [39], [40]. This approach, however, requires sophisticated equipment to precisely measure the sound propagation distance, and also requires line-of-sight between the keyboard and equipment. Both requirements hinder attack plausibility and application. Also, the attenuation of acoustic signals can be used to localize each keystroke during inputs [41]. However, this method requires placing a smartphone close (within 60 cm) to the victim's keystroke, and the alignment between the devices of the adversary and the victim affects the accuracy of keystroke localization. Berger *et al.* infer keystrokes with the observation that similar sounds are highly likely to come from keys positioned close to each other on the keyboard [42]. This technique aims to reconstruct a single long (7-13 characters) word in the dictionary, whereas the goal of the proposed attack is to reconstruct the entire typed content regardless of whether or not all its constituent words are in the dictionary.

Timing based attacks: Keystroke timing patterns can be another source to infer keystrokes [10]–[12]. For example, [10] infers keystroke sequences by using the inter-keystroke timing information collected from the arrival times of the SSH packets. However, these timing-based attacks all require a training process to statistically generate the attack models.

Wireless signal based attacks: There are emerging research efforts performing keystroke eavesdropping attacks using wireless signals due to the ubiquitous deployment of wireless infrastructures, the radio signal nature of invisibility, and the elimination of the line-of-sight requirement. In particular, [2] infers keystrokes by examining the amplitude and phase changes of the wireless signal; [3], [4], [14] utilize the channel condition extracted from the observed wireless signal to distinguish keystrokes; [43] proposes an LTE-based keystroke inference attack, which has a longer operational distance than previous attacks via WiFi signals. All these works require training to construct the relationship between the observed signal feature and the typing.

Camera-based attacks: An intuitive method to infer keystrokes is to use cameras to record the typing process and then identify keystrokes by analyzing the recorded video. Researchers have found that video recording of hand movement [44]–[46], eye movement [47], tablet backside motion [48], or the shadow around fingertips [49], is also able to aid the keystroke inference. However, when the movement of interest does not happen in the presence of a camera, keystroke activities cannot be detected.

Cryptanalysis based attacks: Cryptanalysis is a technique of discovering secrets. Cryptanalysis attacks can be in the form of known-plaintext or ciphertext-only attacks. If we consider the CSI sample as the ciphertext and the typed content as the plaintext, the training-based keystroke inference attacks [3], [4] are indeed known-plaintext attacks, as the attacker must know some plaintext (i.e., typed content) and the corresponding ciphertext (i.e., CSI) for training. Our attack does not require training data. Thus it is a ciphertext-only attack. Existing ciphertext-only attacks attempting to decode the ciphertext of natural language are largely based on the statistical information about the ciphertext [50], [51]. For example, [50] regards the author of an instant message conversation as the plaintext and applies character frequency analysis to instant messages for authorship identification. [51] recovers the plaintext by using a statistical language model and a dynamic programming algorithm.

Nevertheless, collecting statistical information implies acquiring a large amount of ciphertext. This may not be suitable for the wireless based keystroke inference, because collecting the wireless statistics does require a long period of observation. As mentioned earlier, this can prevent the attacker from collecting sufficient reliable statistics for keystroke inference. Our method is based on the self-contained feature of words instead and thus does not require the long-time observation about wireless statistics.

8 Conclusion

We identify a new type of keystroke eavesdropping attack. Compared with all previously discovered attacks, the attack can bypass (1) the requirement of the training phase, (2) the requirement to deceive the user or bypass the user's antivirus and firewall software to install malware on the target device, and (3) the requirement of line-of-sight between the attacker's device and the keyboard. This attack is constructed based on the CSI extracted from the wireless signal. We also propose defense techniques against this attack. We implement the discovered attack and the developed defense called *TypeGuard* on the USRP X300 platform, and conduct experiments to validate both. Experiment results demonstrate the feasibility of the proposed attack to infer English words and source code, as well as the effectiveness and efficiency of *TypeGuard* against the attack.

ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation under Grant No.1948547.

REFERENCES

[1] S. Fang, I. Markwood, Y. Liu, S. Zhao, Z. Lu, and H. Zhu, "No training hurdles: Fast training-agnostic attacks to infer your typing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (Toronto, Canada), pp. 1747–1760, ACM, 2018.

[2] B. Chen, V. Yenamandra, and K. Srinivasan, "Tracking keystrokes using wireless signals," in *Proc. of the 13th Annual International Conf. on Mobile Systems, Applications, and Services*, MobiSys '15, pp. 31–44, ACM, 2015.

[3] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using WiFi signals," in *Proc. of the 21st Annual International Conf. on Mobile Computing and Networking*, MobiCom '15, pp. 90–102, ACM, 2015.

[4] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan, "When CSI meets public WiFi: Inferring your mobile phone password via WiFi signals," in *Proc. of the 23Nd ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pp. 1068–1079, 2016.

[5] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iPhone: Decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proc. of the 18th ACM Conf. on Computer and Communications Security (CCS)*, pp. 551–562, 2011.

[6] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "ACCessory: Password inference using accelerometers on smartphones," in *Proc. of the Twelfth Workshop on Mobile Computing Systems and Applications*, HotMobile '12, pp. 9:1–9:6, ACM, 2012.

[7] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *Proc. of the IEEE Symposium on Security and Privacy*, pp. 3–11, 2004.

[8] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," in *Proc. of the 12th ACM Conf. on Computer and Communications Security (CCS)*, pp. 373–382, 2005.

[9] J. Wang, K. Zhao, X. Zhang, and C. Peng, "Ubiquitous keyboard for small mobile devices: Harnessing multipath fading for fine-grained keystroke localization," in *Proc. of the 12th Annual International Conf. on Mobile Systems, Applications, and Services*, MobiSys '14, pp. 14–27, ACM, 2014.

[10] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on SSH," in *Proc. of the 10th Conf. on USENIX Security Symposium - Volume 10*, SSYM'01, 2001.

[11] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. of the 16th ACM Conf. on Computer and Communications Security (CCS)*, pp. 199–212, 2009.

[12] K. Zhang and X. Wang, "Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems," in *Proc. of the 18th Conf. on USENIX Security Symposium*, SSYM'09, pp. 17–32, 2009.

[13] A. Compagno, M. Conti, D. Lain, and G. Tsudik, "Don't skype & type!: Acoustic eavesdropping in voice-over-ip," in *Proc. of the 2017 ACM on Asia Conf. on Computer and Communications Security*, ASIA CCS '17, pp. 703–715, 2017.

[14] Z. Zhang, N. Avazov, J. Liu, B. Khoussainov, X. Li, K. Gai, and L. Zhu, "WiPOS: A POS terminal password inference system based on wireless signals," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7506–7516, 2020.

[15] J. Katz and Y. Lindell, *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[16] "Statistical distributions of English text." http://www.data-compression.com/english.html, 2017.

[17] M. Davies, "Word frequency data from the Corpus of Contemporary American English (COCA)." http://www.wordfrequency.info/free.asp, 2017.

[18] Q. Pu, S. Gupta, S. Gollakota, and S. Patel, "Whole-home gesture recognition using wireless signals," in *Proc. of the 19th Annual International Conf. on Mobile Computing and Networking*, MobiCom '13, pp. 27–38, ACM, 2013.

[19] S. Fang, Y. Liu, W. Shen, and H. Zhu, "Where are you from? confusing location distinction using virtual multipath camouflage," in *Proceedings of the 20th annual international conference on Mobile computing and networking*, pp. 225–236, 2014.

[20] F. Adib and D. Katabi, "See through walls with WiFi!," in *Proc. of the 2013 ACM Conf. on SIGCOMM*, pp. 75–86, 2013.

[21] F. Adib, C.-Y. Hsu, H. Mao, D. Katabi, and F. Durand, "Capturing the human figure through a wall," *ACM Trans. Graph.*, vol. 34, pp. 219:1–219:13, Oct. 2015.

[22] S. Fang, I. Markwood, and Y. Liu, "Manipulatable wireless key establishment," in *2017 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, IEEE, 2017.

[23] A. Goldsmith, *Wireless Communications*. New York, NY, USA: Cambridge University Press, 2005.

[24] G. Wang, Y. Zou, Z. Zhou, K. Wu, and L. M. Ni, "We can hear you with Wi-Fi!," in *Proc. of the 20th Annual International Conf. on Mobile Computing and Networking*, MobiCom '14, pp. 593–604, ACM, 2014.

[25] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, pp. 561–580, Oct. 2007.

[26] J. Wang and D. Katabi, "Dude, where's my card?: RFID positioning that works with multipath and non-line of sight," in *Proc. of the 2013 ACM Conf. on SIGCOMM*, pp. 51–62, 2013.

[27] S. Kumar, E. Hamed, D. Katabi, and L. Erran Li, "Lte radio analytics made easy and accessible," in *Proc. of the 2014 ACM Conf. on SIGCOMM*, pp. 211–222, 2014.

[28] S. Fang, Y. Liu, and P. Ning, "Wireless communications under broadband reactive jamming attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 3, pp. 394–408, 2016.

[29] K. Pärlin, M. M. Alam, and Y. Le Moullec, "Jamming of UAV remote control systems using software defined radio," in *2018 International Conference on Military Communications and Information Systems (ICMCIS)*, pp. 1–6, IEEE, 2018.

[30] SparkFun Electronics, "Sparkfun transceiver breakout - nrf24l01+ (rp-sma)." https://www.sparkfun.com/products/705, 2022.

[31] M. Ettus, *USRP user's and developer's guide*. Ettus Research LLC, 2005.

[32] J. Shlens, "A tutorial on principal component analysis," *CoRR*, vol. abs/1404.1100, 2014.

[33] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals & Systems (2Nd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[34] I. S. on Subjective Measurements, "IEEE recommended practice for speech quality measurements," *IEEE Transactions on Audio and Electroacoustics*, vol. 17, pp. 227–246, Sep 1969.

[35] "London attack: Assailant shot dead after 4 killed near Parliament." http://www.cnn.com/2017/03/22/europe/uk-parliament-firearms-incident/index.html, 2017.

[36] P. J. Salzman, *The Linux Kernel Module Programming Guide*. Paramount, CA: CreateSpace, 2009.

[37] "2012 yahoo! voices hack." https://en.wikipedia.org/wiki/2012_Yahoo!_Voices_hack, 2017.

[38] M. Zhou, Q. Wang, J. Yang, Q. Li, P. Jiang, Y. Chen, and Z. Wang, "Stealing your android patterns via acoustic signals," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1656–1671, 2021.

[39] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free attacks using keyboard acoustic emanations," in *Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pp. 453–464, 2014.

[40] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, and M. Gruteser, "Snooping keystrokes with mm-level audio ranging on a single phone," in *Proc. of the 21st Annual International Conf. on Mobile Computing and Networking*, MobiCom '15, pp. 142–154, ACM, 2015.

[41] J. Yu, L. Lu, Y. Chen, Y. Zhu, and L. Kong, "An indirect eavesdropping attack of keystrokes on touch screen through acoustic sensing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 337–351, 2021.

[42] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *Proc. of the 13th ACM Conf. on Computer and Communications Security (CCS)*, pp. 245–254, 2006.

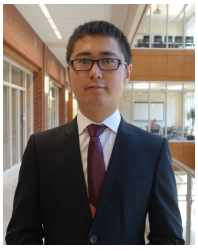
[43] K. Ling, Y. Liu, K. Sun, W. Wang, L. Xie, and Q. Gu, "Spidermon: Towards using cell towers as illuminating sources for keystroke monitoring," in *IEEE Conf. on Computer Communications (INFOCOM)*, pp. 666–675, 2020.

[44] D. Balzarotti, M. Cova, and G. Vigna, "Clearshot: Eavesdropping on keyboard input from video," in *Proc. of the IEEE Symposium on Security and Privacy*, pp. 170–183, 2008.

[45] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha, "Beware, your hands reveal your secrets!," in *Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pp. 904–917, 2014.

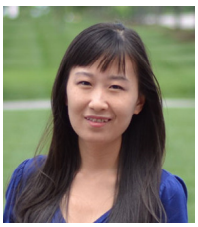
[46] Q. Yue, Z. Ling, W. Yu, B. Liu, and X. Fu, "Blind recognition of text input on mobile devices via natural language processing," in *Proc. of the 2015 Workshop on Privacy-Aware Mobile Computing*, PAMCO '15, pp. 19–24, ACM, 2015.
[47] Y. Chen, T. Li, R. Zhang, Y. Zhang, and T. Hedgpeth, "Eyetell: Video-assisted touchscreen keystroke inference from eye movements," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 144–160, 2018.
[48] J. Sun, X. Jin, Y. Chen, J. Zhang, R. Zhang, and Y. Zhang, "VISIBLE: Video-assisted keystroke inference from tablet backside motion," in *Proc. of the 23th Annual Network and Distributed System Security Conf.*, NDSS '16, The Internet Society, 2016.
[49] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao, "Blind recognition of touched keys on mobile devices," in *Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pp. 1403–1414, 2014.
[50] A. Orebaugh, "An instant messaging intrusion detection system framework: Using character frequency analysis for authorship identification and validation," in *Proc. of the 2006 International Carnahan Conf. on Security Technology*, pp. 160–172, Oct 2006.
[51] J. Mason, K. Watkins, J. Eisner, and A. Stubblefield, "A natural language approach to automated cryptanalysis of two-time pads," in *Proc. of the 13th ACM Conf. on Computer and Communications Security (CCS)*, pp. 235–244, 2006.

Edwin Yang received his M.S. degree from Yonsei University, Seoul, Korea, in 2017. He is working toward the Ph. D. degree in Computer Science at the University of Oklahoma. His research interests are in the area of mobile system security and Internet-of-things (IoT) security.



Song Fang received his Ph.D. in computer science from the University of South Florida in 2018. He is now an assistant professor in the School of Computer Science, University of Oklahoma. His research interests include wireless and mobile system security, cyber physical systems and IoT security, and mobile computing. He is also interested in applying machine learning in security.

Ian Markwood received his Ph.D. in computer science from the Univ. of South Florida in 2018. He is now a security researcher with BlackHorse Solutions, contracting for the US Government.



Yao Liu received her Ph.D. in computer science from the North Carolina State Univ. in 2012. She is now an associate professor at the Dept. of Computer Science and Engineering, Univ. of South Florida. Her research is related to computer and network security, with an emphasis on designing and implementing defense approaches that protect emerging wireless technologies from being undermined by adversaries. Her research interest also lies in cyber-physical systems security, especially smart grid security.

Shangqing Zhao received his Ph.D. in electrical engineering from the Univ. of South Florida in 2021. He is now an assistant professor in the School of Computer Science, Univ. of Oklahoma (Tulsa campus). His research primarily focuses on novel mobile system design, mobile and network security. His recent research is equally focused on machine learning for network and security applications.

Zhuo Lu is an Associate Professor in the Department of Electrical Engineering, University of South Florida. He received his Ph.D. degree in computer engineering from North Carolina State University, Raleigh NC, in 2013. His research interests include network science, cyber security, data analytics, cyber-physical systems, mobile computing and wireless networking. He is a member of IEEE, ACM, and USENIX.

Haojin Zhu received the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Canada, in 2009. He is now a professor with the Dept. of Computer Science and Engineering, Shanghai Jiao Tong Univ., China. His current research interests include network security and privacy-enhancing technologies. He is a senior member of IEEE and a member of ACM.