Acoustic Keystroke Leakage on Smart Televisions

Tejas Kannan
University of Chicago
tkannan@uchicago.edu

Synthia Qia Wang
University of Chicago
qiaw@uchicago.edu

Max Sunog
University of Chicago
msunog@uchicago.edu

Abraham Bueno de Mesquita
University of Chicago Laboratory Schools
abebdm@uchicago.edu

Nick Feamster
University of Chicago
feamster@uchicago.edu

Henry Hoffmann
University of Chicago
hankhoffmann@cs.uchicago.edu

***Abstract*—Smart Televisions (TVs) are internet-connected TVs that support video streaming applications and web browsers. Users enter information into Smart TVs through on-screen virtual keyboards. These keyboards require users to navigate between keys with directional commands from a remote controller. Given the extensive functionality of Smart TVs, users type sensitive information (e.g., passwords) into these devices, making keystroke privacy necessary. This work develops and demonstrates a new side-channel attack that exposes keystrokes from the audio of two popular Smart TVs: Apple and Samsung. This side-channel attack exploits how Smart TVs make different sounds when selecting a key, moving the cursor, and deleting a character. These properties allow an attacker to extract the number of cursor movements between selections from the TV's audio. Our attack uses this extracted information to identify the likeliest typed strings. Against realistic users, the attack finds up to 33.33% of credit card details and 60.19% of common passwords within 100 guesses. This vulnerability has been acknowledged by Samsung and highlights how Smart TVs must better protect sensitive data.**

I. Introduction

Internet-connected television (TV) devices have experienced massive growth over the last decade. These devices, called "Smart TVs", are expected to reach over 266 million units sold globally by 2025 [36]. Unlike their traditional counterparts, Smart TVs allow users to browse the Web, access video streaming applications (e.g., Hulu), and purchase products. This increased functionality brings about new security risks [1], [21], [35], [40], [42], [69].

Users typically enter information into Smart TVs through on-screen virtual keyboards. Common platforms, such as AppleTVs [4] and Samsung Smart TVs [55], allow users to navigate these keyboards with a hardware remote controller containing a direction pad (Figure 1). Users type by issuing directional commands to sequentially move a cursor between desired keys. Given the wide-ranging capabilities of Smart TVs, users enter sensitive information, such as passwords and credit card details, with these virtual keyboards [21]. Thus, Smart TVs must ensure the privacy of user keystrokes. On popular Smart TV platforms, such as Samsung's Tizen [55], the default keyboard makes sounds as users type.

This work presents a new side-channel attack against Smart TV keyboards that uses the TV's audio to discover typed strings. In this attack, audio forms a side-channel because the TV does not intend to convey the user's exact keystrokes through its acoustic feedback. Developing this attack requires answering four key questions:

(Q1) How can the attack identify sounds made by the Smart TV that contain valuable information about keystrokes?
(Q2) How can the attack use these sounds to discover when a user is typing and how they navigate the virtual keyboard?
(Q3) How can the attack use the extracted movement information to recover user-typed strings?
(Q4) How can the attack overcome real-world challenges, such as discovering the type of entered information (e.g., passwords, credit cards, etc.) and accounting for user typing behavior?

Our attack addresses the first question (Q1) by exploiting how different Smart TV platforms make distinct sounds. Further, platforms use a small number of sounds, and each sound is consistent because it comes from the platform itself. These properties allow the attack to identify important sounds by pre-recording sounds from common Smart TVs and using Fourier analysis to match these references against the sounds observed at runtime. This method augments standard signal processing techniques to handle distinct features of Smart TVs, such as the conflation of adjacent sounds due to rapid movements on the keyboard.

We answer the second question (Q2) by drawing on the insight that multiple popular Smart TVs make *different* sounds for the following actions: (1) moving the keyboard cursor, (2) selecting a key, and (3) deleting a character. Furthermore, compared to other actions on the TV, the key selection sound is unique to the keyboard. Thus, an adversary with auditory access can learn three critical pieces of information about user

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24072
www.ndss-symposium.org

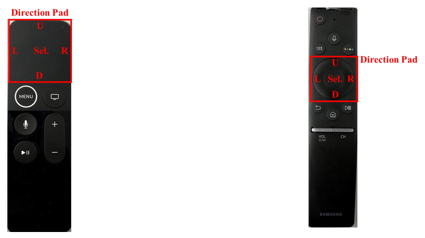

Fig. 1. Remote controllers for AppleTVs (left) and Samsung Smart TVs (right) annotated with the direction pad.

keystrokes.

- *When* a user is typing, due to sounds unique to the keyboard.
- The *length* of the entered string, by tracking the number of selections and deletions.
- The *number* of cursor movements between selections.

The TV's audio, however, does not reveal the *direction* of user movements on the virtual keyboard (Q3). This lack of directionality means that many possible strings may correspond to the audio for a given sequence of keyboard movements. We address this challenge by using a prior dictionary to measure the probability of typing each possible string. This design allows the attack to construct a ranked list of the likeliest results across all possible movement directions. We further leverage user typing patterns and string-length properties to infer the class of entered information, such as passwords or credit cards (Q4). This feature allows the attack to customize the prior without making additional assumptions.

Smart TV keyboards do not require users to take the shortest path between keys, so the extracted movement counts only provide an upper bound on the distance between selections (Q4). Accounting for suboptimal paths that users may traverse increases the search space and complicates keystroke recovery. We tackle this problem by exploiting user behavior through keystroke timing: we observe that users tend to pause when correcting suboptimal paths. By identifying such pauses, the attack only accounts for suboptimal paths when needed.

We evaluate this attack against two popular Smart TV brands: Apple [4] and Samsung [55]. These brands account for about 15.4% of the global Smart TV market, with Samsung being the most popular platform [66]. Importantly, the audio profiles of both TVs exactly match the properties enabling our attack. When assuming optimal typing behavior in an emulated environment, our attack recovers 99.95% of credit card numbers (CCNs), 97.65% of full credit card details (CCN, security code, ZIP code, and expiration date), and up to 99.10% of common passwords [41] within 100 guesses. We extend these results to a realistic setting through a user study[1]. For ten subjects typing into real applications on a Samsung TV, the attack recovers 53.33% of CCNs, 33.33% of full credit card details, and up to 60.19% of common passwords [41] within 100 guesses. We further investigate this attack against users typing passwords on an AppleTV. On this platform, the attack has a top-100 password recovery accuracy of 31.00%; this lower result occurs because users take more suboptimal paths on this device. These top-100 recovery rates mean our attack can verify the extracted information against rate-limited online services [32].

[1]Our university's institutional review board (IRB) approved this study.

Prior work has studied side-channels on user keystrokes from two angles with similar elements. First, multiple attacks use the audio from mechanical keyboards to infer keystrokes [6], [10], [14], [70]. These attacks, however, do not work on Smart TVs due to the differences in keyboard types. Mechanical keyboards allow users to jump between keys instantaneously. In contrast, Smart TVs require users to scroll across the virtual keyboard through directional commands. This difference in dynamics necessitates a new method for string recovery. Second, HomeSpy sniffs unencrypted infrared (IR) signals between Smart TVs and their remote controllers, enabling keystroke recovery [21]. Recent Smart TVs, however, have remotes that do *not* use IR [35], [69], and the HomeSpy attack does not succeed on such devices. Our acoustic attack is agnostic to the TV remote's communication medium.

We disclosed this issue to Apple and Samsung. Apple responded, "while we do not see any security implications, we have forwarded your report to the appropriate team to investigate as a potential enhancement request to take action." The reply provides no further details. We followed up on the enhancement request to confirm our suggestion to mute the TV when typing sensitive information. We have not heard back. Samsung acknowledged the presented security problem and paid a bounty for finding this vulnerability. They further confirmed they have "multiple teams discussing the issue" to implement a long-term fix.

Overall, we make the following contributions:

1) We identify a new side-channel attack against Smart TVs that uses audio to learn about user keystrokes.
2) We build an attack framework that uses acoustic information to identify instances of keyboard activity and extract cursor move counts between keyboard selections.
3) We create a recovery module that infers user-typed strings from the information extracted from Smart TV audio.
4) We identify and exploit common human typing behaviors on Smart TVs to improve the attack in practice.

This work identifies a novel side-channel on Smart TVs that leaks sensitive user information. This result displays another example of how Internet-connected, everyday devices must pay more attention to security [2], [16], [19], [20].

II. Background

This section provides background on Smart TVs and virtual keyboards (§II-A) before describing the notation we use in the rest of the paper (§II-B). We then discuss details of credit card transactions (§II-C) and present an example of acoustic keystroke leakage (§II-D).

A. Smart TVs and Virtual Keyboards

Smart TVs are Internet-connected televisions that support web browsers and third-party applications. Users control Smart

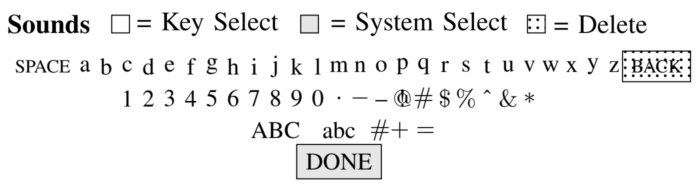

Fig. 2. The default AppleTV keyboard for passwords. The background shows the key's sound when pressed. The only exception is `Done`, which makes the sound when scrolling onto the key; selecting `Done` makes no sound.

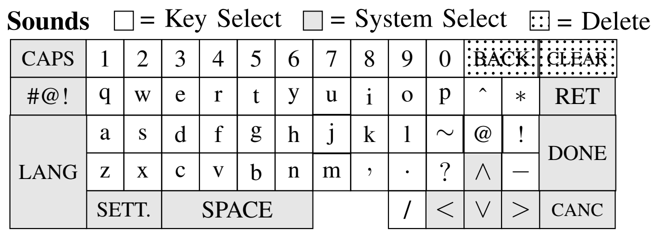
Sounds
= Key Select
= System Select
= Delete
CAPS 1 2 3 4 5 6 7 8 9 0 BACK CLEAR
#@! q w e r t y u i o p ^ * RET
LANG a s d f g h j k l ~ @ !
z x c v b n m , . ? ∧ –
DONE
SETT. SPACE / < ∨ > CANC

Fig. 3. The default keyboard on the Samsung Smart TV. The background denotes the key's sound when pressed.

TVs with wireless remotes that communicate over Bluetooth or infrared [35]. For common platforms [4], [55], these remote controllers contain direction pads that allow users to navigate the TV (Figure 1).

Users enter information into Smart TVs using an on-screen virtual keyboard. These keyboards handle sensitive information, such as passwords and credit cards, as users log into accounts and purchase products (e.g., subscriptions). The keyboard layout depends on the Smart TV operating system (OS) and, if applicable, the current application. We focus on the default keyboards from two Smart TVs: AppleTVs (tvOS) [4] and Samsung Smart TVs (Tizen OS) [55]. AppleTVs use an alphabetical design (Figure 2), and Samsung TVs use a QWERTY layout (Figure 3). On these platforms, users type by moving a cursor to the desired characters with the remote's directional pad. For instance, to reach `j` from `q`, users can press the 'right' button six times and 'down' once. Users may rapidly scroll through system-dependent actions. AppleTV remotes have a directional touchpad, allowing rapid traversal by swiping. On Samsung TVs, users can quickly scroll by holding down a direction button. Finally, Samsung TVs support horizontal wraparound; for example, moving left from `CAPS` places the cursor on `CLEAR`. There is no vertical wraparound. AppleTV supports no wraparound of any kind.

Virtual keyboards have multiple views, each containing a different character set. Users can switch views by selecting a "view-change" key, e.g., `ABC` in Figure 2 or `#@!` in Figure 3. We observe a crucial property of both TVs:

(P1) Upon opening the keyboard, the cursor always starts on the same key in the same view. For AppleTVs, the starting view is the lowercase letters (Figure 2) with the cursor on `a`. On Samsung Smart TVs, the cursor starts on `q` within the lowercase QWERTY keyboard (Figure 3).

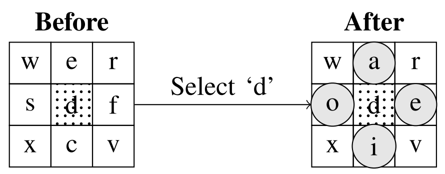

Fig. 4. Example of dynamic key suggestions on the Samsung Smart TV upon selecting the character `d`.

1) Acoustic Properties: A critical property of popular Smart TVs is that their default keyboards make sounds during user interaction. In particular, both AppleTVs and Samsung Smart TVs have audio profiles with the following properties:

(P2) The sound of moving the cursor, called `KeyMovement`, differs from that of selecting a key, called `KeySelect`.

(P3) The `BACK` and `CLEAR` keys make different sounds than that of other keys. We call this sound `Delete`.

(P4) The `KeySelect` sound is unique to the keyboard. When making selections outside the keyboard (e.g., choosing a video to play), the TV makes a different sound, which we call `SystemSelect`. The keyboard can make the `SystemSelect` sound on special keys (Figure 3).

Unlike the acoustic emanations from mechanical keyboards [6], [70], these sounds originate from the TV OS. Thus, the sounds are consistent and user-independent. Further, each platform has its own version of each sound; the `KeySelect` sound on the AppleTV is *not* the same as `KeySelect` on the Samsung Smart TV.

Both TVs have inaudible shortcuts by pressing buttons on the remote. On AppleTVs, users can change the keyboard view. The Samsung TV allows users to capitalize characters. These actions move the cursor deterministically; e.g., if the cursor is on `d`, it moves to `D` after the capitalization shortcut.

2) Dynamic Keyboards: Samsung keyboards sometimes provide inline suggestions by populating neighboring locations with new characters (Figure 4). This design aims to reduce the user's movement on the keyboard. Moving twice in the same direction clears the suggestions. The keyboard always starts with no suggested keys. These suggestions only occur when users enter "predictable" information such as web searches. There are no suggestions when entering passwords. For passwords, the only dynamic behavior occurs when the keyboard suggests `Done` after the user reaches eight characters. AppleTV keyboards have no dynamic behavior.

B. Notation

Our attack uses move count sequences to recover strings typed on a Smart TV virtual keyboard. A *move count sequence* is an ordered list $S = [M^{(n)} = (k^{(n)}, s^{(n)}, \boldsymbol{t^{(n)}})]_{n=1}^{N}$ where $M^{(n)}$ is the the n^{th} *move*. Each move has three items. The first, $k^{(n)}$, is the number of cursor movements to navigate from the $(n-1)^{th}$ key to the n^{th} key. Second, $s^{(n)} \in Q = \{\texttt{KeySelect}, \texttt{SystemSelect}, \texttt{Delete}\}$ is the sound made upon selecting the n^{th} key where Q denotes the set of *end* sounds. Finally, $\boldsymbol{t^{(n)}} \in \mathbb{R}^{k^{(n)}}$ holds the times of the individual movements.[2] We infer user behavior from $\boldsymbol{t^{(n)}}$ (§IV-C). We denote strings as w where w_ℓ is the ℓ^{th} character and $w_{\ell:r}$ is the substring from position ℓ to r inclusive. We define a *keyboard instance* as a contiguous episode starting with opening the keyboard and ending with the string's submission (e.g., by clicking `Done`). A move count sequence S can have zero or more keyboard instances.

As an example, consider typing the string `test` on the Samsung Smart TV keyboard (Figure 3). With timing information omitted, one possible move count sequence S for this string is below. The last move comes from navigating to `Done`.

$$S = [(k^{(0)} = 4, s^{(0)} = \texttt{KeySelect}), (k^{(1)} = 2, s^{(1)} = \texttt{KeySelect}), (k^{(2)} = 2, s^{(2)} = \texttt{KeySelect}), (k^{(3)} = 4, s^{(3)} = \texttt{KeySelect}), (k^{(4)} = 9, s^{(4)} = \texttt{SystemSelect})]$$

There are an infinite number move count sequences for a single string because users can traverse suboptimal paths between keys. Similarly, one move count sequence can correspond to many strings.

We use S as an intermediate representation for the TV's audio. Our attack works by converting audio into a move count sequence S and then finding strings that match S in the keyboard layout.

C. Credit Card Details

Users type credit card details into Smart TVs when purchasing products (e.g., movies or subscriptions). For example, users must enter their payment information into the Hulu application when creating a new account on Samsung Smart TVs. These purchases are "card-not-present" transactions. Users must provide sufficient payment details to validate the transaction with a card payment network. These details often include the credit card number (CCN), expiration date, security code (CVV), and billing address. The CVV is a 3- or 4-digit sequence of pseudo-random numbers. Popular Smart TV applications, such as Hulu, validate the billing address through a 5-digit ZIP code. Therefore, the 5-tuple of (CCN, expire month, expire year, CVV, ZIP) is sufficient to purchase products, making this information financially valuable to an attacker. We call this 5-tuple the *full credit card details*. We focus on payments in the United States, though the formats of other countries are conceptually similar.

CCNs are generally 15- or 16-digit strings where the first digit identifies the card issuer. We focus on three popular issuers: American Express (AMEX), Visa, and Mastercard. Apart from the last digit, the remaining numbers identify the issuing institution and user account. The final digit ensures the entire CCN satisfies a checksum under Luhn's algorithm. This checksum is essential to our attack's performance when recovering CCNs (§V-B, §VI-B).

To verify stolen credit card information, an attacker must execute a transaction against a card payment service. If the service accepts the transaction, the attacker has valid payment details. Upon rejection, we assume the attacker only knows that *at least* one field is incorrect. Further, we assume all five fields must be correct for a valid transaction. This assumption is conservative, as some services may only validate a subset of fields [56].

[2]We denote vectors in boldface and scalars in plain text.

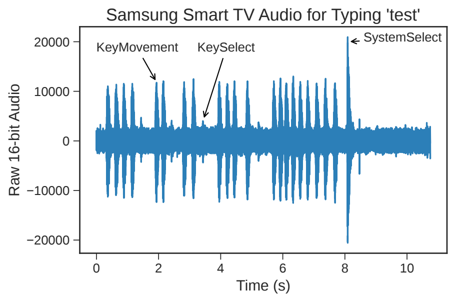

Fig. 5. Example audio from the Samsung TV when typing the string `test`.

D. Example of Acoustic Keystroke Leakage

We provide an example of acoustic keystroke leakage by typing the string `test` on the Samsung TV (Figure 3) and recording the audio. Figure 5 shows the audio produced by the TV (§VI-A describes the experimental setup). With prior knowledge of the TV's sounds, we identify the user's keyboard actions from the raw audio (P2). This identification yields the move count sequence S from §II-B. Visually, this example highlights the distinctive nature of Smart TV sounds.

To recover the typed string, an attacker can combine the keyboard layout (Figure 3) with S. Assuming the user takes shortest paths, the first character is $w_0 \in \{\texttt{t}, \texttt{4}, \texttt{f}, \texttt{c}, \texttt{\^{}}, \texttt{!}\}$ because the cursor starts on `q` (P1). Then, the attacker considers keys at a distance of $k^{(1)} = 2$ from each possible w_0. Iteratively continuing this process yields possible strings such as `ft6f`, `test`, `tukt`, and so forth. The attack can identify the most likely result using a prior dictionary over all possible strings. For instance, the attacker can correctly recover `test` under an English prior. This general method allows an attacker to recover keystrokes from a Smart TV's audio.

III. Threat Model

We consider an attacker with passive audio access to a Smart TV running Samsung's Tizen OS [55] or AppleTV's tvOS [4]. We assume there is only one TV in the target location. An attacker can gain this access to the TV in one of two ways:

1) The adversary can hijack an adjacent device containing a microphone. Such devices are prevalent due to the growth of smart speakers (e.g., Amazon Echo) [22]. It is reasonable to assume that these devices have vulnerabilities exposing their microphone feed; prior attacks have both turned such speakers into wiretaps [9], [29] and compromised other smart devices [2], [3], [16], [20], [54]. Depending on the vulnerability, an adversary can exploit these nearby devices remotely, launching our attack even without physical access to the target TV.

These assumptions correspond to the threat model of previous keystroke attacks on Smart TVs [21].

2) The attacker can place a malicious microphone near the target TV. This method requires stronger capabilities, as this adversary generally needs physical access to place the microphone. We consider this threat despite this stronger assumption because it remains reasonable for venues with transient populations, such as hotel rooms, vacation homes, and other short-term rental facilities. This vector may be possible without physical access if the TV is audible in an adjacent space (e.g., through the wall of an apartment) or if the attacker leverages long-range (e.g., laser) microphones [43], [54].

We emphasize what the adversary does *not* assume. The attacker makes no assumptions about the type of information (e.g., passwords, credit card details, etc.) or even that the user is typing at all. The attacker must determine when the user is typing and the class of entered information. When the user is typing, the attacker does not assume the user takes an optimal path between keys. Further, the adversary has no other knowledge about the user. This assumption is conservative; for example, knowing personal information improves password guessing [62]. Finally, the attacker must validate inferred keystrokes against online services (e.g., by directly entering a password guess into an online site). The adversary cannot access leaked information on which to validate results offline. This setting mirrors online password guessing and is more challenging due to rate limits and service lockouts [62].

We compare this threat to four related attacks: video-based attacks, attacks on voice inputs, attacks on infrared remotes, and attacks on virtual remotes.

a) Video: Similar to shoulder surfing [65], an adversary with video access to the Smart TV's screen can read keystrokes. Gaining reliable video access, however, is harder than capturing only audio because the attacker needs an unobstructed view of the TV. Therefore, video access is sensitive to the malicious device's location. The audio threat is stealthier because it can operate anywhere near the TV within audible range, allowing the recording device to occupy less conspicuous places. Finally, the audio attack is not sensitive to small changes in the location of the TV or microphone.

b) Voice Inputs: Some Smart TVs support voice inputs. This method replaces virtual keyboards by allowing users to speak their desired string into the remote controller. Voice inputs leak information through acoustic signals, as attackers with audio access to the target TV can use speech recognition [18] to discover the entered string. Smart TVs, however, do not support voice inputs for all information types. For example, Samsung Smart TVs block voice inputs for password fields; instead, the TV forces users to enter passwords with the virtual keyboard. Thus, an attacker who only targets voice inputs cannot steal all forms of highly sensitive information. Our attack targets the virtual keyboard's audio and can discover private information such as passwords.

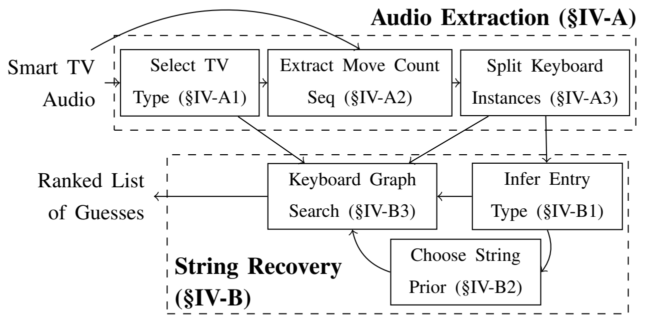
Audio Extraction (§IV-A)
Smart TV Audio
Select TV Type (§IV-A1)
Extract Move Count Seq (§IV-A2)
Split Keyboard Instances (§IV-A3)
Ranked List of Guesses
Keyboard Graph Search (§IV-B3)
Infer Entry Type (§IV-B1)
Choose String Prior (§IV-B2)
String Recovery (§IV-B)

Fig. 6. The keystroke acoustic attack framework.

c) Infrared Remotes: HomeSpy steals user keystrokes by sniffing infrared (IR) signals between Smart TVs and remote controllers [21]. However, this attack only works for IR remotes. Newer remotes communicate over Bluetooth [35], [69], and HomeSpy does not succeed on Bluetooth remotes. In contrast, the audio attack is agnostic to the remote control communication protocol.

d) Virtual Remotes: SPOOK exploits vulnerabilities when pairing virtual remotes to steal Smart TV keystrokes [35]. Enforcing human attestation during the pairing process stops this attack. This patch, however, does not thwart the considered audio threat.

IV. Acoustic Attack Framework

This section describes our acoustic attack against Smart TV keyboards. The attack consists of two core modules (Figure 6). The first component (§IV-A) uses the Smart TV's audio to identify the platform (§IV-A1) and extract move count sequences (§IV-A2). The module then isolates instances of a user typing (§IV-A3). The second module uses each isolated move count sequence to recover the typed string (§IV-B). This module first infers the type of entered information (e.g., passwords, credit card details, etc.) (§IV-B1). From this inference, the attack customizes a prior dictionary over possible strings (§IV-B2). We combine this prior with the keyboard layout to guess the likely typed strings (§IV-B3). This section concludes by discussing improvements based on timing patterns (§IV-C).

The attack acts on fixed-length audio recordings. This property is necessary to identify the platform (§IV-A1) and information (§IV-B1) types. The attacker can obtain such recordings by splitting the microphone feed during prolonged silence (e.g., at night). The attack automatically identifies and isolates the instances of users typing within the larger recording. This property implies the adversary cannot launch the attack in real-time. However, this limitation is manageable as it enables the attacker to make minimal assumptions. Further, the value of sensitive information, such as passwords and credit cards, does not significantly diminish over these delays.

A. Audio Extraction

1) TV Type Selection: The first step in the attack pipeline uses acoustic properties to identify the Smart TV platform.

Different Smart TVs use unique audio profiles, so the TV's sounds fingerprint the system. On the AppleTV and Samsung Smart TV, the `KeySelect` sounds uniquely identify keyboard usage (P4). Thus, for a given recording, we match instances of the `KeySelect` sound for each platform (§IV-A2 describes this matching). We classify the TV type as the platform with the most matching `KeySelect` sounds over the recording's duration. When no matches exist, the platform type is unknown, and we abort the attack. Such cases indicate background noise, an unsupported TV, or no keyboard use.

2) Extracting Move Count Sequences: Smart TVs make distinctive sounds (P2), (P3), leading to keyboard interactions with audio such as that of Figure 5. This section describes a procedure to separate and classify the TV's relevant sounds (Q1). This process extracts a move count sequence, S, from raw audio (Q2).

From Figure 5, the Smart TV's sounds appear as regions with high amplitude. Thus, the module finds each amplitude peak and identifies the surrounding region as a candidate sound. This process follows prior acoustic attacks against mechanical keyboards [10].

The module must then classify each candidate sound as one of `KeySelect`, `SystemSelect`, `KeyMovement`, `Delete`, or `Unknown`. We use a nearest-neighbor classifier due to the consistency of Smart TV sounds over time. We pre-collect recordings for each sound on the target Smart TV platforms. We call these recordings the reference sounds. At runtime, the framework creates the spectrogram of the candidate and reference sounds using a Fourier transform length of 1,024, a segment length of 256, and 32 points overlapping between segments. We then compute the L1 distance between the candidate and reference spectrograms. We select the proper reference sounds using the inferred TV type (§IV-A1). The module classifies the candidate based on the lowest distance. If the best distance exceeds a threshold, the sound is `Unknown` and therefore unrelated to the TV. We empirically determine these thresholds for each reference sound to filter out false positives on a set of training examples.

Three challenges arise with this nearest-neighbor approach.

(a) *Dimension Mismatch:* The candidate and reference spectrograms may have a different number of timesteps. This dimensionality mismatch prevents computing the L1 distance directly. We solve this issue by instead sliding the smaller spectrogram over the larger one. We return the minimum sliding window distance.

(b) *Background Noise:* The L1 distance is sensitive to background noise. We mitigate this problem by limiting the comparison to the frequency bands which contain each reference sound. Further, we min-max normalize each spectrogram and mask out normalized values below a threshold τ. We empirically set $\tau = 0.7$ based on our training examples. This masking avoids comparing low-amplitude noise in the target frequency bands.

(c) *Rapid Scrolling:* Users can quickly scroll across keys (§II-A). This action generally creates unique sounds for each movement. These movements, however, occur in rapid succession and get clipped as a single candidate. We deduplicate these movements by finding the spectrogram peaks [61] and counting the number of peaks at frequencies tuned for the `KeyMovement` sound. We use the number of peaks to identify the number of individual movements.

After classifying each sound, the framework builds the move count sequence S. The n^{th} move $M^{(n)}$ completes with $s^{(n)}$, the n^{th} instance of an end sound (§II-B). Upon observing $s^{(n)}$, we count the number of cursor movements between the $s^{(n-1)}$ and $s^{(n)}$; this count is $k^{(n)}$. Finally, we set the times $\boldsymbol{t^{(n)}}$ using the amplitude peak times for each detected cursor movement in $M^{(n)}$. These extracted moves $M^{(n)}$ form the move count sequence for the full recording.

This method produces accurate results. For users typing credit card details (§VI-A), the module identifies the correct number of movements on 98.95% of instances and misses only a single selection. This algorithmic approach shows an adversary can automate the extraction and launch the attack at scale.

3) Splitting Keyboard Instances: The move count sequence S (§IV-A2) encodes the user's behavior over an entire recording. Each recording can have zero or more keyboard uses. The attack must isolate each keyboard instance (§II-B) to recover the individual user-typed strings. This phase produces disjoint sequences $S_1, \ldots, S_\ell$ such that $\bigcup_{r=1}^{\ell} S_r \subseteq S$ where each S_r holds a single keyboard instance. We perform this splitting using both acoustic and timing information. The details depend on the platform, as discussed below.

a) Apple: The AppleTV keyboard has no dynamic behavior (§II-A2) and requires users to select `Done` upon completion (Figure 2). `Done` is the only key to make the `SystemSelect` sound. We can thus split S using moves with an ending sound of `SystemSelect`. We only retain splits S_r with at least one move ending in a `KeySelect`; otherwise, the user did not interact with the keyboard (P4).

b) Samsung: This platform is more complicated for two reasons. First, `Done` is not the *only* key to make the `SystemSelect` sound (Figure 3). Second, the keyboard can dynamically suggest `Done` (§II-A2), and this suggestion makes the `KeySelect` sound. Therefore, splitting the move count sequence using `SystemSelect` sounds is incorrect. Instead, we use timing information based on the insight that applications incur noticeable latency upon submitting a string (e.g., executing a search query). We identify these delays as outliers in the times between adjacent moves. In particular, the cutoff of $\texttt{avg}(\texttt{move\_diffs}) + 1.5 \cdot \texttt{stddev}(\texttt{move\_diffs})$ creates the proper splits in almost all cases (§VI-C) where `move_diffs` is the array below for all $n \in \{1, 2, \ldots, len(S) - 1\}$.

$$\texttt{move\_diffs}[n] = t_1^{(n+1)} - t_{k^{(n)}}^{(n)} \qquad (1)$$

This method can make mistakes. For example, the attacker will incorrectly split keyboard instances if a user takes a long pause. However, such failure cases rarely occur in practice (§VI-C). We omit splits without any moves ending in `KeySelect`.

Credit cards are an exception to this time-based approach. Credit card information is entirely numeric, so `Done` is the only required key that makes the `SystemSelect` sound (Figure 3). Further, when selecting numbers, the keyboard never dynamically suggests `Done`. Thus, we can split credit card sequences using the `SystemSelect` sound. The challenge, however, is the attack has no prior knowledge that a user is entering credit card details. We address this problem by combining both methods. We first split S using `SystemSelect` sounds and identify any credit card entries (§IV-B1 explains this identification). We then remove the credit card sequences and re-split the remainder using timing. This combined approach addresses the dynamic behavior of Samsung Smart TV virtual keyboards.

B. String Recovery

1) Inferring Entry Types: Users enter various classes of information into Smart TVs; we focus on three types: passwords, credit cards, and English words. The first two classes are highly sensitive, and the last class encapsulates inputs such as web searches. These information types have unique details, and the attack should customize the string recovery to each class to optimize performance (§IV-B2). The attack infers the information type to make these customizations without introducing additional assumptions.

We detect credit card information on the Samsung TV by counting the string lengths (P3) of consecutive keyboard instances. Forms accepting credit card details collect five pieces of information with distinctive lengths (§II-C). We use these lengths to fingerprint credit card details and look for consecutive instances matching the sizes of each field. We assume the CCN comes first, and the year follows the month; otherwise, this matching is order-insensitive. For a given match, we identify each field by its length and position.

If a keyboard instance is not in a credit card field, we must distinguish whether it is a password or an English word. To make this distinction, we leverage how the Samsung TV employs dynamic suggestions when expecting English words (§II-A). These suggestions are *not* present for passwords, allowing the attack to identify the information type from this discrepancy. We use a Random Forest classifier [12] to determine the presence of dynamic keyboard behavior for each instance S_r. The Random Forest provided better results than other considered models, such as gradient-boosted decision trees. The input features are a histogram of movement counts in S_r, which are valuable because suggestions reduce the average distance between selections. We train this model on generated move count sequences for passwords and English words on the Samsung keyboard (Figure 3). We mimic suggestions by manually recording the TV's suggestions after the first selection; after, we use the likeliest characters from an English dictionary [57]. We bias this classifier toward passwords due to their greater value for an attacker. We classify S_r as an English word if the model's prediction probability exceeds 0.6. We set this cutoff empirically to reach over 99% recall on passwords in our validation set.

We limit our analysis of AppleTVs to passwords. AppleTVs have no browser, so the only English inputs are video titles or application-specific searches. These searches are far less valuable than passwords. Further, there is no location to enter credit card details, as all payments occur through the user's Apple account. Thus, on AppleTV, the framework assumes all keyboard inputs are passwords.

2) Choosing the Prior String Dictionary: Smart TV audio does not provide the direction of keyboard movements. Thus, a move count sequence corresponds to many possible strings, and the attack must distinguish these possibilities. We make these determinations using a prior dictionary over possible strings. The prior depends on the information type. For example, credit card numbers (CCNs) are numeric, while passwords have larger character sets. The attack customizes the prior using the inferred information type (§IV-B1).

We design priors for each string type. The CCN prior enforces the prefixes of AMEX, Visa, and Mastercard (§II-C); the remaining digits appear uniformly at random. The security code (CVV) prior consists of digits occurring at random. For efficiency reasons, we do not materialize the full CCN and CVV priors. The expiration month and year priors enforce valid dates up to 2035. We construct the ZIP prior using the 33,120 United States ZIP codes from 2019 and weigh ZIPs by population. We follow previous work by building an N-gram model [33] for passwords from leaked datasets [41]. The English prior uses word prefixes from the Wikipedia corpus [57]. We weigh prefixes by frequency and only keep words with at least 100 appearances. This prior has 95,892 words.

These string priors represent options that work well in practice (§VI). We emphasize that an adversary can change these priors without fundamentally altering the attack framework.

3) Keyboard Graph Search: The attack recovers strings from move count sequences S_r by performing a variant of Dijkstra's algorithm on the Smart TV's keyboard graph (Q3). This keyboard layout is known from the inferred TV type (§IV-A1). Within Dijkstra's algorithm, we use search states of the form below.

$$\mathtt{z} = (\mathtt{key}, \mathtt{str}, \mathtt{move\_num}, \mathtt{keyboard\_view}) \tag{2}$$

When expanding the state z, we use the move $M^{(n)}$ (§II-B) from S_r where $n = \mathtt{z.move\_num}$. We then find the set of neighbors V by retrieving the keys in the current keyboard view at a distance $k^{(n)}$ from `z.key` which make the sound $s^{(n)}$. We consider neighbors at the distance $k^{(n)}$ both with and without wraparound, as users often ignore this feature. We create the candidate string for each neighbor $v \in V$ by appending v to `z.str` and accounting for the key's action. This process is often concatenation, though special keys (e.g., `BACK`) have different behavior. We compute the edge weight for this neighbor using the string prior (§IV-B2).

$$\mathtt{weight} = -\log\left(\frac{\mathtt{Count(candidate\_str, Prior)}}{\mathtt{Count(z.str, Prior)}}\right) \tag{3}$$

We give special keys (e.g., `CAPS`) a weight of zero and ignore neighboring keys with zero count. We use Dijkstra's algorithm with these weights to find the maximal path. With this method, the score for each string w is logically equivalent to the following probability under the prior.

$$p(w) = p(w_1) \cdot \prod_{\ell=2}^{len(w)} p(w_\ell | w_{1:(\ell-1)}) \tag{4}$$

We construct guesses upon exhausting the available moves in S_r, i.e., when `z.move_num` $= len(S_r)$. Before producing a guess, we validate the string based on the information type. This validation is crucial for credit cards, as each CCN must pass a checksum (§II-C). We continue searching until producing a ranked list of L guesses. The search always starts from the keyboard's fixed start key (P1).

This graph search forms the backbone of the string recovery process. However, to achieve a useful attack, the procedure must also overcome the following four challenges: (1) suboptimal paths, (2) errors in audio extraction, (3) multiple keyboard views, and (4) dynamic suggestions. We discuss each of these challenges below.

a) Suboptimal Paths: As presented, the search assumes that users take an optimal (i.e., shortest) path between keys. This assumption manifests by finding neighboring keys at a distance $k^{(n)}$. Users, however, may not take optimal paths (Figure 7), and considering only optimal paths can cause the recovery to miss the true string (Q4). We handle suboptimal movements by expanding the search at move $M^{(n)}$ to find keys within a distance range of $I = [k^{(n)} - d^{(n)}, k^{(n)} + d^{(n)}]$ (§IV-C1 describes how to set $d^{(n)}$). We discount suboptimal paths using the factor $\gamma^{|d-k^{(n)}|}$ where $d \in I$ is the considered number of movements. We multiply this discount factor with the string probability before applying the logarithm in Equation 3. This procedure allows the search to consider suboptimal movements without penalizing optimal paths.

b) Audio Extraction Errors: The audio extraction can make mistakes, and these errors generally occur when conflating adjacent movements (§IV-A2). We handle these errors as suboptimal paths by setting the tolerance $d^{(n)}$ to be at least the number of rapid scrolls (§IV-C2). With this tolerance, the search can still consider the correct neighbor even when miscounting the number of movements.

c) Keyboard Views: Virtual keyboards have multiple views, and users can inaudibly change views with the remote controller (§II-A). Such changes are not present in S_r. We handle these view changes using an exhaustive search. When adding a state to the search queue, we also add all states reachable through an inaudible view change. For example, the Samsung TV allows users to capitalize keys with the remote. When adding the state (`d`, '`str`'$||$`d`, `move_num`, `lower`), we also push (`D`, '`str`'$||$`D`, `move_num`, `upper`) onto the search queue where $||$ is concatenation. This method works because the cursor moves deterministically across view changes (§II-A).

d) Dynamic Suggestions: The Samsung TV contains a final complication: dynamic suggestions (§II-A2). The suggestions are unknown beforehand, making them a quantity the attacker must predict. We perform this prediction using the most common characters in the English dictionary that follow the current state's string. The keyboard suggests up to four characters (Figure 4). Since our predictor does not exactly match the Smart TV, we use the top six guessed characters. We add these keys to the neighbor set V when the number of movements is at most four, as the suggestions are adjacent to the cursor (Figure 4). Finally, with dynamic suggestions, we always use a tolerance $d^{(n)} \geq 1$ because users make an additional movement to clear the suggested keys (§II-A2).

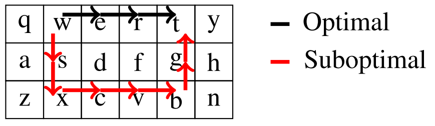

Fig. 7. Optimal and suboptimal paths between `w` and `t`.

C. User Timing Patterns

Through a user study (§VI), we observe that people exhibit consistent timing patterns when typing on Smart TVs. This section highlights two beneficial aspects of keystroke timing. The attack first leverages timing to detect suboptimal paths (§IV-C1). Second, keystroke timing can provide information on movement directions (§IV-C2).

1) Suboptimal Paths: Virtual keyboards do not enforce that users take an optimal path between keys (Figure 7). As described, the string recovery module supports suboptimal paths by expanding the set of considered neighbors using the tolerance terms $d^{(n)}$ (§IV-B3). The attack must determine how to set each $d^{(n)}$.

One possible method is to set $d^{(n)}$ to a fixed $D > 0$ for every move $M^{(n)}$. However, this approach leads to low recovery performance when the information type has no strong prior. For example, CCNs and CVVs place near-uniform priors over all digits (§IV-B2). Under this strategy, the procedure scores strings equally if they use the same number of suboptimal paths. This equality occurs independent of *when* the suboptimal moves happen due to the uniform prior. Therefore, the order of guesses depends on the arbitrary tie-breaking scheme between equal priorities on the search queue.

We create a better method based on typing behavior. Through a user study (§VI-A), we find that people tend to *pause* when correcting a suboptimal path. This indicator, however, is noisy, and we address this noise using an iterative solution. For each $M^{(n)}$, we compute $u^{(n)}$, the largest time difference between adjacent movements for all $n \in \{1, 2, \ldots, len(S)\}$

$$u^{(n)} = \max_{q \in \{2, \ldots, k^{(n)}\}} (t_q^{(n)} - t_{q-1}^{(n)}) \tag{5}$$

We sort the moves in decreasing order of their maximal delays $u^{(n)}$. We use this ordering to consider increasingly more moves with suboptimal paths. This process works as follows. We begin the search (§IV-B3) with $d^{(n)} = 0 \ \forall n$. If this search ends with fewer than L guesses, we expand it by selecting the move index $n' = \arg\max_n u^{(n)}$. We set $d^{(n')} = D$ and the remaining $d^{(n)}$ to 0. We re-execute the search with these parameters. If this process again produces fewer than L results, we expand the candidate suboptimal paths by selecting the top two highest-delay moves. We continue this expansion until reaching L results. An individual search may produce less than L results because (1) strings can have a zero count in the prior, and (2) results may not pass the information type's validity check (e.g., CCN checksums). Note that we continue to account for rapid scrolls and dynamic suggestions (§IV-B3) on the remaining $d^{(n)}$ at each step. Overall, this timing-based design better identifies where suboptimal paths occur (§VI-E).

2) Direction Inference: Users can rapidly scroll across keys using the TV's remote (§II-A). Each scroll constitutes movements in a single direction; to change direction, users pause to switch buttons on the remote. Further, neither AppleTV nor Samsung keyboards support vertical wraparound. Therefore, if the user rapidly scrolls across at least four keys, the user moves horizontally with high probability. This cutoff follows from the number of keyboard rows (Figures 2, 3). Inferring horizontal directions allows the attack to reduce the search space. For example, if we detect four horizontal movements from `q` on the keyboard in Figure 3, then the neighbor set is $V_{dir} = \{\texttt{t}, \texttt{\^{}}\}$. Without directions, the neighbor set is $V = \{\texttt{t}, \texttt{4}, \texttt{f}, \texttt{c}, \texttt{\^{}}, \texttt{!}\}$.

The attack infers these directions by identifying rapid scrolls through movement timing. Consider the n^{th} move with times $\boldsymbol{t^{(n)}}$ (§II-B). We compute the time differences between adjacent movements, $\boldsymbol{\alpha^{(n)}}$, and create the cutoff $c^{(n)} = \texttt{median}(\boldsymbol{\alpha^{(n)}})$. We assign directions by observing windows of four adjacent movements. If all time differences in a window are at most $c^{(n)}$, these movements are considered part of a rapid scroll and given a direction of "horizontal." The remaining movements are in "any" direction. We ignore the inferred directions for suboptimal paths because we do not know *where* the suboptimal movement occurs. We find that direction inference never harms the recovery (§VI-E).

V. Attack Results in Emulation

We first assess the efficacy of the string recovery module (§IV-B) under ideal conditions. This recovery is nontrivial because a single move count sequence can refer to many strings (§II-B, §IV-B3). This section first describes the emulation setup (§V-A) before presenting the results for recovering credit cards (§V-B) and passwords (§V-C).

A. Setup

The emulation environment evaluates string recovery (§IV-B). We algorithmically generate move count sequences using the Smart TV's keyboard graph. This sequence uses optimal paths apart from randomly choosing when to use available wraparound. We focus on two types of private information: credit cards and passwords.

1) Credit Card Recovery: We use credit card details with five fields: credit card number (CCN), expiration date (mm/yy), security code (CVV), and ZIP code (§II-C). When entering these details into existing applications, the Samsung Smart TV provides users with the full QWERTY keyboard (Figure 3). We randomly generate semantically valid fake Visa, Mastercard, and AMEX CCNs. We select random expiration months and choose arbitrary years between 2023 and 2033. We generate CVVs as three (Visa and Mastercard) or four (AMEX) digit random numbers. Finally, we randomly select valid ZIP codes weighted by population. We create 6,000 entries, with 2,000 for each provider.

The search outputs a list of guesses for each field. The adversary, however, can only validate the full details by getting every field correct (§II-C). We compute the rank for the full details by first exhaustively searching over the top 10 CCNs, three CVVs, three ZIPs, two months, and two years in this order. If we find no complete match, we iteratively expand the search using the following cutoffs: `(CCN: 30, CVV: 5, ZIP: 5, Month: 2, Year: 2)`, `(CCN: 100, CVV: 12, ZIP: 12, Month: 3, Year: 3)`. We select this order of fields based on our uncertainty in the corresponding information. For example, CCNs are the most uncertain as they are long, nearly-random numeric strings.

We compute the attack's results up to 250 CCN guesses and 5,000 guesses for the full details. These cutoffs yield a top-K accuracy which is the rate at which the correct result is in the first K guesses. In this notation, K denotes the guess cutoff i.e., the number of guesses allowed by the adversary. We display $K \in \{1, 5, 10, 50, 100, 250\}$ for CCNs and $K \in \{1, 10, 100, 1000, 2500, 5000\}$ for the full details. In practice, attackers validate the results against rate-limited online services (§III). From prior work on password guessing, an attacker can make at least 100 guesses against a single service with a sufficient delay between trials [32]. Unlike passwords, payment details are global, and an attacker can circumvent the limits of an individual site by simultaneously testing against different services. Thus, we focus on the top-1000 accuracy for the full details.

2) Password Recovery: We evaluate the recovery of passwords from the 2014 PhpBB password leak [41]. To make this set realistic, we follow NIST guidelines [17] and only include passwords with at least eight characters. We purposefully select diverse strings by enforcing that 25% of the passwords have at least one special character, number, uppercase letter, or lowercase letter. We match the credit card benchmark by selecting 6,000 passwords. We experiment with two different 5-gram language models [33] using: (1) the same PhpBB set and (2) the RockYou password leak [41].

We measure the performance for the guess cutoffs $K \in \{1, 5, 10, 50, 100, 250\}$ and focus on the top-100 accuracy. This latter cutoff makes our attack applicable to online settings [62]; from prior work, attackers can make at least 100 password guesses against a majority of popular websites [32].

B. Credit Card Recovery

We attack credit card details on the Samsung TV. Figure 8 shows the Top-K accuracy on the CCN and the full details. The CCN alone can be valuable for attackers, especially when

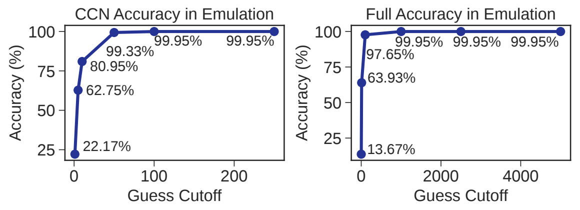

Fig. 8. Accuracy on CCNs (left) and full credit card details (right) in emulation on the Samsung Smart TV.

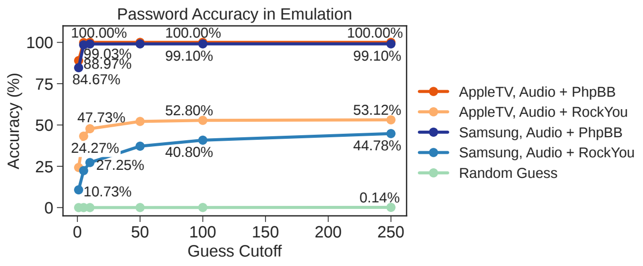

Fig. 9. Password recovery in emulation. The data labels show the Top-K accuracy for $K \in \{1, 10, 100, 250\}$.

vendors do not validate all payment information [56]. The attack discovers both CCNs and full payment details. For CCNs, the attacker reaches an 80.95% top-10 and 99.95% top-100 accuracy. On the full details, the attacker finds 63.93% (top-10) and 99.95% (top-1000) of the full payment information. This top-1000 accuracy means the attack can use optimal move count sequences to find and validate 99.95% of payment details against rate-limited services (§V-A1).

We highlight two aspects of these results. First, the attack succeeds on all three providers. On the full details, the top-10 accuracy is 61.55% (Visa), 57.00% (Mastercard), and 73.25% (AMEX). The AMEX results are better because the CCN has only 15 digits. The top-1000 accuracy is above 99.90% for all providers. Second, on average, only 16.26% of the potential CCN guesses satisfy the checksum (§II-C). This result highlights the importance of the checksum; without this validation, the CCN ranks would be roughly 6× larger.

C. Password Recovery

We further recover passwords on both platforms. Figure 9 shows the results. The baseline accuracy is the expected result of randomly guessing from the set of 184,388 PhpBB passwords. We discuss four takeaways.

First, with the PhpBB prior, the attacker reaches at least 84.67% top-1 and 99.03% top-10 accuracy on both TVs. These results vastly exceed random guessing, confirming the value of optimal move count sequences. With these results, the adversary can reliably verify the discovered passwords in rate-limited contexts [32].

Second, the string prior plays a significant role in the attack's efficacy. Under the RockYou prior, the top-10 accuracy drops to 47.73% (AppleTV) and 27.25% (Samsung). These results occur due to less string overlap between the PhpBB and RockYou lists. Nevertheless, knowing the move count sequence allows the attack to consistently outperform random guessing by over 330× even when the target password may not be in the string prior.

Third, the attack performs better on the AppleTV for two reasons. First, the AppleTV keyboard (Figure 2) limits the number of movement directions compared to the Samsung keyboard (Figure 3). Thus, the attack has fewer neighbors to consider for each move. Second, on Samsung instances, we must infer whether the keyboard uses dynamic suggestions (§IV-B1). This inference has a 99.23% recall on passwords. The misclassified cases lead to incorrect guesses, as the attack wrongly uses the English prior on a keyboard with suggestions. Nevertheless, the attack still displays high accuracy on Samsung systems. Appendix A-A contains further experiments measuring the keyboard layout's impact on string recovery.

Finally, the attack finds passwords with diverse character sets. On the AppleTV under the PhpBB prior, the top-1 accuracy is 92.33% (special), 92.80% (numeric), and 76.64% (uppercase). For the Samsung Smart TV, these figures are 88.60% (special), 89.93% (numbers), and 73.68% (uppercase). The uppercase accuracy is lower due to inaudible case changes (§II-A); the lowercase version of the password has the same move count sequence and often a higher score.

In total, the attack reliably discovers user keystrokes when given optimal move count sequences.

VI. Attack Results on Users

We evaluate the full attack on human users typing credit card details (§VI-B), passwords (§VI-C), and web searches (§VI-D) into Smart TVs. Finally, we quantify the benefits of keystroke timing (§VI-E).

A. Setup

We conduct a user study with ten subjects (Appendix A-B). Our university's institutional review board (IRB) approved this study, and subjects were compensated $25 for an hour. Each subject types three sets of credit card details, ten passwords, and ten web searches. Users type credit card details into the Hulu application on the Samsung TV as a part of the account creation flow. Subjects enter passwords on the Samsung TV as if connecting to a WiFi network. Users type passwords into the AppleTV as if logging into their Apple account. Subjects type web searches using the Samsung TV's browser. To not bias behavior, we do not inform the subjects of the attack beforehand; we only state that we were studying how users type on Smart TVs. The attack's efficacy strongly suggests that ten users are sufficient to show this vulnerability (§VI-C).

We construct lists of passwords and credit cards using the same method as in emulation (§V-A). Half of the passwords contain at least one special character. We select web searches by sampling words with at least five characters from news headlines [28]. We assign the same lists to subjects A/F, B/G, C/H, D/I, and E/J, enabling a comparison of how different users type the same strings. We use the same passwords on both platforms. Users did *not* enter their own personal

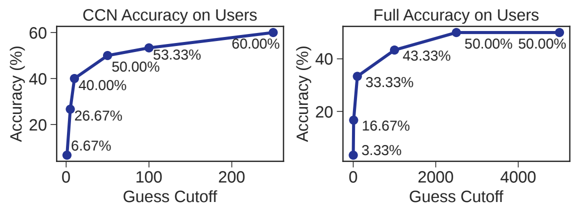

Fig. 10. Accuracy for credit card numbers (left) and full credit card details (right) on the Samsung Smart TV.

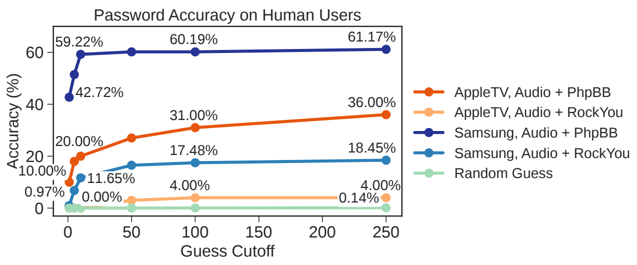

Fig. 11. Password recovery accuracy for human users. The data labels show the Top-K accuracy for $K \in \{1, 10, 100, 250\}$.

information; we provided the strings to type. Although having users type their own details would constitute a more realistic experiment, doing so would be a clear ethical violation. For each information type, users iteratively interacted with the relevant workflow, entering new strings each time. We recorded each experiment in its entirety. The attacker has no prior knowledge of the number of entered strings and must identify the individual keyboard instances. We measure the attack's performance using the same guess cutoffs as in §V-A.

We use an A1625 AppleTV with tvOS version 16.3.2 [4] and a model UN55MU6300 Samsung Smart TV running Tizen software version T-KTMAKUC-1310.1 [55]. These TVs have different remote controllers (Figure 1). The recording device was a commodity Fifine K699B microphone placed about 5.5 feet from the TV. We set the TV volume to 100. Users did not speak during the experiment. There was some background noise from servers running in the same room, but it did not interfere with the users' ability to hear the television audio.

We use a discount factor of $\gamma = 0.5$ on the AppleTV and $\gamma = 10^{-2}$ on the Samsung Smart TV. This discrepancy exists because users take more suboptimal paths on the AppleTV (§VI-C). We further account for this difference in suboptimal paths by setting the suboptimal tolerance to $D = 6$ on the AppleTV and $D = 4$ on the Samsung platform. Beyond these tolerances, the discount factors become so small that they effectively eliminate exploration.

B. Credit Card Recovery

The subjects enter 30 total credit card details on the Samsung Smart TV into the signup flow for Hulu, a popular video streaming application. This application uses the system's provided keyboard (Figure 3). Each user completes the signup flow three times, and the attacker identifies the credit card details within this larger interaction.

The attack successfully identifies that the user enters credit card details in 29 of the 30 total interactions. The one failure occurs because the subject returns and edits a previous field, breaking the length-based matching (§IV-B1). After identifying the presence of credit cards, the attacker extracts the entered values (§IV-B3). Figure 10 shows the accuracy on CCNs and full payment details. The attack reaches a top-100 accuracy of 53.33% (16 / 30) on CCNs and a top-1000 accuracy of 43.33% (13 / 30) on the full details. These values mean the attacker can confirm payment details against rate-limited services 43.33% of the time (§V-A). This leakage is significant due to the value of credit card details; the discovered information is sufficient to make online payments on behalf of the target.

These results highlight how the attack works across users and providers. The attack successfully finds at least one CCN in the top 15 guesses for every user. We further recover three pairs of the same details typed by different users. Nevertheless, human behavior impacts the recovery, as we always observe different results across distinct users typing the same CCN. Finally, the attack is successful on the three considered credit card providers, finding 6 / 12 AMEX, 3 / 6 Mastercard, and 4 / 12 Visa details within 1,000 guesses.

Compared to emulation (§V-B), the attack performs worse against human users. This trend occurs because users take suboptimal paths. When typing CCNs, users take the optimal path between keys only 89.35% of the time. The framework considers these suboptimal paths with a discounted score (§IV-B3), causing guesses that use suboptimal paths to have a lower rank. Despite this phenomenon, the attack still successfully discovers payment information typed by human users.

C. Password Recovery

Users enter passwords from the PhpBB leak [41] into both TVs. Figure 11 displays the attack's results. With the PhpBB prior, the attack reaches a 33.00% (AppleTV) and 60.19% (Samsung) top-100 accuracy. These results far exceed random guessing from the set of 184,388 PhpBB passwords. Thus, both platforms leak user-typed passwords through audio. The choice of prior is significant; the top-100 accuracy on Samsung drops by over $3.4\times$ when the attacker instead uses the RockYou prior. Nevertheless, under the RockYou prior, the attack still finds passwords over $100\times$ more often than random guessing on the Samsung TV. Thus, the TV's audio provides significant information to discover passwords in practical settings.

The attack performs worse on the AppleTV because users take more suboptimal paths on this system. On AppleTV, users traverse the optimal path between keys only about 45.35% of the time; this rate is 85.94% on the Samsung TV. We hypothesize that this discrepancy occurs due to the sensitivity of the AppleTV remote's navigation touchpad (Figure 1). These suboptimal path rates also explain the dropoff between these results and those in emulation (§V-C).

The attack handles the challenges of distinct users and diverse passwords. On the Samsung TV, the attack finds at least two passwords in the top 10 guesses for *every* user with the PhpBB prior. Further, when considering unique passwords, the attack has a top-100 accuracy of 42% (21 / 50) for *both* typing users. Thus, the attack is successful across users even after controlling for the typed string. We also observe the ability to recover passwords with various characters, achieving top-100 accuracies of 59.62% (special), 63.79% (numeric), and 43.75% (uppercase) with the PhpBB prior on the Samsung TV. These top-100 figures are 9.62 % (special), 18.97% (numeric), and 12.50% (uppercase) with the RockYou prior.

The attack splits Samsung keyboard instances using timing (§IV-A3). This method correctly identifies 98 / 100 passwords, where the two failures result from long pauses while typing. Further, we infer when a subject types a password (§IV-B1), and this classifier has an accuracy of 99.02% on this set. The only misclassification occurs on the password `naarf666` due to the suffix with repeated characters. This typing pattern matches that of dynamic keyboards, which produce better suggestions toward the string's end. These failures, however, represent a vast minority of cases, showing how the attack correctly handles instance splitting and keyboard classification.

This performance matches similar keystroke side-channel attacks in other contexts. For example, the MoLe attack uses smartwatch accelerometers to infer keystrokes on mechanical keyboards [63]. MoLe involves similar dynamics to Smart TVs, as it recovers strings from keyboard movements. For English words from a known set, the MoLe attack achieves 50% accuracy within 24 guesses. Our attack has a top-5 accuracy of up to 51.46% on the Samsung TV. Note that our setting is more challenging, as the PhpBB set is over 35× larger than the English dictionary used in this prior work.

Overall, these results provide strong evidence that ten users are sufficient to demonstrate this vulnerability. Under normality assumptions, the 95% confidence interval for the attack's top-100 accuracy on the Samsung TV is $61.48 \pm 1.97 \cdot 26.84/\sqrt{10} = (44.76\%, 78.21\%)$ with the PhpBB prior.[3] For the RockYou prior, the confidence interval is $(8.49\%, 26.99\%)$. The lower bounds of these intervals exceed random guessing by at least 150×. Thus, a larger study would likely not alter the main conclusion: Smart TV audio provides a *useful* signal for discovering user keystrokes.

D. Web Search Recovery

We apply the attack against web searches on the Samsung Smart TV. This scenario is unique because the attack must overcome dynamic keyboard suggestions (Figure 4).

Figure 12 displays the attack's accuracy. The attack finds 15% of words within 100 guesses. This result is lower than that of passwords (§VI-C) for two reasons. First, the attacker must predict the suggested key values (§IV-B3). Second, the attack infers that the user is typing an English word (§IV-B1), and

[3]This mean across users does not exactly match Figure 11 because the attack sometimes detects more than ten password entries for a subject.

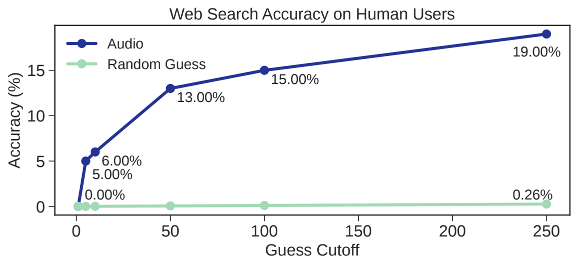

Fig. 12. Accuracy for human users typing web searches on the Samsung Smart TV with dynamic suggestions.

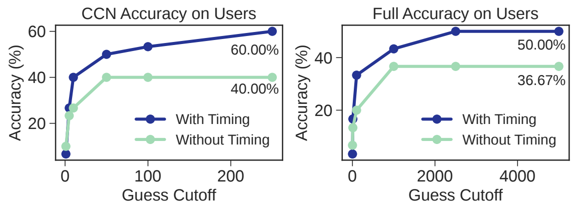

Fig. 13. Comparison between timing-based and exhaustive identification of suboptimal paths.

this classifier is only 42.71% accurate in identifying keyboards with dynamic suggestions. This low accuracy results from the classifier's designed bias toward passwords, as passwords are more sensitive. Thus, on 57.29% of the entries, the attack cannot recover the target string because it uses a keyboard without suggestions. If we force the pipeline to use keyboards with suggestions, the attack reaches 12.00% (top-10) and 27.00% (top-100) accuracy. These rates nearly double what we observe when inferring the keyboard type. Overall, the attack still vastly outperforms random guessing from the set of 95,892 English words, showing how an attacker can learn about user keystrokes on dynamic keyboards from the TV's audio.

E. Impact of Keystroke Timing

The attack leverages two timing properties to infer user behavior (§IV-C). This section evaluates these features.

The string recovery module infers the presence of suboptimal paths using timing (§IV-C1). We evaluate this feature by comparing the recovery of user-typed credit cards with and without timing-based identification. The baseline considers suboptimal paths at every move. As shown in Figure 13, keystroke timing improves the attack. The baseline reaches a 40.00% top-100 accuracy on CCNs and a 36.67% top-1000 accuracy on the full details. Under these cutoffs, the timing-based feature shows recovery rates of 53.33% (CCNs) and 43.33% (full details). This improved performance displays the benefits of using timing to identify suboptimal paths.

The attack infers movement directions during rapid scrolls (§IV-C2). We evaluate this feature by executing password recovery with and without direction inference. We focus on the Samsung TV, as this keyboard supports navigation in all four directions (Figure 3). For both priors, direction inference never hurts recovery rates. However, the benefits are modest. With the RockYou prior, only 3 of the 19 recovered passwords have a strictly better rank with direction inference. Nevertheless, we include this feature due these benefits.

We emphasize that the attack uses the same timing metrics for all ten users. The attack's success across different individuals highlights the generality of these features.

VII. Discussion

Improvements: The results from extracting passwords (§VI-C) underscore the language prior's significance. Thus, improving the language model is a promising method to improve this attack. Previous work shows that neural networks can learn effective language models over passwords [38]. We did not pursue these methods due to their computational demands. However, it may be possible to adapt the technique of [13] to make this neural network method more efficient, and we aim to explore this approach in the future.

We assume the attacker does not know personal information about the target user (§III). If we relax this assumption, an adversary with user-specific knowledge can improve our audio attack using a customized string prior. Furthermore, with access to personal information, the adversary may already know valuable details such as ZIP codes. These insights suggest that combining user-specific knowledge and Smart TV audio will result in a more effective attack.

Limitations: The presented attack only works against AppleTVs and Samsung Smart TVs. The attack does not directly apply to platforms that do not exhibit the acoustic properties in (P2) - (P4). For example, Roku devices do not make a distinct sound when deleting a character (P3), do not have sounds unique to the keyboard (P4), and mute the keyboard when rapidly scrolling. However, the incompatibility of our attack with these platforms does not guarantee the security of these other systems. Our work only shows these audio properties are *sufficient* for keystroke leakage; it does not prove these properties are *necessary*. In fact, prior work indicates that these properties are not all required. For instance, previous attacks infer when a user is typing on a Smart TV through movement timing [21], removing the need for (P4). We emphasize that our attack's success against Samsung Smart TVs constitutes a significant threat, as Samsung is the single most popular TV platform [66]. Extending this attack to other Smart TVs is a promising source of future work.

Our attack does not apply to every keyboard on the considered platforms. Some applications customize their keyboards and mute the audio, eliminating the acoustic side-channel. However, we show that the default system keyboards are insecure. Thus, applications that use the system defaults, such as the Samsung TV's browser, are vulnerable. Further, both TVs have areas (e.g., connecting to WiFi) where users enter sensitive information into the default keyboard.

Finally, some applications allow users to enter credentials over the Internet through an external device. This option bypasses our attack by avoiding the TV's keyboard. However, applications (e.g., Hulu) may not enforce that users take this option, and the alternative involves typing sensitive information (e.g., payment details) into an insecure Smart TV keyboard. Further, there are cases where users *must* type sensitive information into a virtual keyboard. For example, on the Samsung TV used in our evaluation (§VI-A), users must type their WiFi password with the TV's default keyboard. There is no option to enter the credentials through an external device. Thus, Smart TVs contain numerous practical settings vulnerable to our acoustic keystroke attack.

Defenses: A complete defense is to mute the Smart TV's audio when typing, especially for sensitive fields. However, this defense can harm the user experience. We describe three possible alternatives.

First, prior work randomizes the layout of virtual keyboards in a structured manner to eliminate knowledge of the keyboard design [47]. This approach limits the negative impact on user experience by only adding gaps and swapping rows. While our attack can model this randomization under the framework of suboptimal paths, we can no longer use timing to detect such paths (§IV-C1). We estimate the impact of this defense by randomly adding movements to generated move count sequences for credit card security codes (CVVs). The top-100 recovery rate falls to 30.32% in emulation, compared to the 100% rate achieved under current conditions. These figures suggest this defense reduces, but does not eliminate leakage.

Second, our attack leverages how Smart TVs initialize the cursor to the same key (P1). Thus, a possible defense involves randomizing the start position. This strategy, however, is flawed because some fields (e.g., CCNs) require users to finish on `Done`. We can use this property to search strings *in reverse*, starting with `Done`. In emulation, we recover 99.88% (top-100) of CCNs even when randomizing the start position.

Finally, Smart TVs could use the same sound for all keyboard actions. This design thwarts our attack, as it breaks (P2) - (P4). Security, however, is not guaranteed. For example, it may be possible to discern keyboard movements and selections using timing, even if both make the same sound. Further study is required to prove the security properties of this technique.

VIII. Related Work

A. Keystroke Side-Channel Attacks

Prior work uses side-channels such as electromagnetic emanations [24], [60], reflections [7], smartphone accelerometers [37], [45], timing [58], and performance counters [67] to infer keystrokes on personal computers and mobile devices.

Three types of prior attacks deserve closer comparison to our work. First, previous methods infer keystrokes using acoustics from mechanical keyboards [6], [10], [14], [70], [71]. Our attack also uses audio, but we target Smart TVs. This setting differs from prior work because Smart TVs make a small number of distinct sounds and use virtual keyboards with a dedicated cursor. Thus, our attack must estimate the cursor's position, and errors in this tracking cascade to the remainder of the string. Mechanical keyboards have no cursor, and errors in identifying a single keystroke do not propagate. These differences require new string recovery methods.

Second, existing work discovers keystrokes on mechanical keyboards using smartwatch accelerometers [31], [34], [39], [63]. Similar to Smart TVs, these attacks track the user's keyboard position, and tracking errors will cascade. Unlike our work, these attacks use a different side-channel and can only measure the movement of one hand. This limited information leads to lower efficacy than observed in our work (§VI-C).

Third, and finally, HomeSpy steals user keystrokes by sniffing unencrypted infrared signals between Smart TVs and their remotes [21]. However, many Smart TVs support remotes that communicate over Bluetooth [35], [69], and HomeSpy cannot view such signals. Our attack works with any communication medium employed by the remote controller.

B. Security of Smart Devices

Previous work highlights the need for increased security on smart devices [19], [25], [46], compromising smart lights [52], device software platforms [16], [23], smart locks [20], and smart speakers [9], [29], [68]. Acar et al. [2] use DNS rebinding to remotely access devices. Other work uses the communication patterns of smart devices to infer user behavior [5], [59]. Unlike these systems, our attack targets Smart TVs.

Previous studies analyze the security of Smart TVs, finding that platforms contain privacy violations [42]. SPOOK gains root access to Smart TVs through a weakness when pairing virtual remotes [35]. EvilScreen compromises Smart TVs using issues supporting multiple communication protocols [69]. Other work exploits software vulnerabilities to control Smart TVs [1], [40]. Enev et al. [15] use energy consumption to infer the TV's content. Unlike this prior work, our attack leverages the audio from Smart TVs to discover user keystrokes.

C. Attacks on Passwords and Payment Details

Prior work guesses passwords in offline settings with language models [33], [38], [44], [64] and text transformations [30]. Our attack also applies language models to measure password likelihood. However, we use these models to complement Smart TV audio. Wang et al. [62] use personal information to guess passwords under rate limits. This work improves our attack, as personalizing the language prior would increase string recovery from Smart TV audio.

Existing attacks compromise credit cards through skimming [50], [56] and relay attacks [26], [51]. Bond et al. [11] attack ATMs by exploiting weak random number generation. Our work also targets credit card details, although our attack steals this information from card-not-present transactions on Smart TVs. Further, our attack complements previous vulnerabilities on physical cards. For example, the credit card's magnetic stripe does not contain the printed CVV [56]. Our attack steals the CVV from Smart TV forms. Thus, adversaries launching attacks on physical cards can gain valuable information by also exploiting Smart TV virtual keyboards.

D. Audio Identification

Existing systems propose methods for audio analysis. Shazam uses spectrogram peaks for music identification [61]. Other work uses neural networks to analyze human speech [8], [27], [49] and environmental sounds [48], [53]. Our attack uses a nearest-neighbor classifier to detect sounds from Smart TVs. Our contribution with this approach is not in developing new signal processing techniques. Indeed, prior work on audio processing may exhibit more advantageous properties (e.g., resistance to background noise). Through our procedure, we instead show that it is *possible* to identify Smart TV sounds algorithmically. An attacker can then leverage this information to recover keystrokes (§VI). Note that our framework supports any accurate audio identification procedure; changing the audio extraction method does not alter the essence of the attack.

IX. Conclusion

Smart TVs allow users to enter information through on-screen virtual keyboards. Users interact with these keyboards by sequentially scrolling across keys via directional commands on a remote controller. Popular Smart TVs, such as Apple's tvOS [4] and Samsung's Tizen [55], make sounds as users type. This work develops and demonstrates a new side-channel attack against Smart TVs that uses these sounds to expose user keystrokes. Our attack is based on how the AppleTV and the Samsung TV produce different sounds when the user (1) moves between keys, (2) selects a key, and (3) deletes a character. Using these sounds as input signals, an attacker can extract the number of movements between keyboard selections. Our attack combines this extracted information with a prior over possible strings to identify likely keystrokes. Under ideal conditions in an emulated environment, the attack recovers 99.95% of credit card details within 1,000 guesses and up to 99.10% of common passwords within 100 guesses. On ten realistic users, the attack achieves a top-1000 accuracy of 43.33% on credit card details and a top-100 accuracy of up to 60.19% on common passwords when typing on a popular Samsung Smart TV. This attack was acknowledged by Samsung, and it demonstrates how Smart TVs must consider the privacy implications of all features interacting with sensitive user data.

Acknowledgment

We thank our anonymous Shepherd, reviewers, and artifact evaluators for their helpful feedback to improve this paper. This work was supported by the National Science Foundation (NSF) through the grants CCF-1822949, CCF-2119184, CNS-1764039, CNS-1956180, CNS-2313190, and CNS-2124393.

References

[1] Y. Aafer, W. You, Y. Sun, Y. Shi, X. Zhang, and H. Yin, "Android SmartTVs vulnerability discovery via log-guided fuzzing," in *USENIX Security Symposium*, 2021, pp. 2759–2776.

[2] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based attacks to discover and control local IoT devices," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, 2018, pp. 29–35.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the Mirai botnet," in *USENIX Security Symposium*, 2017, pp. 1093–1110.

[4] Apple, "tvOS," https://developer.apple.com/tvos/, 2023, Accessed: 03-2023.

[5] N. Apthorpe, D. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the smart home private with smart(er) traffic shaping," in *Symposium on Privacy Enhancing Technologies*, Stockholm, Sweden, jul 2019, pp. 128–148.

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

[7] M. Backes, T. Chen, M. Dürmuth, H. P. Lensch, and M. Welk, "Tempest in a teapot: Compromising reflections revisited," in *IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 315–327.

[8] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 449–12 460, 2020.

[9] M. Barnes, "Alexa, are you listening?" https://labs.withsecure.com/publications/alexa-are-you-listening, 2017, Accessed: 03-2023.

[10] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006, pp. 245–254.

[11] M. Bond, O. Choudary, S. J. Murdoch, S. Skorobogatov, and R. Anderson, "Chip and skim: Cloning EMV cards with the pre-play attack," in *IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 49–64.

[12] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[13] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," in *USENIX Security Symposium*, 2019, pp. 267–284.

[14] A. Compagno, M. Conti, D. Lain, and G. Tsudik, "Don't Skype & type! acoustic eavesdropping in voice-over-IP," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 703–715.

[15] M. Enev, S. Gupta, T. Kohno, and S. N. Patel, "Televisions, video privacy, and powerline electromagnetic interference," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 537–550.

[16] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy*. IEEE, 2016, pp. 636–654.

[17] P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. K. Greene, and M. F. Theofanos, "Digital identity guidelines," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST Special Publication 800-63B, Includes updates as of March 2, 2020, 2017.

[18] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 2013, pp. 6645–6649.

[19] W. He, V. Zhao, O. Morkved, S. Siddiqui, E. Fernandes, J. Hester, and B. Ur, "SoK: Context sensing for access control in the adversarial home IoT," in *IEEE European Symposium on Security and Privacy*. IEEE, 2021, pp. 37–53.

[20] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart locks: Lessons for securing commodity internet of things devices," in *Proceedings of the 11th ACM on Asia conference on computer and communications security*, 2016, pp. 461–472.

[21] K. Huang, Y. Zhou, K. Zhang, J. Xu, J. Chen, D. Tang, and K. Zhang, "HomeSpy: The Invisible Sniffer of Infrared Remote Control of Smart TVs," 2023.

[22] S. C. M. Insights, " Smart speaker market volume worldwide from 2015 to 2027 ," https://www.statista.com/forecasts/1367982/smart-speaker-market-volume-worldwide, Statista, 2022, Accessed: 03-2023.

[23] Y. Jia, B. Yuan, L. Xing, D. Zhao, Y. Zhang, X. Wang, Y. Liu, K. Zheng, P. Crnjak, Y. Zhang *et al.*, "Who's in control? On security risks of disjointed IoT device management channels," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1289–1305.

[24] W. Jin, S. Murali, H. Zhu, and M. Li, "Periscope: A keystroke inference attack using human coupled electromagnetic emanations," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 700–714.

[25] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni, "Understanding IoT security from a market-scale perspective," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1615–1629.

[26] Z. Kfir and A. Wool, "Picking virtual pockets using relay attacks on contactless smartcard," in *First International Conference on Security and Privacy for Emerging Areas in Communications Networks*. IEEE, 2005, pp. 47–58.

[27] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, "A clockwork RNN," in *International conference on machine learning*. PMLR, 2014, pp. 1863–1871.

[28] R. Kulkarni, "One week of global news feeds," https://www.kaggle.com/datasets/therohk/global-news-week, 2020, Accessed: 04-2023.

[29] M. Kunze, "Turning google smart speakers into wiretaps for $100k," https://downrightnifty.me/blog/2022/12/26/hacking-google-home.html\#the-fixes:~:text=remotely\%20control\%20it.-,A\%20cooler\%20attack\%20scenario,-Attacker\%20wishes\%20to, 2022, Accessed: 03-2023.

[30] E. Liu, A. Nakanishi, M. Golla, D. Cash, and B. Ur, "Reasoning analytically about password-cracking software," in *IEEE Symposium on Security and Privacy*. IEEE, 2019, pp. 380–397.

[31] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1273–1285.

[32] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin, "A measurement study of authentication rate-limiting mechanisms of modern websites," in *Proceedings of the 34th annual computer security applications conference*, 2018, pp. 89–100.

[33] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 689–704.

[34] A. Maiti, O. Armbruster, M. Jadliwala, and J. He, "Smartwatch-based keystroke inference attacks and context-aware protection mechanisms," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 795–806.

[35] J. Majors, E. B. Yi, A. Maji, D. Wu, S. Bagchi, and A. Machiry, "Security properties of virtual remotes and spooking their violations," 2023.

[36] R. . Markets, "Global Smart TV unit sales from 2018 to 2025 [Graph]," https://www.statista.com/statistics/878372/smart-tv-unit-sales-worldwide/, Statista, 2020, Accessed: 03-2023.

[37] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iPhone: Decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 551–562.

[38] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, lean, and accurate: Modeling password guessability using neural networks," in *USENIX Security Symposium*, 2016, pp. 175–191.

[39] Ü. Meteriz Yŷldŷran, N. F. Yŷldŷran, and D. Mohaisen, "AcousticType: Smartwatch-enabled cross-device text entry method using keyboard acoustics," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–7.

[40] B. Michéle and A. Karpow, "Watch and be watched: Compromising all Smart TV generations," in *IEEE Consumer Communications and Networking Conference*. IEEE, 2014, pp. 351–356.

[41] D. Miessler, "Leaked password databases," https://github.com/danielmiessler/SecLists/tree/master/Passwords/Leaked-Databases, 2021, Accessed: 04-2022.

[42] H. Mohajeri Moghaddam, G. Acar, B. Burgess, A. Mathur, D. Y. Huang, N. Feamster, E. W. Felten, P. Mittal, and A. Narayanan, "Watching you watch: The tracking ecosystem of over-the-top tv streaming devices," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 131–147.

[43] R. P. Muscatell, "Laser microphone," *The Journal of the Acoustical Society of America*, vol. 76, no. 4, pp. 1284–1284, 1984.

[44] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proceedings of the 12th ACM conference on Computer and communications security*, 2005, pp. 364–372.

[45] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "ACCessory: Password inference using accelerometers on smartphones," in *proceedings of the twelfth workshop on mobile computing systems & applications*, 2012, pp. 1–6.

[46] M. O. Ozmen, X. Li, A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering IoT physical channel vulnerabilities," in *Proceedings of*

the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022, pp. 2415–2428.

[47] W. Pak, Y. Cha, and S. Yeo, "High accessible virtual keyboards for preventing key-logging," in *International Conference on Ubiquitous and Future Networks*. IEEE, 2016, pp. 205–207.

[48] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *International Workshop on Machine Learning for Signal Processing*. IEEE, 2015, pp. 1–6.

[49] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with SincNet," in *IEEE Spoken Language Technology Workshop*. IEEE, 2018, pp. 1021–1028.

[50] M. Roland and J. Langer, "Cloning credit cards: A combined pre-play and downgrade attack on EMV contactless," in *7th USENIX Workshop on Offensive Technologies*, 2013.

[51] M. Roland, J. Langer, and J. Scharinger, "Applying relay attacks to Google Wallet," in *International Workshop on Near Field Communication*. IEEE, 2013, pp. 1–6.

[52] E. Ronen and A. Shamir, "Extended functionality attacks on IoT devices: The case of smart lights," in *IEEE European Symposium on Security and Privacy*. IEEE, 2016, pp. 3–12.

[53] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal processing letters*, vol. 24, no. 3, pp. 279–283, 2017.

[54] S. Sami, Y. Dai, S. R. X. Tan, N. Roy, and J. Han, "Spying with your robot vacuum cleaner: Eavesdropping via lidar sensors," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 354–367.

[55] Samsung, "Tizen TV," https://docs.tizen.org/platform/what-is-tizen/profiles/tv/, 2023, Accessed: 03-2023.

[56] N. Scaife, C. Peeters, and P. Traynor, "Fear the reaper: Characterization and fast detection of card skimmers," in *USENIX Security Symposium*, 2018, pp. 1–14.

[57] I. Semenov, "Wikipedia word counts," https://github.com/IlyaSemenov/wikipedia-word-frequency, 2022, Accessed: 04-2022.

[58] D. X. Song, D. A. Wagner, X. Tian *et al.*, "Timing analysis of keystrokes and timing attacks on SSH," in *USENIX Security Symposium*, vol. 2001, 2001.

[59] V. Srinivasan, J. Stankovic, and K. Whitehouse, "Protecting your daily in-home activity information from a wireless snooping attack," in *Proceedings of the 10th international conference on Ubiquitous computing*, 2008, pp. 202–211.

[60] M. Vuagnoux and S. Pasini, "Compromising electromagnetic emanations of wired and wireless keyboards," in *USENIX security symposium*, vol. 1, 2009.

[61] A. Wang, "The shazam music recognition service," *Communications of the ACM*, vol. 49, no. 8, pp. 44–48, 2006.

[62] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1242–1254.

[63] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "Mole: Motion leaks through smartwatch sensors," in *Proceedings of the 21st annual international conference on mobile computing and networking*, 2015, pp. 155–166.

[64] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 391–405.

[65] S. Wiedenbeck, J. Waters, L. Sobrado, and J.-C. Birget, "Design and evaluation of a shoulder-surfing resistant graphical password scheme," in *Proceedings of the working conference on Advanced visual interfaces*, 2006, pp. 177–184.

[66] B. Wire, "Smart TV streaming device market share worldwide as of 2020, by platform [Graph]," https://www.statista.com/statistics/1171132/global-connected-tv-devices-streaming-market-share-by-platform/, Statista, 2021, Accessed: 03-2023.

[67] B. Yang, R. Chen, K. Huang, J. Yang, and W. Gao, "Eavesdropping user credentials via GPU side channels on smartphones," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 285–299.

[68] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "DolphinAttack: Inaudible voice commands," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 103–117.

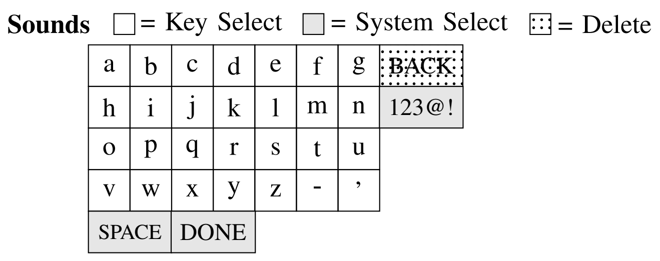

Fig. 14. The ABC keyboard layout. The background denotes the key's sound when pressed.

TABLE I
EMULATED TOP-K PASSWORD ACCURACY UNDER THE PHPBB PRIOR FOR DIFFERENT KEYBOARD LAYOUTS.

Layout	**Guess Cutoff (Top-K)**		
	$K = 1$	$K = 5$	$K = 10$
ABC	85.40%	99.55%	99.93%
AppleTV	88.97%	99.85%	100.00%
Samsung	84.67%	98.48%	99.03%
Random Guess	5.4×10^{-4}%	2.7×10^{-3}%	5.4×10^{-3}%

[69] Y. Zhang, S. Ma, T. Chen, J. Li, R. H. Deng, and E. Bertino, "EvilScreen attack: Smart TV hijacking via multi-channel remote control mimicry," *arXiv preprint arXiv:2210.03014*, 2022.

[70] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free attacks using keyboard acoustic emanations," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 453–464.

[71] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Transactions on Information and System Security*, vol. 13, no. 1, pp. 1–26, 2009.

APPENDIX A
SUPPLEMENTARY INFORMATION

A. Keyboard Layouts

We extend our evaluation by considering the efficacy of string recovery on different keyboard layouts. In addition to the default keyboards on AppleTV (Figure 2) and Samsung (Figure 3) devices, we study a third keyboard shown in Figure 14. This design resembles that of the YouTube application on Samsung Smart TVs. This keyboard does not have dynamic suggestions. We emphasize that the YouTube application mutes the keyboard and is not susceptible to the presented attack. Nevertheless, we include this layout to evaluate whether any popular keyboard designs form an effective defense in isolation. In all cases, we assume the keyboard layout remains fixed and is known to the attacker (§III).

Table I shows the top-K accuracy ($K \in \{1, 5, 10\}$) against common passwords under the PhpBB prior. We perform this evaluation in emulation and use the same password list as in §V-C. We observe that the keyboard layout has only a small impact on the attack's accuracy. For all three keyboard designs, the attack yields a top-1 accuracy above 84%; this figure is orders of magnitude higher than random guessing. We note that the AppleTV layout shows the highest recovery accuracy.

We hypothesize this trend stems from the linear keyboard design in which users only move in three directions to navigate between characters (Figure 2); e.g., when the cursor is on `a`, the user can only move ”left”, ”right,” or ”down.” In contrast, the other two layouts allow movements in four directions; this design means the attacker's search must consider larger neighbor sets. Nonetheless, the attacker still exhibits high recovery rates when targeting these two keyboards. These results suggest that the layout of a fixed keyboard does not constitute an effective defense against this attack.

B. *User Information*

The users in our study were aged 22 through 29 (mean 24.3). All ten users were either undergraduate or graduate students at our university. Six subjects identified as male, three as female, and one as non-binary. To the best of our knowledge, none of the users were cybersecurity experts. Every subject had previously used a Smart TV. Three users owned a Samsung device, and one owned an AppleTV. Our goal was to corroborate the feasibility of the attack as seen in emulation by collecting data from users who have some experience with Smart TVs. Thus, we did not record additional information about the subjects, as further details were not essential to this goal. We do *not* claim to provide a demographic or socioeconomic breakdown of typing patterns on Smart TVs.

APPENDIX B
ARTIFACT APPENDIX

A. *Description & Requirements*

1) How to access: The code for this project is available in a public GitHub repository: https://github.com/tejaskannan/smart-tv-keyboard-leakage. The included README file describes the setup and execution process. We provide links to the data and intermediate results in the “Benchmarks” section. The artifact is also registered under the Digital Object Identifier (DOI) `10.5281/zenodo.10151215`.

2) Hardware dependencies: There are no required hardware dependencies. However, if you have a Samsung Smart TV or Apple TV, you can use these devices to record new videos of typing on each system's virtual keyboard. You may then process these recordings with the provided code. Note that the code requires video for debugging purposes; the attack immediately isolates the audio and only uses acoustic information. We verified our implementation on an A1625 AppleTV with tvOS version 16.3.2 and a model UN55MU6300 Samsung Smart TV running Tizen software version T-KTMAKUC-1310.1. Other devices may exhibit differences.

3) Software dependencies: We have tested our code on a Ubuntu 20.04 system. The code requires Python 3.8 with Anaconda 23.0.1 and Java (`openjdk` version 17.05). We include the remaining dependencies (Python packages and Java libraries) in the GitHub repository. See the setup instructions in the repository's README for more details.

4) Benchmarks: We include two sources of data with this artifact.

(Source 1) Contains the dictionary priors, word lists, and results for every experiment.
(Source 2) Holds the recordings of subjects in our user study typing on Smart TV keyboards.

Refer to the GitHub README for the links to these data sources.

B. *Artifact Installation & Configuration*

We implement the attack using both Python and Java. The Python portion implements audio extraction, and the Java code implements string recovery. We include detailed installation instructions in the GitHub README.

The Python project is best managed using an Anaconda virtual environment. The file `environment.yml` contains the virtual environment configuration. Once you create the virtual environment with these dependencies, you can install the local package using `pip`.

The Java code uses `openjdk` version 17.05. The code relies on SQL, JUNIT, and JSON parsing libraries. The GitHub repository contains the JAR files for these repositories. To execute the Java code, reference these JAR files in your CLASSPATH environment variable.

The final piece of installation involves downloading the required data. You should fetch the entire folder containing the dictionary priors and precomputed results (Source 1); the directory is about 385 MB compressed and 1.1 GB uncompressed. To execute end-to-end experiments on real data, you will also need the video recordings from the user study (Source 2). Each video is large (e.g., roughly 1 GB), so we recommend starting with a single subject (e.g., Subject A).

C. *Major Claims*

- (C1): In emulation, our attack discovers over 99% (top-1000) of full credit card details and up to 84% (top-1) of common passwords. Experiment (E1) proves these results, as reported in Figures 8-10.
- (C2): Against real users, the attack infers over 50% (top-1000) of full credit card details, up to 42% (top-1) of common passwords, and 15% (top-100) of web searches. We prove these values in experiment (E2). Figures 11-14 contain the results validating this claim.

D. *Evaluation*

1) Experiment (E1): [60 human-minutes + 1 compute-hour]: This experiment generates new benchmarks on which to evaluate the attack in emulation. We then execute the attack on these benchmarks and analyze its performance.

[How to] The detailed execution steps are in the README under the sections AUDIO EXTRACTION → EMULATION, STRING RECOVERY, and ANALYSIS → ATTACK RESULTS IN EMULATION. We summarize this information below.

[Preparation] This experiment requires the dictionary priors and word lists (Source 1). There is no additional setup.

[Execution] There are two main steps to running this experiment: (1) generating benchmarks and (2) executing the string recovery.

1) The scripts `generate_[*]_benchmark.py` create move count sequences for lists of common passwords and credit cards. These move count sequences feed into the string recovery phase.
2) The Java program `SearchRunner.java` (in `search/smarttvsearch`) executes the string recovery on a single benchmark file. We provide shell scripts (`run_[*]_bechmark.sh`) to execute the recovery on multiple files.

[Results] The files `benchmark_[*]_recovery.py` (in the `analysis` folder) generate plots displaying the attack's performance on passwords and credit card details, respectively. The resulting plots should match Figures 8 (credit cards) and 9 (passwords). Even with changes to the exact benchmark data, it should be evident that the attack outperforms random guessing by orders of magnitude.

2) Experiment (E2): [60 human-minutes + 2 compute-hours]: This experiment processes recordings from our user study and executes the attack on this realistic data.

[How to] We provide detailed steps in the README under the sections AUDIO EXTRACTION → REAL RECORDINGS, STRING RECOVERY, and ANALYSIS → ATTACK RESULTS ON USERS. We summarize the instructions below.

[Preparation] This experiment requires the dictionary priors, word lists (Source 1), and user video recordings (Source 2). The individual videos are large, and we suggest running the experiments end-to-end on a single user first (e.g., Subject A). For convenience, we provide the move count sequences and final results for all users (Source 1).

[Execution] There are two main steps to running this experiment: (1) processing recordings and (2) executing the string recovery.

1) The scripts `make_spectrogram.py` and `split_keyboard_instances.py` extract move count sequences from recordings of Smart TVs. The first script isolates the audio. The second program identifies key sounds, finds instances of using the keyboard, and creates the move count sequences. The output from this step is a file of move count sequences in the same format as the benchmarks from (E1).
2) The search process mirrors that of the previous experiment. To execute the search on all users, we provide the shell scripts `run_user_[*].sh` (in the folder `search/smarttvsearch`). We note that the attack must consider suboptimal paths against realistic users. This behavior slows down the attack, and the string recovery can take upwards of 30 minutes for each information type.

[Results] The files `user_[*]_recovery.py` (in the `analysis` folder) display the attack's results on each information type for real users. The created plots should match Figures 10 (credit cards), 11 (passwords), and 12 (web searches).