Privacy Controls for Always-Listening Devices

Nathan Malkin
University of California, Berkeley
nmalkin@cs.berkeley.edu

Serge Egelman
University of California, Berkeley
International Computer Science Institute
egelman@cs.berkeley.edu

David Wagner
University of California, Berkeley
daw@cs.berkeley.edu

ABSTRACT

Intelligent voice assistants (IVAs) and other voice-enabled devices already form an integral component of the Internet of Things and will continue to grow in popularity. As their capabilities evolve, they will move beyond relying on the wake-words today's IVAs use, engaging instead in continuous listening. Though potentially useful, the continuous recording and analysis of speech can pose a serious threat to individuals' privacy. Ideally, users would be able to limit or control the types of information such devices have access to. But existing technical approaches are insufficient for enforcing any such restrictions. To begin formulating a solution, we develop a systematic methodology for studying continuous-listening applications and survey architectural approaches to designing a system that enhances privacy while preserving the benefits of always-listening assistants.

ACM Reference Format:
Nathan Malkin, Serge Egelman, and David Wagner. 2019. Privacy Controls for Always-Listening Devices. In *New Security Paradigms Workshop (NSPW '19), September 23–26, 2019, San Carlos, Costa Rica.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3368860.3368867

1 INTRODUCTION

As the Internet of Things grows, it will continue to provide both major conveniences and significant privacy challenges. One existing category of technologies that exemplifies these tradeoffs is intelligent voice assistants—for example Apple's Siri, Microsoft's Cortana, Google Assistant, and Amazon Alexa—which respond to users' voice commands, such as playing music, looking up information, or activating a smart-home device. These assistants are available in most smartphones, as well as in stand-alone smart-speakers, and their voice-enabled interfaces have also been added to conventional gadgets like televisions, microwaves [22], and even toilets [2]. In all of these form factors, the voice assistant operates by always listening for "wake-words" (such as "Hey Siri" or "Ok Google"), then recording and analyzing any audio that follows it, in order to extract (and act on) the user's instructions.

Even the current setup, with users explicitly triggering the device, presents privacy difficulties [10]. Most people do not understand when a smart device is listening and where it is sending data [9]. Some may not even realize that their recordings are stored in the cloud—though law enforcement certainly does, and has requested the collected data as part of investigations [12]. Advertisers have also sought to exploit the insights these devices offer into shoppers' lives. Patent filings from Amazon and Google have described designs for using data they collect from smart speakers for targeted advertising [18].

NSPW '19, September 23–26, 2019, San Carlos, Costa Rica

ACM ISBN 978-1-4503-7647-1/19/09…$15.00
https://doi.org/10.1145/3368860.3368867

1.1 Towards Always-Listening Devices

The patent filings provide clues to other features that we may expect to one day see in our assistants. One notable assumption is that the devices will be constantly listening and analyzing data [5]: for example, the assistants may be expected to detect a fire alarm going off, a baby crying, or a child "engaging in mischief" [8]. More simply, users may want to vary their sentence structure when addressing their devices ("turn on the lights, Siri" instead of "Siri, turn on the lights"). All of these features require the devices to expand the scope of their listening. Today, the device's microphone is already always on, but its task is very narrow: detecting the device's wake-word. Future use cases, however, will require listening for more things and performing more sophisticated analysis.

In fact, always-listening (or always-watching) devices are already on the market. As one example, the Google Clips camera takes pictures continuously and then selects the "best" photos among them [14]. Less sophisticated but more common, in-home cameras are marketed for a variety of purposes: security, watching pets, and surveillance of nannies and other domestic workers. Most are already connected to the Internet, and, with time, they too are likely to incorporate AI-powered or other "smart" features. Voice interfaces may similarly benefit from an "always-on" capability, and, as their features expand, it is highly likely that more people will welcome them into their homes without full knowledge of potential privacy ramifications.

1.2 New Privacy Challenges

What does it mean for consumers' privacy if a device is always listening? Rather than waiting for a wake-word to start capturing the audio surrounding the device, they will be "passively listening" to everything until they hear content relevant to their functionality. This calls for a shift in how we think about privacy on these devices—and in our homes. Previously, we opted in to interactions with our assistants, addressing them by name. Soon, the default will be for our speech to always be collected, and we need to think about how to opt out (i.e., block) the devices from gaining access to certain speech. But with the trigger word gone, users will have even less insight into when these devices are capturing or processing audio,

significantly increasing the chances of a device capturing audio in unexpected—and potentially privacy-invasive—circumstances.

Lack of transparency will not be the only privacy issue for future passive listening devices. Depending on where in the home they are located, they can listen in on different conversations with varying degrees of sensitivity. Based on what they hear, they will be able to infer highly private and intimate details about our lives. This further raises difficult questions of whether this data is stored and who has the opportunity to review it. Compounding the challenges, multiple people could be involved in a conversation, each with different privacy expectations and preferences. Many, due to their status in the household, may lack any administrative access or accounts with the service. Some, like visitors, might not even know about the device's presence. Meanwhile, as assistant-enabled devices become smaller and more ubiquitous, their form factor and hardware limitations will make privacy indicators and feedback cues less feasible.

The picture is further complicated by the ecosystem that will surround these devices. We envision the ecosystem following the path of smartphones, where manufacturers will serve as a mediator (like the Android and iOS operating systems do) for a multitude of third-party applications. These apps will provide narrowly defined functionality, such as calendaring, ride sharing, etc. Some voice assistants already support third-party functionality (e.g., "skills" for Alexa), and are actively promoting their platform to developers.

What will the privacy protections be when these devices shift to passive listening and start sharing audio with third parties? Will the third parties also get full access to all audio from within our homes? Because of liability and reputational concerns, platforms themselves will likely want to impose restrictions on the data third parties collect. The question—and challenge—is, how? How can a passively-listening device identify when it should or should not be listening? How can a person specify which conversations are fair game for an app and which are private? Can a conversation be appropriate for one app while being inappropriate for another?

Answering these questions requires new approaches for both privacy and security. While existing paradigms—from capability-inspired permission systems to access control—present a full spectrum of options, they are by and large inadequate for the problems posed by passively-listening devices. Granting a "microphone permission" to Siri or Alexa does nothing to limit what it might hear and how it will use the data. Instead, we argue that privacy controls must make use of the content of communication rather than metadata about it. These would expand on the capabilities of existing permission systems, which do not examine data directly, but rely on metadata about it to determine if flows are appropriate. However, these methods may have only limited applicability to speech controls, since conversations may have identical attributes, differing only in their content.

In this paper, we begin to explore the design space for possible solutions to the problem of privacy for always-listening devices. We first describe how passive listening applications might work, and provide examples, in order to create a clear picture of the problem we are trying to solve. We then enumerate several potential approaches for ensuring end-users' privacy when interacting with passive listening apps and discuss the trade-offs of these approaches. We also examine how the different approaches can be evaluated and compared. Taken in concert, we hope that our work represents a first step towards a future where—though our devices might inevitably be always listening to us—they are able to respect our privacy wishes in the process.

2 SCOPE & ASSUMPTIONS

Trying to impose limits on the behaviors of an always-listening device inevitably raises questions of trust. Suppose we develop a "permission system" for always-listening devices. Where should it run? Is it external to the smart speaker? If so, who will operate it? Why are they to be trusted? How will it stay up-to-date? How will it work when the voice assistant takes other form factors? Alternately, the privacy controls may be built in to the device itself. But can the platforms really be trusted to police themselves?

These problems are vast and complex; to make progress in this space, we begin by defining a threat model where the smart speakers themselves are trusted, as are the platforms that operate them. These platforms, however, are open to third-party applications, and these are the ones from whom we want to protect users' privacy.

In other words, under our threat model, the manufacturers and their software are fully trusted with all audio their devices may overhear. This does not mean these platforms are incapable of violating users' privacy; indeed, there are a multitude of ways for them to do this, either for their own purposes (e.g., selling information about users) or for others (for example, state-sanctioned backdoors or surveillance). Yet, in trying to define a trusted computing base, the line must be drawn somewhere, and the hardware manufacturer is a reasonable place to start. We hope that eventually these assumptions can be relaxed, but for now, we will trust the hardware and its operating system, including any first-party functionality, and assume that they can safely hear everything that happens around the device.

Whom *don't* we trust, then? We believe that, as they are today, assistant platforms will be open to third-party developers, who will create apps that provide certain functionality which requires an always-listening smart speaker. If such an app were a standalone device, it would receive all audio from the microphone—24/7—and if it happened to be malicious, it would use that audio towards some nefarious ends. Our goal is to develop a system that can prevent this sort of abuse by enforcing limits on the audio an always-listening app may access.

We believe this is a realistic and practical threat model. First, it matches how existing ecosystems have developed, for example, mobile operating systems rely on central app stores to offer third-party-provided functionality. This also matches how existing voice assistants have positioned themselves: both Amazon and Google already allow third-party "skills" for their voice assistant and encourage their creation by, for example, offering hundreds of thousands of dollars in prizes to developers [6].

Overall, this architecture is likely to be more beneficial to privacy, as there is a single trusted party controlling the microphone. The alternative is for each "smart" device to have its own microphone and custom logic about when it listens and how it reacts. This means that each device would be responsible for privacy on its own. As we know from the sad state of IoT security, this may not be ideal.

Some further simplifying assumptions that we will make—in the interest of making the scope of the problem more manageable—is assuming that the passive-listening devices in question will be targeted at consumers (rather than businesses), located in a home environment, and fixed (rather than portable). However, we note that passive listening in the workplace presents its own interesting set of challenges, from both a technical and security perspective, some of which have begun to be explored in the literature [11].

3 CURRENT VOICE ASSISTANTS

Smart speakers with passive listening apps are unlikely to just show up, in the near future, in industry showrooms and on store shelves. The technology, and specifically the current state of natural language processing, is simply not ready yet, though it is advancing rapidly. Consequently, we may reasonably expect a relatively gradual progression in functionality from today's intelligent voice assistants to those with more passive capabilities. As such, it may be useful to review the current behavior of voice assistants and smart speakers, as a sort of baseline for any future developments.

Large numbers of people have embraced smart speakers and other intelligent voice assistants: reports suggest that 86 million smart speakers were sold in 2018 alone [21]. Intelligent voice assistants come embedded in a variety of devices: smart speakers and displays (e.g., Amazon's Echo and Echo Show), laptops, smartphones, smart plugs, and many other consumer electronics [2, 22]. Regardless of the specific form factor, these share two things in common: an always-on microphone and an Internet connection.

An on-device speech model has been pre-trained to identify the assistant's wake-word among the ambient sounds. When the wake-word has been recognized, the device records the subsequent audio until a pause in speech or a timeout has been reached. The speech is then sent to the assistant's web servers, where it is processed, analyzed, and the requested action (if understood) is triggered.

The exact processing pipeline used by popular IVAs remains proprietary, but common steps include automatic transcription (going from speech to text), domain detection (understanding the general type of query; e.g., it is about travel), intent detection (recognizing the user's specific goal; e.g., booking tickets), and slot filling (inferring the parameters of the query; e.g., destination and date for the tickets). These steps may not necessarily happen sequentially; feedback from later stages (such as slot filling) may result in updates, for example, to the understanding of the intent. Alternately, the stages may happen all at once, with a single model trained to perform domain detection as well as slot filling [15].

In addition to one-shot interactions (a request or query followed by a response), IVAs have also started introducing more complex interactions. Some may keep a small amount of state [16]: for example, after inquiring about an artist, a user can ask "when were they born?" without naming the person again. IVAs can also engage in rudimentary dialogue, usually asking for confirmation or follow-up questions. Notably from a privacy perspective, when a follow-up question is asked, the microphone "opens" (i.e., starts recording) directly, without hearing a wake-word.

The audio, its inferred transcript, and the assistant's response are stored by the companies indefinitely by default. Users are able to review these interactions and delete them, though many do not know about these capabilities [10]. Besides this, the only privacy control available on a device is a physical mute button or switch. This is typically available on smart speakers, but fewer other assistant-enabled devices.

Third parties and their capabilities. The major voice assistants today allow third-party developers to provide additional functionality for the voice assistants. These are called "skills" for Alexa and "Actions" for Google. To implement this functionality, developers are provided with a declarative API, which they use to list and provide examples of the intents their app fulfills and the slots (parameters) they expect to fill. Most often, users must invoke a skill directly, e.g., "Alexa, tell SmartHome App to turn on the lights." However, certain skills may be invoked automatically, without the user uttering their name, if the platform detects a user's intent as matching the one implemented by the app [3]. When an app is invoked, it is provided with the parsed data as well as a transcript of the original utterance. At present, third party apps for both Amazon and Google do not get access to the underlying audio for their requests.

Privacy and security challenges. Existing smart speakers pose privacy problems along a variety of dimensions. The device must be trusted to only listen for its wake-word (and record only after it is said) and not the rest of the time. Even if it does this faithfully, accidental activations can occur, often without the user's knowledge; in some cases, this has even led to entire conversations being shared with third parties [17]. Adopting our paper's threat model (i.e., focusing only on the risks of third-party apps), possible attacks include "skill squatting" [7] (malicious apps impersonating legitimate ones by adopting similarly-sounding names). Skills where the assistant automatically selects the appropriate app based on the request ("ok Google, hail me a cab") may also be targeted by attackers for impersonation. Such attacks, as well as the information targeted by attackers and their motivation, may carry over to always-listening apps as well.

4 APP CAPABILITIES AND DESIGNS

Our goal is to envision privacy solutions for passive listening apps, but none exist today which could be analyzed for their properties and capabilities. Therefore, as a first step, we will define what these capabilities may be. Understanding the variety of designs and use cases for these apps will help us ensure that any proposed solutions are able to address the full spectrum of features these apps may develop.

4.1 What Functionality Will Apps Provide?

Since we are discussing technology that is largely hypothetical, we can only speculate on the needs developers may seek to address. We have come up with examples that—while far from exhaustive—are plausible and representative of the potential use cases. Appendix A lists a range of sample apps, along with utterances or conversations that may trigger them. Some examples include:

- A calendaring app, which picks up on plans you make and adds them to your calendar automatically
- A foreign language learning tool, which can help when you're speaking a foreign language by correcting mistakes or suggesting vocabulary

- A "swear jar" to keep track of how many times you say undesirable words
- A kitchen helper, which can keep track of ingredients and answer questions about recipes
- "Artificial memory" that saves and organizes information (such as names and birthdays) mentioned in a conversation for easier recall
- A baby monitor can alert you to when your child is crying (or, when they grow older, if they're fighting with a sibling)
- A music DJ, which adjusts the music in the room based on the mood of the party (as inferred from the conversations)

4.2 What Audio Do Apps Need to Provide?

The examples above represent a range of use cases for always-listening apps. What audio would each of them want to capture to provide the necessary features? This depends on an app's specific functionality, of course, but more generally on:

(1) How are apps invoked?
(2) How do apps interact with users?

4.2.1 How are apps invoked? Techniques can range from today's direct invocations via wake-words to fully-passive listening.

Direct address with wake-word. This is the way assistants and their skills are triggered today; for example, "Mycroft, turn on the lights." While we expect this to remain the most popular way of invoking apps, existing paradigms may offer sufficient privacy protections. In particular, the fact that users are directly naming the app they want to summon means privacy violations are likely to happen only in the event of accidental invocations.

Flexible trigger-word invocation. A variant of the wake-word approach is allowing users to construct more natural-sounding queries where the trigger word need not be the first word in the sentence: "can you turn on the lights, please, Alexa?" While the addressee is still semantically clear, identifying it requires more sophisticated analysis.

Call to action without trigger word. A variant of the familiar app invocation may happen when a user issues a call to action that is not preceded by a trigger word. For example, instead of saying, "Computer, louder" or "Computer, pause" the user may want to simply say "louder" or "pause."

Purely passive (just listening). Apps may listen for speech that is not directed at them, but may still be relevant to their functionality. (We refer to these as "passive-listening apps," in contrast to the broader category of always-listening apps, which may use any of the other invocation modes.) For example, they could keep track of the conversation to have context if they are eventually invoked (e.g., knowing which song people are discussing when they decide to request information about the artist), or they could take actions silently (e.g., making a note that the household is out of milk based on someone's comment, without speaking up and interrupting the conversation). Compared to other invocation modes, the relevant speech here may be longer, have fewer keywords, more ambiguous grammatical structures, and require more context.

4.2.2 How do apps interact with users? Another important variable to consider is the kind of feedback apps provide to users: do they respond immediately or take an action in the background? This design decision has import implications for the user's acceptability to detect an accidental or privacy-violating invocation.

Provide immediate responses. This is how today's voice assistants behave. This provides an opportunity for immediate feedback (audio or visual) that a particular skill was invoked.

Engage in dialog with the user. In addition to providing an immediate response, the app may ask for confirmation or clarifying questions. This has the privacy advantage of providing immediate feedback, but is technically challenging, as the app must maintain state and context.

From a privacy perspective, the implication of this interaction mode is that the system has to not only detect the initial invocation, but understand whether any follow-up utterances are directed at the app. (This is another classification problem that malicious apps could try to exploit.)

Background action or no response. In this mode, the app does nothing, or performs an action through a channel that is not the system itself. Conversely from the previous mode, this might have challenges as far as feedback is concerned, especially in combination with the passive listening (non-)invocation mode.

4.3 How Are New Apps Installed?

Other considerations in the design space include whether apps are installed through a visual interface or by talking to the device. The latter case presents additional challenges, as it significantly limits the channels for presenting information to the user, such as a privacy notice or any additional information about the app.

4.4 Legal and Ethical Considerations

Always-listening devices raise a host of legal and ethical issues. This paper does not aim to explore these exhaustively, nor resolve them; we encourage other scholars to take up these questions. Here, we provide a sampling of the issues that will need to be addressed.

GDPR and other privacy laws. The European General Data Protection Regulation (GDPR) contains a number of provisions that may affect the deployment of always-listening devices. How do requirements like data minimization and limitations on storage interact with the smart speaker's need to hear and analyze all audio around it? Can the device be transparent about what it heard without creating a conversation log that may run afoul of further regulations (and may make many users uncomfortable)? What additional liabilities are added by other privacy laws, like the "Right to Be Forgotten"?

Wiretap laws. The United States currently lacks comprehensive privacy legislation like Europe's GDPR (though some narrowly targeted laws, like COPPA, the Children's Online Privacy Protection Act may still be applicable). However, a number of individual states have laws against wiretapping, requiring all parties to consent if their conversation is to be recorded. (Remaining states are single-party consent states, where only one party needs to provide consent.) How does an always-listening device collect consent?

Informed consent. Whether under the US or European regulatory regime, informed consent is likely to be a core requirement for the deployment of always-listening devices. But how should an assistant obtain consent? Current smart speakers sidestep these questions by requiring their owners to obtain consent from all parties who might be recorded. (They are typically located in the home, making this a more tractable task.) Will always-listening devices be able to get away with the same requirements, or will they need to be more proactive about obtaining consent?

Ethical concerns. Always-listening devices are a potent force for surveillance, both within a household and on a larger scale, by companies and states. Having access to an individual's conversations (especially in a private place, like one's home) is ripe for abuse; the possibilities are, quite literally, Orwellian. Even outside any dystopian visions, a device that is privy to so many conversations is certain to exacerbate any power imbalances, both between people within the home and between consumers and the companies providing their services. Without appropriate checks and balances, they are likely to exacerbate any power imbalances. In light of all these concerns, it is reasonable to ask whether always-listening devices are even ethically desirable.

Regardless of one's stance on that particular question, always-listening devices are all but inevitable. As we have seen, in many ways, they already here, even if law and ethics have not fully caught up. We believe that robust regulations will be vital to preventing privacy violations. However, our hope is that we can develop technical guarantees that can work in concert with strong legislation to constrain the capabilities of always-listening devices. This is the focus of our research and the remainder of this paper.

5 ATTACKER MODEL

So far we have discussed the behavior and capabilities of benign apps. However, some apps may be privacy-invasive, whether accidentally or due to malevolence. Our goal is to protect against these malicious applications. To do this most effectively, it will help to understand what constitutes an attack, the attackers' motivations, and the information they may target.

5.1 Defining Attackers and Privacy

As the goal of our system is to ensure users' privacy, an attack is any action that results in a privacy violation, and an attacker is someone who engages in these actions. Of course, this merely invites the next question: what constitutes a privacy violation?

Traditional permission systems, such as those used by smartphone operating systems, focus privacy controls on a handful of "sensitive" data types, such as a user's location or their contacts. However, in reality, there are many more types of information whose leakage may be considered a privacy violation by users. This is because it is impossible to divide information into "sensitive" and "not sensitive." People freely share even "sensitive" information, like health facts, in certain situations, e.g., with doctors, support groups, and families. On the other hand, they may consider their purchases not sensitive, yet object to the sharing of their shopping habits. These apparent contradictions make sense if we observe that people feel upset when their information is shared in a way that runs counter to their expectations.

This notion is formalized in Helen Nissenbaum's theory of privacy as contextual integrity (CI) [13]. CI models information flows with variables including the data type, the subject (whom the information is about), the sender, the recipient, as well as the transmission principle (any stipulations or conditions attached to the sharing). Each flow occurs within a specific context, which is governed by communal norms and individual expectations. Privacy violations occur when any of these variables change and information starts flowing contrary to established expectations. For example, people may feel that their privacy has been violated if their information gets passed along to a new party (a change in the recipient), or if their identities become attached to previously anonymous data (a change in the transmission principle).

The contextual integrity model has important implications for our quest to design privacy controls for always-listening assistants. It suggests that it is not enough for our system to prevent access to the data types enumerated above (even if this were feasible). Instead, it must strive to ensure the contextual integrity of conversations: that the applications only get access to the data that is relevant to their purpose, and everything else remains off-limits.

Contextual integrity also illuminates a type of privacy violation that may be almost impossible to prevent: information collected for one purpose being used for another. A simple example is data collected for the purpose of fulfilling functionality being repurposed for advertising. This violates contextual integrity because the information flows beyond the original recipient to a third party for an unintended purpose. If the data in question is speech and the recipient is an always-listening app, our platform may be powerless (from a technical point of view) to curb such further spread of information once it has left the system. Yet, this is still a very important consideration, since any such privacy violations will necessarily impact the overall trustworthiness of the system as a whole. This problem is likely most amenable to non-technical solutions: careful vetting of developers and terms, contracts, and legislation that specify consequences for misuse of data.

5.2 Attacker Motivations

Contextual integrity suggests an intimidatingly large set of privacy violations: anything outside established norms of information flow. However, an attacker generally does not set out to violate privacy for the sake of violating privacy. Instead, they are guided by their own personal (or organizational) reasons. If we understand these motivations, then we may be able to deploy more targeted defenses. Thus, it is worth considering the question: what are attackers trying to achieve?

Marketing. In today's economy, a common cause for privacy violations is the developer's desire to collect user data for advertising purposes. We expect similar motivations will drive many privacy-violating always-listening apps. The data may include specific facts (e.g., location, gender) or inferences that can be made about the individual (e.g., interests, income). This information may be used directly by the developer or sold to a data broker.

Theft of private information. Another common motivation for malware is theft of secrets that can be valuable on the black market. This is typically financial information (e.g., credit card numbers,

bank accounts) but can also include personal identification numbers (e.g., Social Security Number or equivalent) and account usernames and passwords. An always-listening device is likely to eventually pick up this sort of information, making it a particularly attractive target for attackers.

Profiling household occupants. It is possible that software may want to profile and understand the habits of the people living in a household for purposes other than advertising. Example scenarios include a utility or government agency wanting to ascertain how many people are living in a particular household or a criminal syndicate deciding which houses to burglarize. While the latter appears far-fetched (it does not scale, and the economics appear questionable), the former is drawn from real-world privacy concerns surrounding smart meters.

Reputational damage. Some attackers may be motivated by causing reputational damage to their victim—whether a specific individual (e.g., politician, celebrity) or a class of people based on some characteristic or behavior. For this type of attacker, there is no single "data type" that they may be trying to collect (though any data type from above may be considered useful); instead, lifestyle or ambient information may be targeted.

Unintentional privacy violations. Finally, it is important to consider that privacy violators may actually be well-intentioned, and their violations are entirely accidental (or, less charitably, due to negligence), for example, due to a poorly-trained classifier. In fact, this is currently the primary source of privacy violations by voice assistants, and we expect mistakes to remain the dominant cause of privacy violations due to inherent difficulties in natural language processing (and software engineering more generally). How do we prevent these mistakes from happening? Defending against more targeted attacks can help protect against accidental ones as well; however, there may be more specialized techniques that can be used if we assume an app is simply confused rather than malicious.

5.3 Information Targeted by Attackers

Having examined attackers' motivations, we now consider whether there is certain information they may be especially driven to obtain.

Indiscriminate data collection. One way an attacker may seek to satisfy their goals is by collecting any and all data they can get their hands on, storing it, then mining it for useful information later. (This may include any speech, including non-primary languages, as well as nonverbal sounds.) The difficulty of accomplishing this will depend in large part on the design of our always-listening platform.

Targeted collection. Indiscriminate collection is harder to conceal, especially in the presence of any counter-measures, and may also be difficult to scale. As such, attackers may limit their collection to only data they consider valuable (based on their specific motivation). The following attempts to enumerate the specific data types attackers might be after:

- Demographic details, such as location, language, age, gender, income, etc. (to enable targeted advertising)
- Financial and other secrets (bank details, credit card numbers, passwords)
- Security measures (physical—do you lock your doors?—and digital: what operating system you use, whether you use two-factor authentication, etc.)
- Brands you use (for market research)
- Personal details, such as names, dates, pets, phone numbers, etc. (for social engineering)
- Health events (for advertising, as well as insurance purposes)
- People in the house (see discussion in 5.2 above)
- Timing of in-home events (to correlate with other data; less plausibly, for burglaries)
- Political opinions (for advertising, or social control)
- Controversial behaviors (for social leverage)
- Criminal acts (confessing to a crime)
- Crying or other signs of abuse (to share with government)[1]

5.4 Attack Types

The final question we consider is how the attacker will try to collect the information we outlined above. In the case of accidental "attackers," they will collect data through the same methods, but not on purpose. We envision several methods of attack. (Note that each strategy makes certain—possibly contradictory—assumptions about how the always-listening platform operates and may therefore not be applicable to all architectures.)

Direct listening. The attacker may choose to directly listen for the personal (or other) information they are interested in, without any concealment, obfuscation, or other trickery. In doing so, their hope is that the review process does not catch their behavior.

Alternately, attackers may pursue a strategy that provides them plausible deniability and lowers the chance of detection. In general, these approaches can be thought of as *overly broad listening*.

Capturing things when the classifier has lower confidence. Generally, an NLP model will have some confidence score indicating to what extent a user utterance is likely to match a particular intent. One simple strategy is to capture audio even for lower-than-expected confidence scores, in the hope that something useful (for the attacker) is captured.

Always find named entities relevant. If the app performs its own named-entity recognition, it could always deem the entities relevant to the app's purpose (either because they legitimately do not know or maliciously).

Relevant keywords in irrelevant contexts. For example, a flight-booking app hears you talking about drinking a flight of beers. As with other scenarios, benign false positives can be expected to dominate actually malicious behavior.

Homophones (similar-sounding words). For example, a malicious app might listen for "past word" instead of "password," thus evading a filter.

Listening "around" appropriate times (past the end). A passive listening app may be privy to entire conversations, which it then

[1]Note that this type of data sharing may be considered a privacy violation even if it is in the public interest. System designers will therefore need to navigate the attendant ethical questions when deciding how to handle these scenarios.

attempts to process to fulfill its functionality. But what is the boundary of a topic or conversation? This is a hard technical challenge even for benign apps. Malicious ones may try use this as an "excuse" to keep listening even after the relevant conversation is over.

Inference / side channels. An app could attempt, for example, to infer gender by voice, income by products used, travel patterns by timing, and so forth.

6 EVALUATION

Now that we have defined attacker goals and motivations, we can start thinking about how to design protections against them. But once we have a candidate system, how should we evaluate it? In this section, we consider how we might evaluate a potential platform against our goals of ensuring user privacy and security.

Rigorous evaluation criteria are needed because different approaches to assuring privacy will necessary involve trade-offs, and we need ways of comparing them. We propose that there are two broad axes for evaluation:

(1) **Effectiveness**: how well does the system achieve its goal of increasing users' privacy?
Metrics include: functionality loss, privacy gain.
(2) **Usability**: how well would the system work for real people?
Metrics include: usability comfort, trust, acceptability, surprise, and reviewer effort (if applicable).

Effectiveness seeks to formalize the notion of how well a system achieves its goal of preserving users' privacy. Since this is the primary purpose of the system, it is a core metric for success and evaluation. Of course, there may be multiple ways of measuring effectiveness—and different components to it.

However, the system's *usability* is also crucial. The notion of usability captures the extra burden a privacy-enhancing system places on users, in terms of time, effort, cognitive load, and other expenses. Usability can be at tension with effectiveness, as a system that zealously guards a user's privacy may bother them with warnings, notifications, and needlessly blocked false positives. On the other hand, usability can be seen as contributing to the system's overall effectiveness: an unusable system will not be welcomed by consumers or adopted by companies. Usability too can be broken down into further subcomponents.

6.1 Evaluation Metric Details

The metrics that measure the effectiveness of the platform are functionality loss and privacy gain.

6.1.1 Functionality loss. This metric can be formulated as: "if app A is denied access to resource R, it will lack functionality F." Given clear assumptions about apps, it should be possible to evaluate this metric in an automated manner (see 6.2.1 below).

6.1.2 Privacy gain. While a bit more ambiguous, working with some definitions, again this metric may be evaluated automatically. However, there are a few different ways of measuring it:

Absolute privacy gain. Without the platform, any app would have access to 24 out of 24 hours every day. Suppose that the platform limits some app to listening for only one hour in a given day. (The rest of the audio is blocked from reaching the app because it is considered irrelevant.) This translates to 23 hours of "absolute" privacy gained. However, this figure does not account for *which* audio the app retained access to—it could be that the most sensitive information was not successfully shielded.

Privacy gain for sensitive conversations. Consider an app that, without the platform, only used 2 hours (out of 24) of data. The platform detected that one out of those hours was sensitive and made it off-limits, resulting in a one-hour gain.

Privacy gain based on user inputs. An app provides functionality for cars, flights, and hotels, but the user wants to only use it for flights.

For both functionality loss and privacy gain, we expect to encounter false positives and false negatives, because language is ambiguous and permissions are necessarily coarse. False positives mean the platform allows an app access to some speech, even though the app does not need and may not even want it. False negatives mean there is speech that an app needs for its (proper) functionality, but is denied access to by the platform.

6.1.3 Usability. Usability is traditionally measured by how well users are able to perform specific tasks. This includes how easy it is for users to find the information they want and choose between apps (i.e., make an informed choice about which app to install, based on the permissions it requests). Below, we survey several of the components that go into making a system usable.

Time and effort. How much time is required to make a decision and install an app? How much effort (including cognitive load) is involved in the process?

Comfort, trust, and acceptability. How willing are people to install apps with this system? For example, users may be uncomfortable with a totally opaque solution that outsourced screening to the app store.

Reviewer effort (if applicable). Certain platform architectures may rely on reviewers (professionals or community members) to examine apps for compliance with standards and adherence to rules or its own declared behaviors. In this case, an important metric for the success of the platform is the amount of time (as well as money and effort) that each review is expected to expend.

6.1.4 Surprise. If there is a mismatch between the audio an app gets and a user's expectations for what it should hear, we refer to this as "surprise," and it should be the goal of any architecture to minimize this. There are several ways in which surprise may occur.

Caused by bad or confused ML. For example, imagine that a user says, "I got into a fight," but the system hears "flight" and captures it. This would be surprising but perhaps not interesting from a system-security perspective (unless this was a deliberate attack, along the lines of those discussed above in 5.4). We therefore generally consider this problem outside the scope of our threat model, assuming perfect recognition and machine learning as much as possible, while acknowledging that reality falls far short of these assumptions.

Caused by ambiguously specified or overly broad permissions. For example, a user gives permission to an app for booking flights, but

is surprised that the system captures their conversation about birds flying. In situations like these, both the developer and platform could be at fault: the platform for having overly broad categories, or the developer for choosing too broadly. It could also be a sign that a malicious application was successfully able to evade the platform's privacy protections.

6.2 Evaluation Experiments

The metrics we have defined are only useful if we are able to evaluate and compare potential platform designs along their dimensions. Below, we propose a series of experiments that allow for these comparisons. (There may, of course, be other experimental designs.)

6.2.1 Effectiveness/privacy. Our goal is to measure effectiveness (functionality loss and privacy gain) as defined above (6.1.1 & 6.1.2). This experiment is based on evaluating simulated passive listening apps on the proposed platform design. The first step, then, is to select sample apps. While it may not be necessary to implement their functionality in full, we need to train classifiers that are *equivalent* to those of the apps, in that they should correctly classify speech (or text) that the apps will consider relevant and use for their functionality.

The next step is to generate or collect labeled speech examples. These should include both negative examples (i.e., speech not relevant to the app; this should be relatively easy to source as most things are likely to fall in that category) and positive examples (which may need to be generated specifically for the app). If the platform assumes that the speech will be transcribed before reaching the app, then the examples may be text-only.

Next, we run the app's classifier on the sample speech. Achieving high accuracy is important because our goal is to use this as a proxy for what a real skill *needs* access to for functionality; anything the classifier returns *true* on will be considered functionally necessary.

The final step is to "apply" the platform or permission system to the sample apps and rerun the classifier, now limited by the platform, on the sample speech, measuring what it still has access to. The functionality loss and privacy gains can then be inferred from these results.

6.2.2 Comfort/acceptability. Measuring users' comfort level with a proposed platform calls for a very different approach. For this task, we recommend a user study with role-playing.

In this experiment, we explain the scenario and role-playing to participants and have them choose which skills they would install for their always-listening device. In the control condition, we will make it clear that the app would get access to all data (audio recording/transcripts, 24/7). In the experimental condition, participants will see and interact with the proposed system. The dependent variable in this experiment is the number of apps installed in each of the conditions. If it is higher in the platform condition, this suggests users are more comfortable installing always-listening applications under the proposed system.

6.2.3 Clarity/surprise. The following experiment seeks to measure the level of surprise induced by the proposed privacy platform. Show people triples of {app, permission, example utterance}. That is, provide a description of the app as it would appear on the platform, as well as any information about the "permission" it requested or resources it would have access to. Sample speech, as in the effectiveness experiment, should be drawn from a corpus of positive and negative examples. For each sample app, ask whether they expect the app to receive that utterance given the permission. If they say no, the app has failed at communicating what is in scope. In other words, it is surprising, which is bad! It is important that this evaluation includes positive examples that the app will not get due to the permissions (if these exist); these are the most interesting data points. Additionally, participants can be asked to flag speech they think the app will get, but should not be allowed to.

7 ARCHITECTURE APPROACHES

At this point, we have considered passive listening apps and what they may look like, how and why miscreants may seek to exploit them, and what an evaluation might look like. But we have remained, as much as possible, agnostic to the details and mechanisms a system will use to protect users' privacy in this new paradigm of passive listening apps. Now, we begin to examine the architectural choices a privacy-protective platform may rely on.

7.1 Content-Based vs. Metadata-Based Controls

Today on mobile platforms and personal computers, privacy controls and permission systems operate primarily based on metadata, such as the source of the data, the sender, or the recipient. For example, firewalls can block access to certain ports or entire devices, ad blockers prevent certain domains from receiving data, and file permissions restrict access to users with sufficient privileges. In all of these cases, the system is agnostic to what the underlying data is; it makes its determination based on facts it knows *about* it. Even smartphone permission systems, which ostensibly distinguish between data types like contacts and location, actually operate by restricting access to specific resources and APIs, rather than inspecting the data flows. More general techniques, like access control lists, capability-based controls, and information flow control are also based on metadata.

However, when we are dealing with passively listening applications, and the data is speech, metadata-based systems are insufficient. Two conversations between the same two people may have identical metadata—yet the content will be different, and therefore the conversation could be appropriate for one app but not another. To enforce user preferences for always-listening devices, privacy controls must be able to make this distinction. They therefore have no choice but to go beyond the metadata. Thus, furnishing users with this type of choice requires a solution that draws on a different type of approach: allowing (or disallowing) access based on the content of the communication.

Applying this paradigm in practice will require either advances in speech recognition, if controls are applied on-device, or a much greater degree of trust in platforms, if controls are applied in the cloud. On the bright side, once this paradigm is sufficiently advanced, these methods will be applicable to other domains (e.g., replacing the microphone permission on smartphones with more granular controls).

Inspecting a transmission and using its contents to decide how to treat it is not unprecedented, and there may be lessons to learn from prior instantiations of this paradigm. Deep Packet Inspection

does exactly this, with a variety of Intrusion Prevention Systems relying on this technique to detect attacks. Anti-spam techniques also rely on examining the message body and, in some cases, even use natural language processing to detect unwanted messages.

The multi-level security (MLS) system used by the US military for handling classified information is another interesting example of content-based controls. Under this scheme, documents (as well as their individual sections) are assigned labels based on the sensitivity of the information they contain (confidential, secret, or top secret); an individual's clearance level determines whether they are allowed to access the document (their clearance must be at least as high as the document's classification). Since a document's classification level depends on its content, the system as a whole represents a type of content-based controls. A vast body of research, including a variety of formal methods such as the Bell-LaPadula Model, has examined how to track these labels through a system [1, 4]. However, at that stage the problem becomes another type of metadata-based control, as the system is content-agnostic once a label has been assigned. The content itself is only relevant during the actual classification process. This is a manual (and labor-intensive) operation that entails following a classification guide and checking at what level each piece of information that needs to be communicated should be classified [19]. A number of classification guides have been made public [20], but they are understandably focused on securing government secrets rather than everyday communication. Still, there may be lessons to learn from this system; for example, it may be possible to divide personal information into tiers based on its sensitivity, with apps only being granted "clearance" to certain tiers.

What are content-based controls? The basic principle of content-based privacy controls for always-listening devices is that they will examine some amount of speech, and decide whether to deny an application access to it or allow it through (possibly after certain editing or transformations). At the most general level, there are two families of approaches a system to protect privacy may take. We refer to these as **blacklisting** and **whitelisting**.

Blacklisting. This approach relies on the observation that many people do not consider most day-to-day speech sensitive, and attackers are likely to be after only certain data types (see 5.3 above). Therefore, it may be sufficient for the platform to identify speech that is truly sensitive and prevent apps from getting access to it.

Whitelisting. Taking the opposite tack, this approach envisions that apps' access is *scoped to capabilities*. In other words, apps should only get access to what they need to function. We conceptualize this as a permission system: similar to permission systems in modern smartphone operating systems, apps must declare ahead of time the resources they wish to use, and the platform enforces these restrictions. This approach achieves greater privacy benefits, since it would limit applications to hearing only the speech they need. Furthermore, as discussed in Section 5.1, the contextual integrity model suggests that this approach better aligns with user expectations. Therefore, in the remainder of this paper, we will consider how we might build such a system. (In practice, however, we expect systems to use a combination of blacklisting and whitelisting techniques. Whitelisting can guide the behavior of most apps most of the time, and blacklisting can act as a safe-guard to prevent the leakage of data considered categorically sensitive.)

7.2 What Do Permissions Look Like?

In considering the design of a permissions system for a passive listening app, we must address several related questions:

How do apps declare permissions? In smartphone permission systems, app developers specify upfront which permissions their app requires by including a list of these permissions in the app's manifest. This is a straightforward approach that works well; adopting it for always-listening apps could be one way forward. However, for this to work, a developer must be able to explicitly enumerate the resources they require. This may present challenges due to how natural language processing models currently work: by relying heavily on neural approaches and training on a wide range of examples. How can a developer translate a neural network into a list of permissions? Therefore, we must consider alternate approaches. For example, a developer could be required to submit their app—or just its natural language component—for testing and evaluation.

After solving the challenges of how apps declare their requirements, a permission system needs to address the questions of user involvement: how—if at all—is this information conveyed to the user, which choices do users have, and how much customizability are they allotted? Each of these questions has crucial implications for security and usability.

Do users get a choice? The overarching consensus in the field of usable security is that the burden of making security decisions should be removed from users as much as possible, replaced instead with security (and privacy) by default. Applied to the problem of always-listening apps, this maxim would suggest that an ideal system would ensure that all apps behave in a privacy-respectful manner. Yet, even if this were possible, there would still be an element of user choice: deciding whether to use an app or not. Any privacy-sensitive user would make this decision in part based on what the app is likely to hear, and this will be seeking information to answer that question. They could obtain it from the developer's description of the app, however this may be vague or incomplete. This is where the permission system can come in: by providing a clear and standardized way for a potential user to understand an app and its privacy impact.

How are permissions presented to users? If a permission system decides to share information about the apps with its users, the next question is: how? As pointed out above, permissions declared by apps might not be human-readable. Even if they are, there might be a gap between apps' true behavior and how concepts are interpreted by users. Therefore, care must be taken in ensuring the understandability and usability of the interfaces presented to people.

When are permission requests presented to users? A particular issue in sharing an app's permissions with users is when to present that information. One natural point is at the time of installation. However, research in smartphone permission systems showed that users rarely paid attention when presented with install-time permissions. This motivated the move to "ask-on-first-use" in mobile

operating systems. That system would present its own challenges for always-listening voice assistants, as it may be unclear when an always-listening app is first "used." Furthermore, with audio as the only output channel, the ways an app's permissions could be presented to users are further restricted.

Should users be able to deny or limit permissions? Denying means not allowing certain permissions at all (e.g., an app asks for 5 permissions, but the user only grants 4 of them). Limiting means restricting permissions within an existing hierarchy; for example, if a skill wants access to "travel" but a user wants it to know only about flights. This feature is desirable, but it may add significant complexity to the platform, both from a technical and cognitive perspective, as it would necessitate a user interface for reviewing permissions and for users to make active, informed decisions.

What is the role of other parties in the system, such as app store reviewers or super-users? What choices, capabilities, and responsibilities might we assign to them?

In the next section, we consider and compare several approaches permission designers may take. Each will vary along the dimensions above.

8 PERMISSION SYSTEM DESIGN SPACE

There are many ways to design a permission system. Rather than exhaustively listing all potential variants, here we want to sketch out the different *approaches* one might take.

In comparing and understanding the approaches, it may help to set out our *design goals*. While they may not be as quantifiable as the general evaluation metrics defined in 6.1, they clarify the principles that guide our designs. These include:

Data minimization. The goal of the whitelisting family of approaches is to select speech that is relevant to an app's functionality and appropriate to its purpose—and only that speech. The data it gets is thus minimized. By definition, then, data minimization is a core objective of a whitelisting-based permission system.

Transparency. To gain users' trust, we believe it is imperative for our system not only to act correctly, but to be transparent with users about what is happening with their speech. Therefore, it is important for users to know, which speech of theirs an app would gain access to.

Consent. Users should provide positive consent before an app is installed and thus obtains access to their speech. However, it remains an open question how that consent should be obtained, and what level of interaction and review this process should entail.

Leverage the wisdom of the crowd. We believe that, while it should always be up to an individual to decide whether an app is right for them, we can leverage other people to help in this process. For example, dedicated workers, volunteers, or other users could help determine whether an app's behavior is appropriate. This information could then be shared with prospective users or automatically used as part of the evaluation process of an app.

With these in mind, several potential approaches follow. We note that each approach may work well for certain use cases and not at all for others. This is expected, and a real-world system may employ them in combination. Our goal here is to understand their advantages and disadvantages.

8.1 Approach: Keywords

Looking for keywords in text: only allow sentences that include certain words. This is reminiscent of how today's voice assistants work, except the trigger word does not have to come first.

This approach is obviously insufficient for many purposes but may be the right choice for certain use cases, such as requiring the keywords "remind," "forget," or "remember" for a reminder app.

Challenges. Natural language has many ways of expressing an idea, so creating an exhaustive list of keywords may be difficult for any non-trivial task. (And over-eager keyword lists may create privacy problems of their own.) Furthermore, keywords may have homophones, homographs, and different meanings of the same word. In general, this approach is not applicable to many use cases, especially passive ones.

8.2 Approach: Whitelisting Topics

This involves pre-defining "buckets" that speech might fall into. For example, a conversation might have a topic, or a sentence might have an intent. The system would allow apps to subscribe to certain buckets.

Challenges. For developers to choose a topic for their app, the platform must maintain a reasonably exhaustive list of all conversation subjects human speech may plausibly contain. How can this list be generated? Is it even realistic? Even broad categories (e.g., "food") may generate privacy gains, which is good, but how should the system handle conversations—and apps—that cover multiple topics? Individual sentences can contain multiple, potentially conflicting, topics or partial matches. And many passive use cases cut across topics or are based on higher-level concepts.

8.2.1 Sub-approach: whitelist entities / slot filling. Rather than subscribing to topics, apps would declare the "types" of things they're looking for:

- A unit conversion app is looking for two unit measures plus a quantity.
- A travel app is looking for two destinations and a date.
- A mapping app is looking for a location.

8.3 Approach: Embeddings

Word embeddings are a type of NLP technique that maps words or phrases to a point in vector space. While each dimension does not necessarily have semantic meaning, similar words or phrases end up clustered together. For the permission system, apps could declare the regions from which their speech should come.

Challenges. This approach assumes that all skills would use the same intermediate representation. Recent advances in transfer learning in the NLP domain have made this more plausible, but it's still not clear how realistic this approach is. This approach also suffers from low explainability. On the other hand, it may be combined with other approaches (like "transparency", 8.6 below) to counteract this, and it is notable in enforcing stronger guarantees than some of the other approaches.

8.4 Approach: Derived Features

Have the platform compute "features" over people's speech and expose these to apps, instead of the speech itself. Examples of features may include language, speaker, sentiment, or tone. For certain use cases, these may provide sufficient information with few or no details about the underlying speech being necessary. This bears strong similarities to topic whitelisting (if topics can be considered features) and embeddings approaches.

Challenges. What are the features that the platform should make available? Furthermore, most of the problems from the related approaches are applicable here as well.

8.5 Approach: Local Mode

This follows a slightly different paradigm from the previous approaches: *listen locally and tightly control exfiltration.* The app runs on the device or in the manufacturer's sandbox with full access to all speech. However, it can only talk to the outside world through limited, pre-declared interfaces. The permission system (what users review/allow/deny) is defined over these interfaces—i.e., the outgoing data—rather than over the underlying speech. For example, a calendar app is limited so that it only sends dates over the wire. Local mode could also be used in conjunction with other permission approaches, for example by allowing users to turn it on only for certain periods of time or for certain apps.

Challenges. A full-featured calendar app (to continue with this example) will want to create events that have, at the very least, a title and, ideally, a description with additional information. All of these are open-ended text fields which, by default, are not subject to control. This creates an opportunity for major leaks, though the platform can try to enforce stronger guarantees about the contents of such fields. However, for truly malicious apps, this still leaves open the possibility of leaks through side channels (e.g., exfiltrating data through the pattern of requests).

Potential mitigation. One way to combat this sort of attack, even those that rely on side channels, is by resetting state. The concern with local mode is that you can exfiltrate arbitrary speech in open-ended fields (e.g., event title). Suppose, however, that the classifier/detector only got access to one sentence at a time. (It might be allowed to carry over state from previous sentences, but presumably be restricted in what that state is.) Then, it would be limited to exfiltrating that single sentence as the event title, but not anything else from the larger conversation context.

8.6 Approach: Transparency

In this approach, apps submit a machine learning model that is a speech detector/classifier. Its job is to classify all speech into one of two categories: whether or not it is relevant to the app. When installing an app, users see examples of conversations (drawn from a corpus maintained by the platform) and whether the "relevance detector" classified them as relevant to the app. Users can also try their own examples.

Challenges. This approach requires a static model, but developers may want to frequently retrain their models to improve performance. Asking users to constantly re-review their apps is not usable. Other concerns include edge cases, obscure examples (not covered by the platform's speech corpus), adversarial design, etc.

8.6.1 Sub-approach: crowd-sourced review. Developers supply a human-readable description of the kind of speech their classifier aims to capture. This is compared to the examples actually captured (as in the transparency approach). For example, Mechanical Turk workers may look at the examples and decide if they match the description or not. Then, users only need to review the app descriptions, rather than any speech examples. This also allows for more frequent updates to the relevance models.

8.6.2 Sub-approach: bootstrap from prior examples. Like transparency, but actively restrict apps from accessing speech that is not similar to examples that have been whitelisted or explicitly okayed by other users.

Challenges. How is similarity measured? And who determines it? Answering these questions well determines whether this is a viable approach.

8.6.3 Sub-approach: runtime opt-in. When the app wants to capture speech, request the user's consent (similar to ask on first use permission models in smartphones).

Obviously, this is not feasible for every request, but it could be used in combination with the bootstrapping approach, or when other permission approaches have low confidence.

8.7 Moving Forward

The approaches outlined above represent a (no doubt incomplete) characterization of the solution space for designing a privacy-enhancing platform for always-listening devices. These solutions need to be studied, ideally under realistic use cases, to understand when they work and where they fall down. It seems clear already that none of them will be totally sufficient on its own. Another question, therefore, is how to combine these approaches into a single system that takes advantage of their strengths rather than multiplying problems and security holes. These can be further combined with solutions that already exist or have been suggested for existing voice assistants, such as physical mute buttons, automatic deletion [10], and "sleep words" that temporarily disable a device's listening capabilities.

Furthermore, no solution is likely to be "one size fits all," because of varying privacy preferences and risk tolerances among users. Therefore, a device should be able to customize its operation to a particular user. Due to the changing nature of the technology, this process should be continuous: the device should always be learning about what the user considers acceptable and what they consider to be creepy.

Whether or not we will be able to address these research questions may have a major impact on the real world. Always-listening and always-watching devices are already widespread: smart speakers, displays, phones, and cameras. As they get "smarter" and more processing gets pushed to the cloud, the boundaries will get blurrier: when is the device listening and when is it not? Where is my data and who can access it? If, when these devices arrive on the market, the only privacy solutions are those available today—microphone permissions and physical mute buttons—they are unlikely to be

adopted, and the surveillance power of these devices will remain unchecked. On the other hand, if appropriate privacy controls can be developed, intelligent voice assistants can continue becoming more and more useful while respecting users' privacy wishes in the process. But the time to act is now. Passive-listening devices are a new paradigm, but they will be the new norm.

ACKNOWLEDGMENTS

This work was supported by the Center for Long-Term Cybersecurity (CLTC) at UC Berkeley, National Science Foundation grants CNS-1514211 and CNS-1801501; the National Security Agency's Science of Security program; and the generosity of Cisco and Mozilla.

REFERENCES

[1] Matt Bishop, Elisabeth Sullivan, and Michelle Ruppel. 2019. *Computer Security: Art and Science* (second edition ed.). Addison-Wesley, Boston.
[2] Rich Brown and Molly Price. Janary 5, 2018. Speak, Shower and Shave: Kohler Brings Smarts to Your Bathroom. *CNET News* (Janary 5, 2018).
[3] Paul Cutsinger. 2018. How to Improve Alexa Skill Discovery with Name-Free Interaction and More. https://developer.amazon.com/blogs/alexa/post/0fecdb38-97c9-48ac-953b-23814a469cfc/skill-discovery.
[4] Dorothy Elizabeth Robling Denning. 1982. *Cryptography and Data Security*. Addison-Wesley, Reading, Mass.
[5] Peter Dockrill. 2019. Newly Released Amazon Patent Shows Just How Much Creepier Alexa Can Get. *Science Alert* (May 2019).
[6] Robyn Fisher. 2017. Alexa Skills Challenge Offers $250,000 in Prizes for Best Kid Skills. https://developer.amazon.com/blogs/alexa/post/f9671946-9039-45a4-83a7-ed1e15de682d/alexa-skills-challenge-offers-250-000-in-prizes-for-best-kid-skills.
[7] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. 2018. Skill Squatting Attacks on Amazon Alexa. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 33–47.
[8] Sapna Maheshwari. 2018. Hey, Alexa, What Can You Hear? And What Will You Do With It? *The New York Times* (March 2018), A1.
[9] Nathan Malkin, Julia Bernd, Maritza Johnson, and Serge Egelman. 2018. "What Can't Data Be Used For?" Privacy Expectations about Smart TVs in the US. In *Proceedings of the 3rd European Workshop on Usable Security*.
[10] Nathan Malkin, Joe Deatrick, Allen Tong, Primal Wijesekera, Serge Egelman, and David Wagner. 2019. Privacy Attitudes of Smart Speaker Users. *Proceedings on Privacy Enhancing Technologies* 2019, 4 (2019), 250–271. https://doi.org/10.2478/popets-2019-0068
[11] Moira McGregor and John C. Tang. 2017. More to Meetings: Challenges in Using Speech-Based Technology to Support Meetings. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '17)*. ACM, New York, NY, USA, 2208–2220. https://doi.org/10.1145/2998181.2998335
[12] Christopher Mele. 2016. Bid for Access to Amazon Echo Audio in Murder Case Raises Privacy Concerns. *The New York Times* (Dec. 2016).
[13] Helen Nissenbaum. 2009. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press.
[14] Ben Popper. 2017. Google's New Clips Camera Is Invasive, Creepy, and Perfect for a Parent like Me. *The Verge* (Oct. 2017).
[15] Abhinav Rastogi, Raghav Gupta, and Dilek Hakkani-Tur. 2018. Multi-Task Learning for Joint Language Understanding and Dialogue State Tracking. In *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*. Association for Computational Linguistics, 376–384.
[16] Ruhi Sarikaya. 2018. Making Alexa More Friction-Free. https://developer.amazon.com/blogs/alexa/post/60e1f011-3236-4162-b0f6-509205d354ca/making-alexa-more-friction-free.
[17] Hamza Shaban. 2018. An Amazon Echo Recorded a Family's Conversation, Then Sent It to a Random Person in Their Contacts, Report Says. *The Washington Post* (May 2018).
[18] John M. Simpson. 2017. Home Assistant Adopter Beware: Google, Amazon Digital Assistant Patents Reveal Plans for Mass Snooping. (2017).
[19] U.S. Information Security Oversight Office. 2018. Developing and Using Security Classification Guides. https://www.archives.gov/files/isoo/training/scg-handbook.pdf.
[20] U.S. Office of the Director of National Intelligence. 2014. Office of the Director of National Intelligence Classification Guide. https://www.dni.gov/files/documents/FOIA/DF-2015-00044%20(Doc1).pdf.
[21] David Watkins. 2019. *Global Smart Speaker Vendor & OS Shipment and Installed Base Market Share by Region: Q4 2018*. Technical Report. StrategyAnalytics.
[22] Karen Weise. 2018. Hey, Alexa, Why Is Amazon Making a Microwave? *The New York Times* (2018).

Appendix: Sample Applications

Here, we provide a range of examples of always-listening apps; each is illustrated by sample interactions that may trigger them or a general description of the speech they may find relevant. We organize the apps, loosely, based on the setting where they are most likely to be invoked.

A LIVING ROOM

A.1 Q&A / search

A catch-all for queries you'd ask Google/Siri/etc. today

- Does anyone know who was president in 1837?

A.2 Add to calendar

Automatically add planned meetings and appointments to the user's calendar.

- Let's plan to get lunch at noon next Thursday.
- I took the doctor's appointment at 10 AM tomorrow. I'll pick you up at 9 and we'll go together.

A.3 Remind later

Detect when the user wants to remember something and automatically save it, in order to remind them at an appropriate time.

- I need to remember to water the plants on Friday.
- Remind me to call Morgan!
- Don't forget that Casey has soccer practice on Wednesday.
- The school needs Blake's permission slip back by Thursday.

A.4 Timer

- Start a countdown for 60 seconds
- Can you time how long it takes me to do 10 pull-ups?

A.5 Remember locations, send to map

Take note of locations mentioned during a conversation, so that they can be highlighted on the user's personal map (e.g., in a smartphone app).

- I should get stamps when I'm in the store tomorrow.
- "This cake is amazing, where did you get it?" "Virginia Bakery" "I should remember to stop by next time I'm in Berkeley"

A.6 Learn book/TV/movie/music preferences and make recommendations

- "1984 is the best book I've ever read." "I don't know, I kind of liked Brave New World more."
- "Do you listen to rap much?" "Not a lot. But it's fun to listen when my friends come over during the weekend or at a party.

A.7 Email client

- "Read me the email from Jack that I got this morning"

- "Did I get any new messages?"

A.8 Shazam

- What's the song playing in the background of this commercial?
- That song was dope! I want to look up the artist later.

A.9 Unified music history

Keep a record of any music playing, so that users can recall that information at a later point, or so that it can be used for offering personalized music recommendations.

- Do you remember the name of the track you played for me last night?

A.10 Change music based on mood in the room

While active, the app could keep track of any (non-sensitive) conversations in the room, perform sentiment analysis on them, then select the next track based on the overall mood of the room.

A.11 Read books out loud

- Hey AI, read me a book.
- Stop. Hold on.
- Can you repeat the last paragraph?
- Let's start over from the beginning of the chapter.

A.12 Trivia quiz

- I want to play a game.
- Okay, what's the largest city in Yukon?
- That's right. Want to hear another question?
- What's the smallest city in Yukon?

A.13 Language practice

When a user wants to practice their foreign language skills, the app could serve as their conversation partner, or suggest vocabulary and correct grammar mistakes as the user is talking.

A.14 Swear jar

- That's the third time you've said <expletive> today. Your swear jar is up to $3 now.

A.15 Reservation maker

The app would initiate appointments and reservations on behalf of the user.

- "Let's celebrate your birthday at Fentons next week." "Alright, we need a reservation for 6 people"

A.16 Weather

The app could directly answer queries about the weather, as well as detect when a conversation discusses plans that may be affected by the weather, and offer up the relevant forecast.

- Will it rain tomorrow?
- Do you think I need a coat?
- Is August a good time to go to Thailand?
- [discussing a trip] Are you sure you want to go hiking this weekend? The forecast calls for rain.

A.17 Trip/travel recommendations

- I really want to get away to some place quiet with a beach for Memorial Day weekend. Can you think of anywhere? I don't want to pay more than a couple hundred dollars for tickets.

A.18 Health advice

- You seem to be coughing a lot today. The air quality isn't great; there's a lot of pollen. Would you like us to order you some allergy medicine?

A.19 Fitness monitor

- We've noticed you've been sitting for a while. Do you want to stand up and walk around for a couple of minutes?

A.20 Gift recommendations

- These headphones seem great, I want to get these someday.

A.21 Track baby's vocabulary

- Cat is your baby's third unique word!

A.22 Fact checker

The app could analyze, in real time, the veracity of statements made on TV (or by the speakers on the room).

- [responding to statement on TV] Politifact has rated this claim as False.

A.23 Artificial memory: kids' names edition

The app would keep track of potentially useful information about friends and acquaintances, such as the names or birthdays of their children.

- - Do you have kids?
  - Yeah, three daughters.
  - Oh wow, I didn't know that! How old are they?
  - Well, Drizella is 30, Anastasia is 25, and Ella is 20.

A.24 Ikea helper

While you're trying to put together Ikea furniture, the app could listen, read instructions out loud, and offers help and suggestions.

A.25 Score keeper

The app could gamify household chores by keeping track of children's scores.

- 10 points for Gryffindor!

A.26 Kid monitor

Detect if a child is shouting, fighting with their sibling, or is "engaging in mischief" [8].

A.27 Motion detector

The app can detect people's presence in certain rooms of the house, then turn on the heat/AC/light there, while turning it off in other rooms.

B KITCHEN

B.1 Shopping list

- We should get eggs.
- Don't forget to get milk on your way home.
- We're almost out of butter.

B.2 Fridge monitor

Keep track of items in the refrigerator and when they are likely to expire.

- Oh yay, you got cauliflower. [Reminder that it goes bad after X days.]
- Do we need milk? No, this one is good for another week.
- What's for dinner today? Turkey and mashed potatoes. [remember that these ingredients were (probably) used up]

B.3 Recipe search

- How long do I boil eggs?
- How do I make ratatouille?
- Do we have everything we need for cookies?

B.4 Cooking advisor

- What temperature should I set this oven to for asparagus?
- How hot should the oil be for deep-frying?
- How long has this lasagna been in the oven?
- What is the temperature of a medium rare steak?
- How much sugar should I put in the cake mixer? I do not want it to be too sweet.

B.5 Unit conversion

- How many cups of flour is 200 grams?
- How many teaspoons is one tablespoon?
- What's 600 grams in ounces?

B.6 Is this thing still good?

- Oh no, I left out the ham this morning. Can I still eat it?

B.7 Coffee maker

The app could respond directly to instructions to make coffee, but also be more proactive, for example offering to brew coffee if a user simply mentions that they are tired.

C BEDROOM

C.1 Baby monitor

Inform parents if it hears their baby crying in other room

C.2 Alarm

- Wake me up tomorrow at 8.
- Time to get up!
- Do you want to snooze for another 10 minutes?

C.3 Coffee scheduler

- Have a cup ready to go when I wake up tomorrow.

C.4 Clothes recommendations

- What should I wear today? Maybe shorts? "It's 40 degrees outside; are you sure you want to wear shorts?"

C.5 Laundry reminder

- Hey, it's been a week since you've done laundry. You're almost out of socks.

D BUSINESS/MEETINGS

D.1 Action items

Summarizes action items from meeting

D.2 Equal time monitor

Makes sure everyone in the meeting has a chance to talk and doesn't monopolize the meeting

D.3 Language watchdog

Lets people know when they could have used more inclusive language

D.4 Measuring interruptibility

Determines whether now is a good time for a phone call or other interruption, based on content/tone of the conversation