World-Driven Access Control for Continuous Sensing

Franziska Roesner[1], David Molnar[2], Alexander Moshchuk[2],
Tadayoshi Kohno[1,2], Helen J. Wang[2]

[1] *University of Washington* [2] *Microsoft Research*

ABSTRACT

Modern applications increasingly rely on continuous monitoring of video, audio, or other sensor data to provide their functionality, particularly in platforms such as the Microsoft Kinect and Google Glass. Continuous sensing by untrusted applications poses significant privacy challenges for both device users and bystanders. Even honest users will struggle to manage application permissions using existing approaches.

We propose a general, extensible framework for controlling access to sensor data on multi-application continuous sensing platforms. Our approach, *world-driven access control*, allows real-world objects to explicitly specify access policies. This approach relieves the user's permission management burden while mediating access at the granularity of objects rather than full sensor streams. A trusted policy module on the platform senses policies in the world and modifies applications' "views" accordingly. For example, world-driven access control allows the system to automatically stop recording in bathrooms or remove bystanders from video frames. To convey and authenticate policies, we introduce *passports*, a new kind of certificate that includes both a policy and optionally the code for recognizing a real-world object.

We implement a prototype system and use it to study the feasibility of world-driven access control in practice. Our evaluation suggests that world-driven access control can effectively reduce the user's permission management burden in emerging continuous sensing systems. Our investigation also surfaces key challenges for future access control mechanisms for continuous sensing applications.

1. INTRODUCTION

Continuous sensing is an emerging technology that enables new classes of applications. New platforms, such as Microsoft Kinect [37], Google Glass [14], and Meta SpaceGlasses [26], fundamentally rely on continuous video and depth cameras to support natural user input via gestures and continuous audio sensing for voice commands. Applications on these platforms leverage these capabilities to deliver new functionality to users. For example, WordLens [34] is a Google Glass and iPhone application that uses the camera to continuously scan for words in the real world. It then shows translations of these words overlaid on the user's vision.

These new capabilities raise serious privacy concerns. Consider a user who enters a locker room while wearing a Google Glass. We identify four classes of privacy concerns in this scenario. First, Word Lens or other *untrusted applications* running on the Glass may see sensitive video data, both about the user and about bystanders. Second, the user may *accidentally record bystanders* by forgetting to turn off the camera while entering the locker room. Third, the user may record herself in a locker room mirror and *accidentally share* the recording on social media. Finally, *malicious users* could use the Glass to record others without their knowledge.

Figure 1: **Sensor Privacy Concerns.** *Camera restrictions are common in sensitive locations like locker rooms, where both device users' and bystanders' privacy are at risk from untrusted applications. Continuous sensing platforms like Google Glass will make it harder for honest users to mitigate such risks by managing applications' permissions. World-driven access control allows real-world objects to push policies to devices.*

The first three classes of privacy concerns have *honest* users who *want* to protect against untrusted applications and user error. While protecting against malicious users is also important, current approaches for addressing these privacy concerns do not work well *even for honest users.* In this paper we thus assume that users are honest, but that they may run untrusted applications.

Sensitive locations like locker rooms and bars commonly handle these concerns today by posting explicit policies that prohibit recording or the presence of recording devices (including Google Glass [15]), as in Figure 1. This approach, however, is hard to enforce and does little to protect a user from untrusted applications or user error. Users must notice the sign, then remember to turn off their device.

A new access control challenge. A natural way to address these privacy concerns is with application permissions in the operating system. However, continuous sensing and natural user input pose new challenges to access control design. Today, platforms like Android, iOS, and Windows 8 deny untrusted applications default access to sensitive resources like the camera and GPS. To determine which permissions to grant, these OSes put the user in the loop: with manifests at application installation time (Android, Windows 8) or prompts at the time of sensitive data access (iOS).

Previous work [12, 35], however, has shown that these permission models are flawed. Manifests are out of context with applications' use of sensitive data, making it hard for users to understand what permissions applications need and why. Prompts are disruptive and cause "prompt fatigue," conditioning users to simply click yes.

User-driven access control [35] addresses these flaws by coupling permission granting with user actions within an application (e.g., clicking a special embedded camera button).

Unfortunately, this approach is not well-suited for continuous sensing because it relies on explicit user interactions with the device. By contrast, continuous sensing applications are, almost by definition, designed to automatically "do things for you" without any such explicit actions.

Further, for applications with natural user input, like gesture or voice, the input method itself relies on the camera or microphone being always accessible. In these settings, permission granting models that allow or deny access to entire sensor streams are too coarse-grained.

The fundamental new challenge in permission system design for continuous sensing is thus enabling *fine-grained, automatic* permission granting. For example, how should the OS determine which objects in a video frame are accessible to an application? We pursued multiple unsuccessful attempts to design access control for continuous sensing, which we discuss in Section 3.1. From them, we learned that access control decisions should depend on objects and people around the user, and that these objects should specify their own access policies in a distributed, context-sensitive way.

World-driven access control. Our solution is *world-driven access control*. Using this approach, objects, places, and people would present *passports* to the operating system. A passport specifies how to handle access control decisions about an object, along with, optionally, code stating how the object should be recognized. For example, in Figure 1, the locker room might present a passport suggesting that video or audio recording is prohibited. Passports provide a *distributed, context-sensitive* approach to access control.

In our design (Section 3), a trusted *policy module* in the OS detects passports, extracts policies, and applies those policies dynamically to control applications' access to sensor data. Our design protects the user from untrusted applications while relieving the user of explicit permission management. While users can override policies communicated by passports, applications cannot.

Passports are intended to help users avoid accidentally sharing or allowing applications to access sensitive data. For example, a workplace can publish a passport stating that whiteboards are sensitive, helping the user avoid recording (and later accidentally sharing on social media) photos of confidential information on the whiteboard. In the locker room, a "no-record" policy helps the user avoid accidentally allowing an untrusted application to access the video feed of herself undressing in the mirror.

Passports can also help users respect others' wishes without requiring onerous manual configuration. At the AdaCamp conference, for example, attendees wear red lanyards to opt out of photography [3]. A world-driven access control policy can tell the policy module to remove those attendees from video streams and photos before applications see them. The user does not need to manually check lanyards or remove the device entirely. Our approach allows dynamically changing application permissions based on *context*, such as being at a conference, without explicit user actions.

Making world-driven access control work requires overcoming multiple challenges. First, there are many different *policy communication mechanisms*, ranging from QR codes and Bluetooth to object recognition, each with different tradeoffs. Second, recognizing passports and computing policy decisions induces *latency* for applications. Third, policy decisions may have *false positives and false negatives*. Finally, our approach creates a new problem of *policy authenticity* as adversaries may attempt to move, modify, or remove markers that communicate policies. We describe these and other challenges, as well as our approaches for addressing them, in the context of our implementation in Section 5 and our evaluation in Section 6.

Contributions. We introduce *world-driven access control*, whereby application access to sensor data depends on policies specified by real-world objects. Our approach allows the system to automatically manage application permissions without explicit user interaction, and supports permissions at the granularity of real-world objects rather than complete sensor streams. We contribute:

(1) *World-driven access control*, a permission model for continuous sensing that allows real-world objects to communicate policies using special *passports*. Our design enables a distributed, context-sensitive approach for objects to control how sensitive data about them is accessed and for the system to validate the authenticity of these policies.
(2) An extensible system design and implementation in which a trusted *policy module* protects users from untrusted applications without requiring the user to explicitly manage permissions. We evaluate our prototype in five different settings (Section 6) with representative policies from prior work and real-world policies. Our design's modularity allows us to implement each policy in under 150 lines of C# code.
(3) Empowering objects with the ability to influence access control decisions on users' devices introduces numerous challenges. We crystallize these challenges and explore methods for addressing them. For example, we introduce techniques for mitigating latency and accuracy challenges in detecting and enforcing policies (Sections 3-6), such as by combining multiple means of communicating a policy.

In summary, world-driven access control is intended to to relieve the user's permission management burden while preserving functionality and protecting privacy in emerging continuous sensing applications. This work presents a new design point in the space of access control solutions for continuous sensing applications. We believe that the foundations laid herein will facilitate further work in the field.

Previous work in this area focused on bystander privacy (e.g., visual or electronic opt-outs [7, 16, 36]), or is specific to one type of object alone. We discuss related work in detail in Section 8. Finally, while our focus is on continuous sensing, our policies can also apply to discrete sensing applications, e.g., a cell phone camera taking a photo, as well as to output permissions, e.g., permission for a camera to flash or a phone to ring. We reflect on challenges and discuss other extensions in Section 7, then conclude in Section 9.

2. GOALS AND THREAT MODEL

We consider fine-grained access control for sensor data on platforms where multiple isolated applications desire continuous access to system sensors, such as camera, microphone, and GPS. In our model, the system is trustworthy and uncompromised, but applications are untrusted.

Goals. Our goal is to *help honest users manage applications' permissions*. A user may do so to (1) *protect his/her own privacy* by minimizing exposure of sensor information to untrusted applications, and/or (2) *respect bystanders' privacy wishes*. We seek to help the user achieve these goals with minimal burden; users should not need to continuously manage permissions for each long-running application.

Constraints. We constrain our model in three ways:
(1) We consider only access control policies that are applied at data collection time, not at data distribution time. These policies affect whether or not an application may receive a data item — such as a camera frame or audio event — but do not provide additional data flow guarantees after an application has already accessed the data.
(2) Policies apply to applications, not to the system itself. For example, if a policy restricts camera access, the (trusted) system still receives camera data but prevents it from reaching applications. We observe that any system designed to apply policies embedded in real-world sensor data must, by definition, be able to receive and process sensor inputs.
(3) Users can always use another device that does not run our system, and hence can, if they desire, violate real-world policies with non-compliant devices. Our solution therefore does *not* force user or device compliance and, indeed, explicitly allows users to override access control policies. We consider techniques for verifying device compliance, which have been studied elsewhere [24], out of scope.

Novel threat model. Our approach empowers real-world objects with the ability to broadcast data collection policies. Unlike conventional approaches to access control, like manifests and prompts, we therefore transfer the primary policy controls away from the user and onto objects in the real world. Thus, in addition to untrusted applications, we must consider the threat of *adversarially-influenced real-world objects*. For example, we must consider malicious policies designed to prevent recording (e.g., criminals might appreciate the ability to turn off all nearby cameras) or override policies that prohibit recording (e.g., someone seeking to embarrass users in the bathroom by causing them to accidentally record). These threats guide several design decisions, which we will return to later, like the ability for users to override policy suggestions and our design of passports.

3. WORLD-DRIVEN ACCESS CONTROL

World-driven access control is based on three principles derived from our initial attempts to build an access control mechanism for continuous sensing, which we summarize. We then describe our solution, key challenges we encountered, and how our design addresses these challenges. Figure 2 shows world-driven access control in action, and Figure 3 shows a block diagram of our system.

3.1 Principles from Initial Approaches

Object-driven policies. In user-driven access control [35], applications are granted access to sensitive resources as a result of explicit in-application user interactions with special UI elements (e.g., clicking on an embedded camera button). This technique effectively manages application permissions without the drawbacks of prompts or install-time manifests, and so we attempted to adapt it to continuous sensing.

In some cases, user-driven access control worked well — photos and video chat on Google Glass, for example, are triggered by explicit user actions like winking or using the touchpad. However, many promising continuous sensing applications do not involve explicit user input. For example, the WordLens translation application is *object-driven*, rather than user-driven: it reacts to all words it "sees" and translates them without explicit user input. In future systems, such applications that "do things for you" may run continuously for long periods of time; they are uniquely enabled by continuous sensing and raise the greatest privacy risks. Attempting to inject user actions into such applications — such as allowing users to use a drag-and-drop gesture to grant applications one-time access to information about real-world objects — felt disruptive and artificial.

Figure 2: **World-Driven Access Control in Action.** *World-driven access control helps a user manage application permission to both protect her own privacy and to honor a bystander's privacy wishes. Here, a person asks not to be recorded; our prototype detects and applies the policy, allowing applications to see only the modified image. In this case, the person wears a QR code to communicate her policy, and a depth camera is used to find the person, but a variety of other methods can be used.*

From this, we learned that access control decisions cannot depend (only) on user actions but should instead be object-driven: real-world objects around the user must help determine access control policies.

Distributed organization. The next problem was how to map from objects to access control policies. We first attempted to build a taxonomy of which objects are sensitive (e.g., whiteboards) or not sensitive (e.g., chairs). Unfortunately, a static taxonomy is not rich enough to capture the complexities of real-world objects — for example, not all whiteboards contain sensitive content.

We then attempted to define a hierarchical naming scheme for real-world objects. For example, we might name an object by referring to the room it is in, then the building, and finally the building's owner. We hoped to build a system analogous to the Web's Domain Name System, where objects could be recognized in the real world, mapped to names, and finally mapped to policies. Unfortunately, because objects can move from one location to another, policies may not nest hierarchically, and hierarchies not based on location quickly become complex.

Though there may be value in further attempting to taxonomize the world and its moving objects, we chose to pursue an approach in which policy decisions are not centralized. Instead, our design's access control decisions are *distributed*: each object is able to set its own policy and communicate it to devices.

Context sensitivity. Finally, we faced the challenge of specifying an object's policy. A first approach here is to have static "all-or-nothing" policies. For example, a person or a whiteboard could have a policy saying to never include it in a picture, and certain sensitive words might always mark a conversation as not for recording.

While static policies cover many scenarios, others are more complex. For example, AdaCamp attendees can opt out of photos by wearing a red lanyard [3]. At the conference, the lanyard communicates a policy, but outside it does not. Policies may also depend on applications (e.g., only company-approved apps may record a whiteboard). From this, we learned that policies must be *context-sensitive* and arbitrarily complex. Thus, in addition to static policies, we support

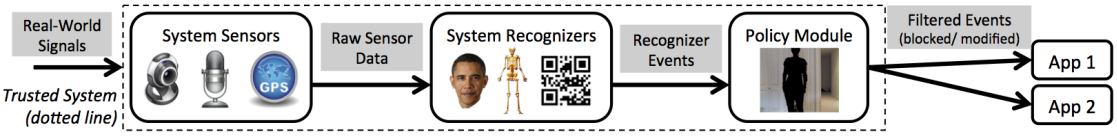


Figure 3: **System Overview.** *We build on the recognizer abstraction [18], in which applications subscribe to high-level events generated by object recognition algorithms. Our contribution is the policy module (Section 3.4), which sees all events, detects policies, and blocks or modifies events accordingly before releasing them to untrusted applications.*

policies that are (appropriately sandboxed) executable code objects, allowing them to evaluate all necessary context.

3.2 Background: Recognizers & Events

A key enabler for world-driven access control is the *recognizer* operating system abstraction [18]. A recognizer is an OS component that processes a sensor stream, such as video or audio, to "recognize" objects or other triggers. Upon recognition, the recognizer creates an *event*, which is dispatched by the OS to all registered and authorized applications. These events expose higher-level objects to applications, such as a recognized skeleton, face, or QR code. Applications may also register for lower-level events, like RGB/depth frames or location updates.

The recognizer abstraction restricts applications' access to raw sensor data, limiting the sensitive information an application receives with each event. Further, this event-driven model allows application developers to write code agnostic to the application's current permissions [18]. For example, if an application does not receive camera events, it cannot distinguish whether the hardware camera is turned off or whether it is currently not authorized to receive camera events.

Previous work on recognizers, however, did not study how to determine which permissions applications should have [18]. While the authors discuss visualization techniques to inform users what information is shared, the burden is on users to manage permissions through prompts or manifests. World-driven access control removes this burden by dynamically adjusting permissions based on world-expressed policies.

3.3 Policies and Policy Communication

We now dive more deeply into our design. We seek a general, extensible framework for communicating policies from real-world objects. Our prototype focuses primarily on policies *communicated explicitly* by the environment (e.g., by QR code or ultrasound), not inferred by the system. However, our system can be extended to support implicit or inferred policies as computer vision and other techniques improve. For example, the recent PlaceAvoider approach [39] — which can recognize privacy-sensitive locations with training — can be integrated into our system.

Designed to be a broad framework, world-driven access control can support a wide range of policy communication mechanisms on a single platform, including QR codes or other visual markers, ultrasound, audio (e.g., embedded in music), location, Bluetooth, or other wireless technologies. Additionally, world-driven access control supports both policies that completely block events (e.g., block RGB frames in the bathroom) and that selectively modify events (e.g., remove bystanders from RGB frames). Appendix A summarizes existing real-world policies (e.g., recording bans) that can be supported by this approach and which motivate the incentives of establishments to deploy world-driven policies.

A world-driven policy specifies (1) when and for how long it should be active, (2) to which real-world objects or locations it applies, (3) how the system should enforce it, and (4) to which applications it applies, if applicable. Elaborating on the latter, policies can be application-specific, such as by whitelisting known trusted applications. For example, a corporate building's policy might block audio input for any application not signed with the company's private key.

While conceptually simple, there are numerous challenges for communicating policies in practice. In our design, we aim to quickly recognize a policy, quickly recognize the specific objects or locations to which it applies, and accurately enforce it. A challenge in achieving these goals is that policy communication technologies differ in range, accuracy, information density, and the ability to locate specific real-world objects. For example, Bluetooth or WiFi can be detected accurately across a wide area but cannot easily refer to specific objects (without relying on computer vision or other object detection techniques). By contrast, QR codes can help the system locate objects relative to their own location (e.g., the person to whom the QR code is attached), but their range and detection accuracy is limited (Section 6). Similarly, a QR code can communicate thousands of bytes of information, while a specific color (e.g., a colored shirt or lanyard [3, 36]) can communicate much less.

We begin to tackle these challenges with several approaches, and return to additional challenges (like recognition latency and policy authenticity) in later sections:

Extending policy range. The range of a technology should not limit the range of a policy. We thus allow a policy to specify whether it is active (1) while the policy signal itself is in range, (2) for some specified time or location, or (3) until an explicit "end policy" signal is sensed. For example, a WiFi-based policy for a building might use the first option, and a QR-code-based policy for a bathroom might use the third option (with "start" and "end" policy QR codes on either side of the bathroom door). There are challenges even with these approaches; e.g., the "end policy" signal may be missed. To overcome missed signals, the QR-code-based policy can fall back to a timeout (perhaps after first notifying the user) to avoid indefinitely applying the policy.

Increasing information density. Policies need not be specified entirely in a real-world signal. Instead, that signal can contain a pointer (such as a URL) to a more detailed remote policy. Based in part on this idea, we introduce passports as a method for dynamically extending the system with new policy code in Section 4.

Combining technologies. Observing that different communications methods have different tradeoffs, we propose the use of *hybrid techniques*: allowing multiple mechanisms to work together to express a policy. For example, to remove bystanders wearing a specific color from RGB frames, a wide-range high-density technology like Bluetooth can bootstrap the policy by informing the system of the opt-out color; the color helps locate the area to remove from a frame.

3.4 Policy Detection and Enforcement

There is considerable diversity in the types of policies objects may wish to convey, and hence our solution must be able to accommodate such diversity. As surfaced in this and later sections, not all policies are trivial to handle.

World-driven policies are enforced by a trusted policy module on the platform (Figure 3). The policy module subscribes to all (or many) of the system recognizers. Any of these recognizers may produce events that carry policies (e.g., detected QR codes, WiFi access points, or Bluetooth signals). The policy module filters events before they are delivered to applications. The result of event filtering may be to block an event or to selectively modify it (e.g., remove a person from an RGB frame).

In more detail, the policy module keeps the following state: (1) default permissions for each application (e.g., from a manifest or other user-set defaults), specifying whether it may receive events from each recognizer, (2) currently active policies, and (3) a buffer of events from each recognizer.

Before dispatching a recognizer event to an application, the policy module consults the default permission map and all active policies. Depending on the result, the policy module may block or modify the event.

Event modification is complicated by the fact that it may rely on information in multiple events. For example, if someone wearing a QR code should be removed from an RGB frame, the modification relies on (1) the person event, to isolate the pixels corresponding to a person, and (2) the QR code event, to pinpoint the person nearest the QR code's bounding box (to whom the policy applies).

The policy module thus uses the buffer of recognizer events to identify those needed to enforce a policy. Because corresponding events (those with the same frame number or similar timestamps) may arrive at different times from different recognizers, the policy module faces a tradeoff between policy accuracy and event dispatch performance. For example, consider a QR code that begins an RGB blocking policy. If the QR code recognizer is slow and the policy module does not wait for it, an RGB frame containing the QR code may be mistakenly dispatched before the policy is detected.

Thus, for maximum policy accuracy, the policy module should wait to dispatch events until all other events on which a policy may depend have arrived. However, waiting too long to dispatch an event may be unacceptable, as in an augmented reality system providing real-time feedback in a heads-up display. We discuss our implementation choice, which favors performance, in Section 5.

3.5 Policy Interface

Each world-expressed policy has a fixed interface (Figure 4), which allows policy writers to extend the system's policy module without dictating implementation. We expect that policies are typically short (those we describe in Sections 5 and 6 all require under 150 lines of C#) and rely on events from existing object recognizers, so policy writers need not implement new recognizers.

The key method is `ApplyPolicyToEvent`, which is called once for each event dispatched to each application. This method takes as input the new event and a buffer of previous events (up to some bound), as well as metadata such as the application in question and the system's configuration (e.g., current permissions). The policy implementation uses this information to decide whether and how to modify or block the event for this application. The resulting event is returned, with a flag indicating whether it was modified.

```
public interface IPolicy {
 string PolicyName();
 void Init();
 void ApplyPolicyToEvent(
  RecognizerEvent inEv, EventBuffer prevEvents,
  out RecognizerEvent outEv, out bool modified);
 void Destroy();          }
```

Figure 4: **Policy Interface.** *Policies implement this interface to block or modify events before they are passed to applications.*

The resulting event is then passed to the next policy, which may further modify or block it. In this way, policies can be composed. Although later modifications are layered atop earlier ones, each policy also receives *unmodified* events in the event buffer used to make and enforce policy decisions. Thus, poorly written or malicious policies cannot confuse the decisions of other policies.

Policy code isolation. Policy code can *only* use this restricted API to obtain and modify sensor events. Policies must be written in managed code and may not invoke native code, call OS methods, or access the file system, network, or GPU. (Isolating managed code in .NET is common with AppDomains [27].) Our restrictions protect the user and the OS from malicious policy code, but they do impose a performance penalty and limit policies to making local decisions. Our evaluation shows that we can still express many realistic policies efficiently.

Policy conflicts. In the case of policy conflicts, the system must determine an effective global policy. If multiple world-driven policies conflict, the system takes the most restrictive intersection of these policies. In the case of a conflict due to a user's policy or explicit action, recall that we cannot force users or their devices to comply with world-driven policies, since users can always use hardware not running our system. The system can thus let the user override the policy (e.g., by incorporating access control gadgets [35] to verify that the action genuinely came from the user, not from an application) and/or use a confirmation dialog to ensure that the violation of the world-driven policy is intentional, not accidental, on the user's part. This approach also gives users the ability to override malicious or questionable policies (e.g., policies that block too much or too little).

4. PASSPORTS: POLICY AUTHENTICITY AND RUNTIME EXTENSIONS

In this section, we simultaneously address two issues: policy authenticity and system extensibility. First, an attacker may attempt to forge, move, or remove an explicit world-driven policy, requiring a method to verify policy authenticity. Second, new policy communication technologies or computer vision approaches may become available, and our system should be easily extensible.

To address both issues, we introduce a new kind of digital certificate called a *passport*. Inspired by real-world travel passports, our policy passport is designed to support policy authenticity by verifying that a passport (1) is issued by a trusted entity, and (2) is intended to apply to the object or location where it is observed. To support extensibility, a passport may additionally contain sandboxed object recognition and/or policy code to dynamically extend the system.

4.1 On the Need for Policy Authenticity

We elaborate here on the need to address attackers who might wish to make unauthorized modifications to existing policies. Attacker modifications may make policies less restrictive (e.g., to trick employees of a company into accidentally taking and distributing photos of confidential information in conference rooms) or more restrictive (e.g., to prevent the cameras of surrounding users from recording the attacker breaking a law). In particular, attackers attempting to change policies may employ the following methods:

1. Creating a forged policy.
2. Moving a legitimate policy from one object to another (effectively forging a policy on the second object).
3. Removing an existing policy.

The difficulty of mounting such attacks depends on the policy communication technology. For example, it may be easy for an attacker to replace, move, or remove a QR code, but more difficult to do so for an ultrasound or WiFi signal. Thus, these risks should be taken into account by object or location owners deploying explicit policies.

The threat of policy inauthenticity also varies by object. For example, it is likely more difficult for an attacker to forge a policy (like a QR code) affixed to a person than one affixed to a wall. Thus, for certain types of objects, such as humans, the system may trust policies (e.g., colored lanyards as opt-outs [3]) without requiring additional verification, under the assumption that it is difficult to physically manipulate them.

4.2 Signing Passports

The contents and origin of passports must be cryptographically verifiable. There are numerous approaches for such verifiability, ranging from crowd-sourced approaches like Perspectives [43] and Convergence [25] to social-based approaches like PGP [44]. While the certificate authority (CA) model for the Web has known limitations [9], because of its (current) wide acceptance, we choose to build on the CA model for world-driven access control. However, we stress that the underlying mechanism is interchangeable.

Building on the CA model, we introduce a policy authority (PA), which is trusted to (1) verify that the entity requesting a passport for an object or location is the real owner (i.e., has the authority to provide the policy), and (2) sign and issue the requested passport. Thus, object or location owners wishing to post policies must submit them to be signed by one of these trusted authorities. For this approach to be effective, it must be difficult for an attacker to obtain a legitimately signed policy for an object or location for which he or she cannot demonstrate ownership.

4.3 Describing Objects via Passports

In addition to containing a policy specification, each signed passport contains a description of the associated object or location. For a fixed location (e.g., on a corporate campus), the description may specify GPS coordinates bounding the targeted area. For a mobile object, the description should be as uniquely-identifying as possible (e.g., containing the license plate of the associated car). Thus, after verifying a passport's signature, the system also verifies that the enclosed description matches the associated object or location.

Having a description in the passport helps prevent an attacker from moving a legitimately signed passport from one object or location to another. Preventing such an attack may be difficult to achieve in general, as it depends on the false positive rates of the object recognition algorithm and what is actually captured by the device's sensors (e.g., the car's license plate may not be in view). The key point is that passports give object owners control over the description, so an object owner can pick the best available description.

We observe that object descriptions may create privacy risks for objects if not designed carefully. We return to the tension between description clarity and privacy in Section 7.

We further support *active object descriptions* contained in the passport. Rather than being simply static (e.g., "car with license plate 123456"), the object description may consist of code (or of a pointer to code) that directly extends the system's recognizers (e.g., with a car recognizer). Beyond aiding policy verification, active descriptions can extend the system's basic object recognition abilities without requiring changes to the system itself. Certificates including code and policy are similar in spirit to the permission objects in .NET Code Access Security [28], the delegation authorization code in Active Certificates [6], the self-describing packets in Active Networks [40], or objects in Active DHTs [13], but we are not aware of previous approaches that dynamically add object recognition algorithms. In our initial design, the active object descriptions are trusted by virtue of the PA's signature, and could be further sandboxed to limit their capabilities (as in Active DHTs [13]).

If the description contained in the passport does not match the associated object, the passport is ignored — i.e., the system applies a sensible default. For a real-world travel passport, the default is to deny access; in our case, requiring explicit allow-access policies on all objects would likely require an unacceptable infrastructure and adoption overhead. Thus, in practice, the system's default may depend on the current context (e.g., a public or private setting), the type of object, or the type of recognizer. For example, a default may deny face recognition but allow QR code events.

We believe the idea of active object descriptions is of interest independent from world-driven policies. By allowing real-world objects to specify their own descriptions, the system's object recognition capabilities can be easily extended in a distributed, bottom-up manner.

4.4 Monitoring Passports

Passport signatures and objects descriptions help prevent attackers from forging or moving passports. We must also consider the threat of removing passports entirely. Again, because a deny-access default may create high adoption overhead, a missing passport allows access in the common case.

As an initial approach, we suggest mitigating this threat by monitoring passports. In particular, a policy owner can monitor a passport directly — by installing a device to actively monitor the policy, e.g., to ensure that the expected QR code or ultrasound signal is still present — or can include in the policy itself a request that devices occasionally report back policy observations. While an attacker may also try to manipulate the monitoring device, its presence raises the bar for an attack: disabling the monitor will alert the policy owner, and fooling it while simultaneously removing the policy from the environment may be difficult for some policy communication technologies (e.g., WiFi).

Stepping back, we have explored the problem space for policy passports. The proposed designs are an initial approach, and we welcome future work on alternative approaches supporting our core concept that *real-world objects can specify*

their own policies and prove their authenticity.

5. IMPLEMENTATION

While world-driven access control may intuitively appear straightforward, it is not obvious that it can actually work in practice. We explore the challenges — and methods for overcoming them — in more detail with our implementation and evaluation. Our prototype runs on a Windows laptop or tablet and relies on sensors including the Kinect's RGB/depth cameras and its microphone, the built-in WiFi adapter, and a Bluetooth low energy (BLE) sensor. Though this platform is more powerful than some early-generation continuous sensing devices, we expect these devices to improve, such as with specialized computer vision hardware [10, 30]. Some continuous sensing platforms — such as Xbox with Kinect — are already powerful enough today.

Sensor input is processed by recognizers [18], many of which simply wrap the raw sensor data. We also implemented a QR code recognizer using an existing barcode library [1], a speech keyword recognizer using the Microsoft Speech Platform, a WiFi access point recognizer, and one that computes the dominant audio frequency.

5.1 Policy Module

As described in Section 3.4, the policy module subscribes to events from all of the system's recognizers and uses them to detect and enforce policies. Before dispatching an event to an application, the system calls the policy module's `FilterEvent()` method, which blocks or modifies the event based on currently active policies. To modify an event, it uses recently buffered events from other recognizers if necessary.

Recall from Section 3.4 the policy accuracy versus performance tradeoff: policies can be detected and applied more accurately if all relevant events have arrived, but waiting too long impacts event dispatch latency. In our implementation we favor performance: to avoid delays in event dispatch, we do not match up events precisely using frame numbers or timestamps, but simply use the most recent events, at the possible expense of policy accuracy. Alternatively, this choice could be made per application or by user preference. We quantify the effects of this tradeoff in Section 6.

5.2 Passports

In addition to static policies (e.g., a simple QR code that activates a pre-installed "block RGB events" policy), our prototype supports passports by transforming certain recognizer events into passport lookups. The system appends the text in a QR code or a Bluetooth MAC address to the base URL at `www.wdac-passport-authority.com/passportlookup/`. If a corresponding passport exists, the server provides its URL. The passport contains a policy DLL that implements our policy interface (Figure 4). The system downloads, verifies, and installs the new policy (if not previously installed).

The process of loading a passport could be optimized by caching server responses. If the system cannot make a network connection, it misses the passport; a future implementation could buffer the network request until connectivity is available, in case the policy is still applicable then. Note that the system does not need a network connection to parse static (non-passport) policies; passports communicated via certain communication technologies (e.g., Bluetooth) could also encode their policy code directly rather than pointing to a server. As noted in Section 4, code in a passport could be signed by a policy authority and sandboxed. Since signature and sandboxing approaches have been well studied elsewhere, we do not include them in our prototype but rather focus on world-driven access control functionality.

5.3 Prototype Policies

Block RGB in sensitive area. We implemented several policies blocking RGB events in sensitive areas, based on QR codes, BLE, WiFi, and audio. For example, a QR code on a bathroom door can specify a policy that blocks RGB until the corresponding "end policy" QR code is detected. We can similarly use a BLE emitter to specify such a policy; BLE has a range of roughly 50 meters, making it suitable for detecting a sensitive area like a bathroom. Similarly, we use the access point (AP) name of our office's WiFi network to trigger a policy that blocks RGB events. (Future APs might broadcast more complete, authenticated policies.)

We also use our audio frequency recognizer to block RGB events: if a trigger frequency is heard three 32 ms timeslots in a row, the corresponding policy is engaged (and disengaged when the trigger has not been heard for three consecutive timeslots). In our experiments, we used a frequency of 900 Hz; in a real setting, ultrasound may be preferable. Ultrasound is already used in real settings to communicate with smartphones [42]. More complex audio communication is also possible, e.g., encoding a policy in the signal [33].

Remove person from RGB. We support several policies to remove bystanders from RGB frames. First, we support bystanders wearing opt-out QR codes — though our implementation could easily support an opt-in instead — and filter RGB events to remove the pixels associated with the person nearest such a QR code's bounding box. To do so, we use the per-pixel `playerIndex` feature of Kinect depth frames. Figure 2 shows a modified RGB frame based on this policy.

As a simpler, more efficient policy to remove people from RGB frames, and inspired by practices at AdaCamp, we also implemented a color-based policy, using a certain shirt color to indicate an opt-out. This policy (1) detects a 10x10 pixel square of the target color, (2) calculates its average depth, and (3) removes RGB pixels near that depth. Thus, this policy applies to anything displaying the target color, and it avoids the latency of the Kinect's player detection. Using a hybrid approach, we also implemented a BLE policy to bootstrap the color policy (discussed more in Section 6).

Block sensitive audio. We implemented a policy that blocks sensitive audio based on our speech keyword recognizer. We use a hard-coded grammar that includes words that may signal a sensitive conversation, such as project codenames like "Natal," as well as explicit user commands such as "stop audio." The policy blocks audio events once such a keyword is detected, and resumes audio events upon command (i.e., when the user explicitly says "begin audio").

6. EVALUATION

We evaluate along two axes:

(1) Policy Expressiveness and Implementation Effort. We show that our architecture can incorporate a wide variety of policies desired by users and proposed in prior work with low developer effort (Section 6.1).

(2) Policy Effectiveness. We explore the effectiveness of policy communication technologies and world-driven policies implemented in our prototype (Section 6.2).

Name	*Detection*	*Enforcement*	*Lines of Code (C#)*
Respectful Cameras [36]	Visual indicator	Blur faces	84 (QR code) – 104 (color)
Brassil [7]	Bluetooth	Blur faces	69 (remove person)
TagMeNot [8]	QR Code	Blur faces	84 (remove person)
Iceberg Systems [20]	Ultrasound	Disable camera	35 (audible sound)
Parachuri et al. [31]	Visual indicator	Remove person	84

Figure 5: **Policies in Prior Work.** *Our architecture generalizes policies from prior work; this table details those we implemented.*

6.1 Policy Expressiveness & Effort

We validate that our policy interface (Figure 4) is expressive enough to capture realistic policies. We implemented a representatives set of policies proposed in prior work relevant to continuous sensing [7, 8, 20, 31, 36], summarized in Figure 5. In some cases, we implemented a slightly different enforcement than the one specified in prior work (e.g., removing people from the frame instead of blurring faces), and we note these differences where applicable. Each policy could be implemented in our prototype with only a modest number of lines of C# code (between 35 and 104, measured using Visual Studio's Code Metrics), indicating that our architecture supports new and diverse policies with modest additional developer effort.

Other policies exist in prior work, such as a policy to encrypt recordings with a shared key derived with a distributed prototol [16]. We determined that our system could support this policy, but for our purposes here we chose not to spend effort reimplementing its protocol. Our prototype can also handle existing real-world policies (Appendix A).

We conclude that our architecture's policy abstraction is sufficiently expressive and that policies can be implemented with modest developer effort.

6.2 Policy Effectiveness

We evaluate policy effectiveness in representative scenarios. We aim not to maximize effectiveness but to explore policy communication technologies and demonstrate the feasibility of world-driven access control. We also view our evaluation methodology itself as a contribution for others studying access control for continuous sensing.

6.2.1 Evaluation Methodology

We designed and evaluated representative scenarios, each with one or more environmental policy triggers (Figure 7). For example, in the Bathroom scenario, we affixed QR codes on and near the bathroom door, and we placed a BLE emitter and a smartphone producing a 900 Hz tone inside. In the Person scenario, a person carried a BLE emitter and wore a QR code and a red color.

We used our prototype to *record* a trace of raw recognizer events for each scenario. We constructed scenarios so that a policy was applicable in one continuous segment (e.g., a person is only seen once in the Person trace) and that the trace was realistic (e.g., we turned on a faucet in the bathroom). We then *replayed* every trace once for each policy present. Pre-recorded traces let us compare multiple policies on the same trace. Since our goal is to evaluate feasibility, we consider only one representative trace of each scenario.

We then *annotated* each trace with ground truth for each policy of interest: whether the ideal enforcement of that policy would modify or block each event. We compared the results of the replay for each policy with our ground truth annotations. In particular, we introduce the following methodology for evaluating the effectiveness of a policy. Considering only those events that *may* be affected by the given policy, such as RGB in the Bathroom trace or audio in the Speech trace, we evaluate the policy as follows.

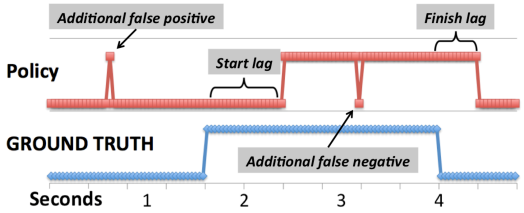


Figure 6: **Example Policy Evaluation.** *We evaluate start lag (false negatives before a policy is correctly applied), finish lag (false positives after a policy should have been stopped), and additional false positives and negatives outside of these lags.*

Specifically, we consider four values: (1) *start lag* (false negatives before the policy is correctly applied), (2) *finish lag* (false positives before the policy is correctly removed), (3) *additional false positives*, and (4) *additional false negatives*. Conceptually, start and finish lag measure the time it takes to recognize the policy. Additional false positives occur outside of this region and additional false negatives occur within this region; neither include the start or finish lag. Figure 6 shows each value visually. These values rely, of course, on specific characteristics of a trace; we attempted to construct realistic evaluation scenarios to get a conceptual idea of how well these policies work in practice.

We ran our traces with an outward-facing Kinect attached to a Lenovo W520 laptop (Core i7, 16 GB RAM, 160 GB SSD). This setup let us log traces without affecting events rates, relying on a large memory to buffer all events until trace completion. Without logging, we achieved similar event rates on a less powerful Samsung tablet (Core i5, 2 GB RAM). We expect that continuous sensing devices will approach the capabilities of these more powerful machines.

6.2.2 Evaluation Results

We describe lessons learned from our experience, supported by policy-specific evaluations (Figures 8–9).

Adding policy memory reduces jitter. We find that certain policy technologies show high variability in detection. For example, QR codes are detected only about every third RGB frame when present, WiFi signals are unreliable at a building's perimeter (presumably far from an access point), and audio frequency is sensitive to other ambient audio (e.g., running water in the bathroom).

We find that we can minimize this detection jitter by adding *memory* to the policy. For example, in the original QR code policy, we consult the most recent QR code event for each RGB event to determine if the policy should be applied. In the "QR code with memory" policy, we look farther into the past, consulting up to five recent QR code events and applying the most recent policy it finds (if any).

Trace Name	*Policy Trigger*	*Intended Policy for **Targeted Event Type***	*LOC*	*Total Events*	*Length*
Bathroom	QR Code	Block **RGB events** in bathroom.	24	10435	83 sec.
Bathroom	Bluetooth	Block **RGB events** in bathroom.	23	10435	83 sec.
Bathroom	Audio Frequency (900 Hz)	Block **RGB events** in bathroom.	35	10435	83 sec.
Person	QR Code	Remove person from **RGB events**.	84	7132	60 sec.
Person	QR Code with memory	Remove person from **RGB events**.	89	7132	60 sec.
Person	Bluetooth + Person	Remove person from **RGB events**.	69	7132	60 sec.
Person	Color	Remove person from **RGB events**.	104	7132	60 sec.
Speaking	Speech Keyword	Block **audio events** after sensitive keyword.	23	4529	42 sec.
Corporate	WiFi	Block **RGB events** in corporate building.	23	21064	191 sec.
Commute	Location	Blur **location events** in sensitive area.	29	475	482 sec.

Figure 7: **Evaluated Policies.** *We constructed scenarios, took traces using our prototype, and evaluated the effectiveness of various policies. In the third column, the type of event targeted by the policy is bolded; the fourth column reports total events in the trace.*

Trace Name	*Policy Trigger*	*# Target Events*	*Start Lag*	*Finish Lag*	*Additional FN*	*Additional FP*
Bathroom	QR Code	1946	0 (0 sec)	0 (0 sec)	0	0
Bathroom	Bluetooth	1946	-50 (-2.1 sec)	183 (7.8 sec)	0	0
Bathroom	Audio Frequency (900 Hz)	1946	0 (0 sec)	0 (0 sec)	81	0
Person	QR Code	1244	279 (13.5 sec)	0 (0 sec)	88	0
Person	QR Code with memory	1244	279 (13.5 sec)	0 (0 sec)	20	0
Person	Bluetooth + Person	1244	210 (10.1 sec)	8 (0.4 sec)	0	30
Person	Color	1244	147 (6.1 sec)	0 (0 sec)	23	23
Speaking	Speech Keyword	375	16 (1.8 sec)	27 (3.0 sec)	0	0
Corporate	WiFi	2823	21 (1.4 sec)	0 (0 sec)	171	417
Commute	Location	475	14 (14.2 sec)	4 (4.1 sec)	0	0

Figure 8: **Evaluation Results.** *This table shows the results from running each policy against the corresponding trace. The first two columns match those in Figure 7. Figure 6 explains how start/finish lag and additional false positives/negatives are calculated.*

(One QR code event, or `null` if no QR code is found, is generated per RGB frame.) This change makes the QR code policy more robust: if at least one of five RGB frames results in a correctly detected QR code, the policy will not jitter in the interim. The tradeoff is that too much memory may lead to a longer finish lag or more false positives.

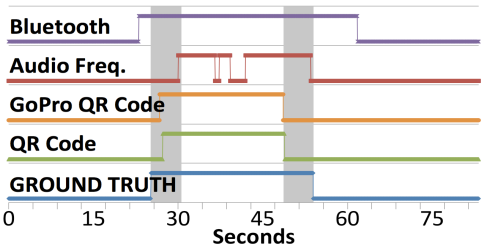
Hybrid techniques can improve performance. Technologies can be combined to exploit their respective strengths. For example, in the Person trace, we used Bluetooth (which has a wider range) to bootstrap person removal using color or the Kinect's person detection (which can localize the object to be modified in a frame). This result challenged our initial intuition that policies should be detectable via the same sensor as the events to which the policy applies. We reasoned that if the system is in a state where it misses a policy trigger (e.g., the camera is off), then it should also be in a state to miss the events to which the policy applies (e.g., RGB events). However, this proposal fails due to differences in the characteristics of different policy technologies.

Bootstrap remote policies early with passports. We experimented with passports in the Person trace, using Bluetooth and QR codes to load person removal policies. Passports let us easily extend our system without rebuilding it. Additionally, the latency incurred by loading policy code can be hidden by bootstrapping it as early as possible. It took on average 227.6 ms (10 trials) to load and install a passport (197.9 ms of which is network latency), increasing the start lag by 5-6 RGB frames if the policy is to be used immediately. Communicating a person's opt-out via BLE, rather than in a QR code on the person, would thus allow enough time to load the policy before the person is encountered.

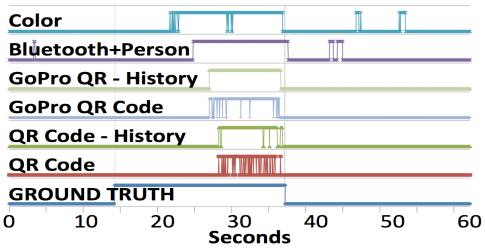
Accuracy and latency may conflict. Recall that we trade off policy accuracy with performance in our implementation (Section 5). Rather than waiting to dispatch an RGB event until the corresponding QR event arrives, we dispatch all RGB events immediately, relying on the most recent (possibly out of date) QR code event to detect a policy. We observe that start lag consists of two components: (1) the time it takes to be in range of the policy (e.g., close enough to the QR code), and (2) the number of unmodified events after the policy comes into range but the trigger is not yet recognized. Our choice to trade off accuracy for performance potentially affects the second component.

We measured the effect of this implementation choice for QR codes. We find that the accuracy impact is negligible: in the Bathroom trace, the difference is just one frame (40 ms). That is, one RGB event contained a QR code and was let through, but the policy was applied by the next RGB event. In the Person trace, the lag is four frames (160 ms); these accuracy differences are not discernible in Figure 9. However, the same performance lag may noticeably impact the user in some cases: to avoid distracting lag between the real and virtual objects in augmented reality, for example, sensor input events must be dispatched in as close to real-time as possible [2]. Systems may differ in their requirements.

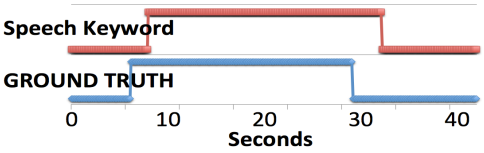
Better technology will improve accuracy. Because the Kinect has a relatively low RGB resolution (640x480, up to 30 frames per second), in two traces we simultaneously collected additional video using a GoPro camera (1280x1080, up to 60 fps) affixed to the Kinect (included in Figures 9a–b.) We synchronized Kinect and GoPro frames by hand, downsampling the GoPro points and marking the policy as enforced if any collapsed frames were blocked or modified. The barcode library could decode QR codes up to one second earlier in the GoPro traces, with fewer false negatives. The GoPro's wide-angle lens also allowed it to decode QR



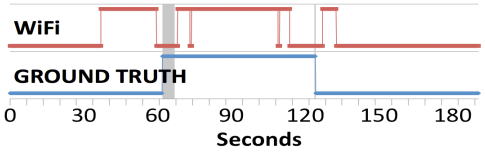
(a) Bathroom Scenario: RGB-Blocking Policies



(b) Person Scenario: RGB-Modifying Policies



(c) Speech Scenario: Audio-Blocking Policy



(d) Corporate Scenario: RGB-Blocking Policy

Figure 9: **Policy Evaluation.** *The grey regions were annotated as "depends" in the ground truth, i.e., the human annotator believed that either blocking or not blocking the event would be acceptable; these events are not counted in Figure 8. Figure 6 shows an example.*

codes in the periphery (e.g., near but not on the bathroom door) that the Kinect did not see. Thus, as mobile cameras improve, so will their ability to detect world-driven policies.

We expect that other cases in which our system failed to match ground truth will also be improved by better technology. For example, the speech keyword recognizer is slow, resulting in large start and finish lags. Similarly, the start lag is long for all person removal policies (Figure 9b) because the trace involves approaching the person down a hall, and no method is effective until the camera is close enough.

World-driven access control is practical. Our experiences suggest that world-driven policies can be expressed with modest developer effort, that the system can be seamlessly extended with passports, and that policies can be detected and enforced with reasonable accuracy given the right choice of technology. Even without advances in computer vision or other technologies, world-driven access control already significantly raises the bar in some scenarios — e.g., the accuracy of our system in the bathroom setting.

7. REFLECTIONS AND FUTURE WORK

Continuous sensing applications will become increasingly prevalent as technologies like Google Glass and Microsoft Kinect become more ubiquitous and capable. World-driven access control provides a new alternative to controlling access to sensor streams. As our work uncovers, important challenges arise when attempting to instantiate this approach in a real system. One key challenge is the tradeoff between policy detection accuracy and enforcement latency. A second key challenge is the need to verify the authenticity of signals communicated from real-world objects. Though we designed and evaluated specific approaches for overcoming both challenges, future insights may lead to alternate solutions. We highlight these challenges in this section, with the hope of providing targets for and benefiting future researchers focused on access control for continuous sensing.

We also highlight a separate challenge: the tension between the specificity of the information broadcast in a policy and an object's privacy. While specific object descriptions (e.g., "no pictures of car with license plate 123456") are valuable both for policy verification and for extending the system's object recognition capabilities, they reveal information to anyone receiving the broadcast. We view this issue as separate from our study of how to effectively communicate and manage world-driven policies, our primary goal, but we believe it is important to consider if world-driven access control systems are deployed with highly expressive policies. Currently, policy creators must balance policy accuracy and description privacy, considering evolving social norms in the process. Since specific object descriptions in passports are nevertheless valuable for verifying that a policy has not been moved by an adversary, we encourage future work on privacy-preserving, expressive policies (e.g., encrypted policies accessible only by trusted policy modules).

A separate privacy issue arises with the use of cryptographic signatures on policies: if the signatures on each object are unique, then they can be used to track objects; we observe, however, that there are many other methods to track objects today (e.g., RFIDs, Bluetooth IDs, MAC addresses), and hence consider the issue out of scope.

Finally, as described, world-driven access control can use diverse mechanisms to communicate policies. We implemented and evaluated several approaches for explicitly communicating policies, and we encourage further study of other approaches. For example, our design is general enough to encompass *implicitly communicated* policies. Implicit policies may rely on improved computer vision (e.g., to recognize sensitive locations, as in PlaceAvoider [39]) or be inferred from natural human behavior (e.g., closing a door or whispering). Other examples of work in this area are Legion:AR [21], which uses a panel of humans to recognize activities, and work on predicting the cost of interruption from sensors [17]. Additionally, to improve policy detection accuracy, systems may include additional *policy-specific sensors*, such as cameras, whose sole purpose is to sense policies, allowing them to be lower-power than user-facing sensors [23].

8. ADDITIONAL RELATED WORK

We have discussed inline prior works that consider bystander privacy, allowing them to opt out with visual (e.g., [36]) or digital markers (e.g., [7, 16]). TagMeNot [8] proposes QR codes for opting out of photo tagging. These designs rely on compliant recording devices or vendors; other systems aim to prevent recording by uncooperative devices, e.g., by flashing directed light at cameras [32]. Recent work studied bystander reactions to augmented reality devices and sur-

veyed design axes for privacy-mediation approaches [11].

Others have considered the privacy of device users. Indeed, Starner considered it a primary challenge for wearable computing [38]. Though wearable devices can improve security (e.g., with location-based or device-based access control, as in Grey [5]), data collected or sent by the device can reveal sensitive information. Such concerns motivate privacy-preserving location-based access control (e.g., [4]). Here, we assume that the device and system are trustworthy.

Others have also restricted applications' access to perceptual data. Darkly [19] integrates privacy protection into OpenCV, and the "recognizer" abstraction [18] helps limit applications' access to objects in a sensor feed; we build on these ideas. PlaceAvoider [39] identifies images captured in sensitive locations and withholds them from applications. PiBox [22] restricts applications from secretly leaking private data, but provides weak guarantees when users explicitly share it. Our approach helps users avoid accidentally sharing sensitive content with untrusted applications.

9. CONCLUSION

Continuous sensing applications on platforms like smartphones, Google Glass, and XBox Kinect raise serious access control challenges not solved by current techniques. We introduced *world-driven access control*, by which real-world objects specify per-application privacy policies. This approach aims to enable permission management at the granularity of objects rather than complete sensor streams, and without explicitly involving the user. A trusted platform detects these policies in the environment and automatically adjusts each application's "view" accordingly.

We built an end-to-end world-driven access control prototype, surfacing key challenges. We introduced *passports* to address policy authenticity and to allow our system to be dynamically extended. We mitigated detection inaccuracies by combining multiple policy communication techniques and introducing memory into policies. Finally, we explored the tradeoffs between policy accuracy and performance, and between broadcast policies and privacy.

Our prototype enforces many policies, each expressed with modest developer effort, with reasonable accuracy even with today's technologies and off-the-shelf algorithms. Our experiences suggest that world-driven access control can relieve the user's permission management burden while preserving functionality and protecting privacy. Beyond exploring a new design point for access control in continuous sensing, this work represents a step toward *integrating physical objects into a virtual world*; we believe our explorations lay the groundwork for future work in this space.

Acknowledgements

We thank Greg Akselrod, Ray Cheng, Lydia Chilton, Jon Howell, Jaeyeon Jung, Benjamin Livshits, Eyal Ofek, Matthai Philipose, Stuart Schechter, Will Scott, Alex Takakuwa, and Margus Veanes for valuable discussions and feedback on earlier drafts; we thank Weidong Cui for his help with evaluation traces. This work was done while the first and fourth author were visiting Microsoft Research. This work is supported in part by a National Science Foundation Graduate Research Fellowship under Grant DGE-0718124 and a Microsoft Research PhD Fellowship.

References

[1] ZXing.Net. http://zxingnet.codeplex.com/.

[2] Abrash, M. Latency – the sine qua non of AR and VR, 2012. http://bit.ly/UbrBL0.

[3] Ada Initiative. Another way to attract women to conferences: photography policies, 2013. http://bit.ly/1bc3x3O.

[4] Ardagna, C. A., Cremonini, M., di Vimercati, S. D. C., and Samarati, P. Privacy-enhanced Location-based Access Control. In *Handbook of Database Security*. 2008, pp. 531–552.

[5] Bauer, L., Garriss, S., McCune, J. M., Reiter, M. K., Rouse, J., and Rutenbar, P. Device-enabled authorization in the Grey system. In *International Conference on Information Security* (2005).

[6] Borisov, N., and Brewer, E. A. Active certificates: A framework for delegation. In *Network and Distributed System Security Symposium (NDSS)* (2002).

[7] Brassil, J. Technical Challenges in Location-Aware Video Surveillance Privacy. In *Protecting Privacy in Video Surveillance*, A. Senior, Ed. 2009, pp. 91–113.

[8] Cammozzo, A. TagMeNot. http://tagmenot.info/.

[9] Clark, J., and van Oorschot, P. C. SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. *IEEE Symposium on Security & Privacy* (2013).

[10] CNXSoft. Qualcomm fast computer vision sdk, 2011. http://bit.ly/rUY7Pa.

[11] Denning, T., Dehlawi, Z., and Kohno, T. In situ with bystanders of augmented reality glasses: Perspectives on recording and privacy-mediating technologies. In *ACM CHI* (2014).

[12] Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., and Wagner, D. Android permissions: User attention, comprehension, and behavior. In *Symposium on Usable Privacy and Security (SOUPS)* (2012).

[13] Geambasu, R., Levy, A. A., Kohno, T., Krishnamurthy, A., and Levy, H. M. Comet: An active distributed key-value store. In *USENIX OSDI* (2010).

[14] Google. Google Glass. http://glass.google.com/.

[15] Gray, R. The places where Google Glass is banned, Dec. 2013. http://www.telegraph.co.uk/technology/google/10494231/The-places-where-Google-Glass-is-banned.html.

[16] Halderman, J. A., Waters, B., and Felten, E. W. Privacy Management for Portable Recording Devices. In *Workshop on Privacy in Electronic Society* (2004).

[17] Hudson, S., Fogarty, J., Atkeson, C., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J., and Yang, J. Predicting human interruptibility with sensors: a wizard of oz feasibility study. In *ACM CHI* (2003).

[18] Jana, S., Molnar, D., Moshchuk, A., Dunn, A., Livshits, B., Wang, H. J., and Ofek, E. Enabling Fine-Grained Permissions for Augmented Reality Applications with Recognizers. In *USENIX Security Symposium* (2013).

[19] Jana, S., Narayanan, A., and Shmatikov, V. A Scanner Darkly: Protecting User Privacy from Perceptual Applications. In *IEEE Symposium on Security and Privacy* (2013).

[20] Kotadia, M. Jamming device aims at camera phones, 2003. http://cnet.co/HEvS8b.

[21] Lasecki, W., Song, Y. C., Kautz, H., and Bigham, J. Real-time crowd labeling for deployable activity recognition. In *Computer Supported Cooperative Work (CSCW)* (2013).

[22] Lee, S., Wong, E., Goel, D., Dahlin, M., and Shmatikov, V. PiBox: A platform for privacy preserving apps. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2013).

[23] LiKamWa, R., Priyantha, B., Philipose, M., Zhong, L., and Bahl, P. Energy characterization & optimization of image sensing toward continuous mobile vision. In *MobiSys* (2013).

[24] Lioy, A., and Ramunno, G. Trusted computing. In *Handbook of Information and Communication Security*, Stavroulakis and Stamp, Eds. 2010, pp. 697–717.

[25] Marlinspike, M. Convergence. `http://convergence.io/`.

[26] Meta. Spaceglasses. `http://spaceglasses.com`.

[27] Microsoft. App. Domains. `http://msdn.microsoft.com/en-us/library/2bh4z9hs(v=vs.110).aspx`.

[28] Microsoft. Creating your own code access permissions, 2013. `http://bit.ly/HFzDKD`.

[29] O'Brien, K. Swiss Court Orders Modifications to Google Street View, 2012. `http://nyti.ms/L3cdNZ`.

[30] Panzarino, M. Inside the revolutionary 3d vision chip at the heart of google's project tango phone, Feb. 2014. `http://tcrn.ch/1fkCuWK`.

[31] Paruchuri, J. K., Cheung, S.-C. S., and Hail, M. W. Video data hiding for managing privacy information in surveillance systems. *EURASIP Journal on Info. Security* (Jan. 2009), 7:1–7:18.

[32] Patel, S. N., Summet, J. W., and Truong, K. N. BlindSpot: Creating Capture-Resistant Spaces. In *Protecting Privacy in Video Surveillance*, A. Senior, Ed. 2009.

[33] Priyantha, N. B., Miu, A. K. L., Balakrishnan, H., and Teller, S. J. The cricket compass for context-aware mobile applications. In *Mobile Computing and Networking* (2001).

[34] Quest Visual. WordLens: See the world in your language. `http://questvisual.com/`.

[35] Roesner, F., Kohno, T., Moshchuk, A., Parno, B., Wang, H. J., and Cowan, C. User-driven access control: Rethinking permission granting in modern operating systems. In *IEEE Symposium on Security and Privacy* (2011).

[36] Schiff, J., Meingast, M., Mulligan, D. K., Sastry, S., and Goldberg, K. Y. Respectful Cameras: Detecting Visual Markers in Real-Time to Address Privacy Concerns. In *International Conference on Intelligent Robots and Systems* (2007).

[37] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. Real-time human pose recognition in parts from a single depth image. In *Computer Vision & Pattern Recognition* (2011).

[38] Starner, T. The Challenges of Wearable Computing: Part 2. *IEEE Micro 21*, 4 (2001), 54—67.

[39] Templeman, R., Korayem, M., Crandall, D., and Kapadia, A. PlaceAvoider: Steering first-person cameras away from sensitive spaces. In *Network and Distributed System Security Symposium (NDSS)* (2014).

[40] Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., and Minden, G. J. A Survey of Active Network Research. *IEEE Communications 35* (1997), 80–86.

[41] The 5 Point Cafe. Google Glasses Banned, Mar. 2013. `http://the5pointcafe.com/google-glasses-banned/`.

[42] Tom Simonite. Bringing cell-phone location-sensing indoors. `http://bit.ly/TVyMEx`.

[43] Wendlandt, D., Andersen, D. G., and Perrig, A. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX Security Symposium* (2008).

[44] Zimmermann, P. R. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.

Place	*Policy*
Mercury	Ask before photo.
Mercury	Avoid people in background of photo.
5Point Cafe	No photos, no Google Glass.
Goodwill	No recording w/in 15 ft. of dressing room.
Local Gym	No cell phone in locker room.
AdaCamp SF	Lanyard color indicates photo preference.
Open Source Bridge	Provide notice before taking photo.
Sirens	Ask before photo.
Sirens	No photos during sessions.
WisCon	Ask before photo.
Street View	Blur sensitive areas, no photos over fences.

Figure 10: **Existing Real-World Policies.** *Today, multiple places request that people follow photo and video policies. We report a sample set of places, along with the policy they request. These policies can be captured by our framework.*

APPENDIX

A. REAL-WORLD POLICIES

We summarize a sample set of real-world locations with sensor privacy policies (Figure 10) that would benefit from world-driven access control. These scenarios motivate the incentive of establishments to install world-driven policies.

The first two settings are bars that request patrons refrain from taking recordings of others without permission. For example, the 5 Point Cafe recently banned Google's Glass device due to privacy concerns [41] and other Glass bans have followed [15]. Similarly, the "local gym" and "Goodwill" settings both prohibit recording devices near dressing rooms. All of these places could specify an explicit "no pictures" policy for a world-driven access control device.

The next four settings are conventions that set explicit policies around photography [3], generally requiring asking before taking a photo or prohibiting recording in certain times or locations (e.g., during a conference session). One policy is more fine-grained, based on explicit communication from attendees using colored lanyards (where a red lanyard means "no photos," a yellow lanyard means "ask first," and a green lanyard means "photos OK"). These colors could serve as the foundation for a world-driven access control policy. As suggested in prior work, such individual policies could also be communicated using a marker on a shirt [36] or via a communication from the bystander's phone (e.g., [7, 16]).

Finally, Google's Street View product, which is powered by cars that regularly drive through the world and capture panoramic views, has raised significant privacy concerns. Switzerland only recently allowed Google to collect Street View data, but it required Google to blur sensitive areas and refrain from taking pictures across fences and garden hedges [29]. These policies could also be enforced by world-driven access control, based on the locations of sensitive areas or presence of fences.

While today's real-world policies tend to focus on the privacy of bystanders, always-on devices like Google Glass will also raise privacy concerns for users. For example, our findings in a user survey (not described here) suggest that many users do not wish to be recorded by their own wearable devices in sensitive locations (e.g., at home, in the bathroom).