Small mention of interesting things

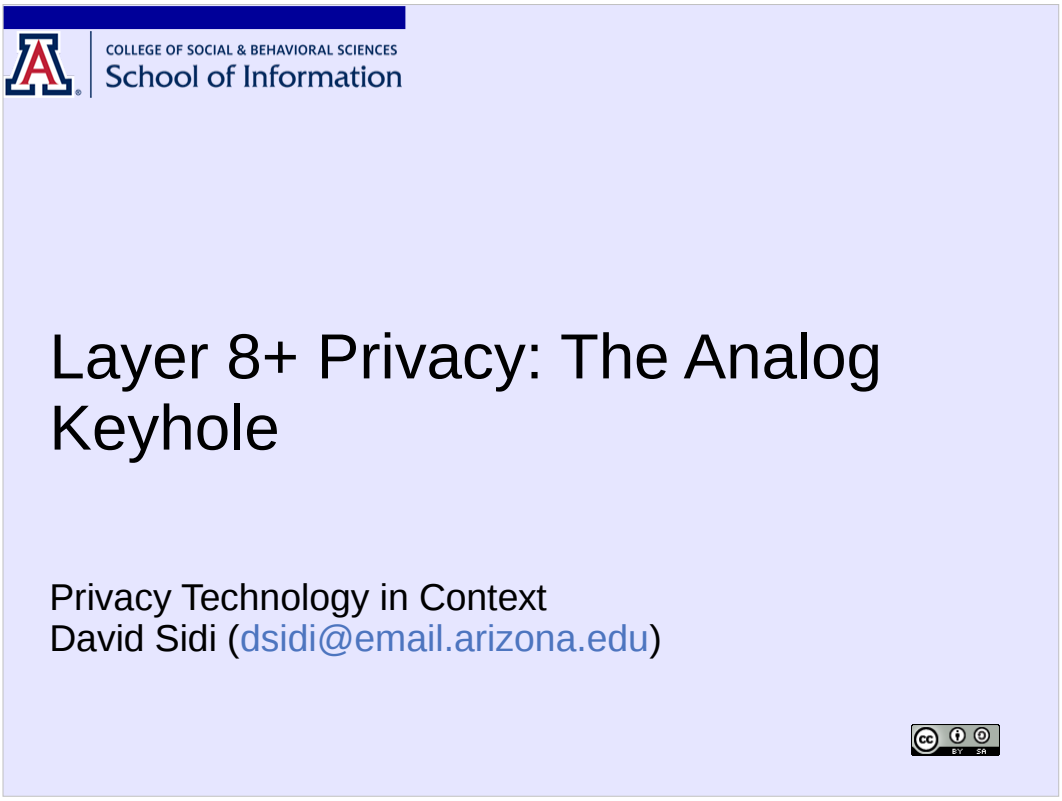
- Schedule:
  - Final server assignment deadline will be extended to the 29th
  - Final projects will be due the day of the final
- TCEs

I’ve moved the small mentions to before the title slide, since it makes more sense this way.

Small mention of interesting things

Small mention of interesting things

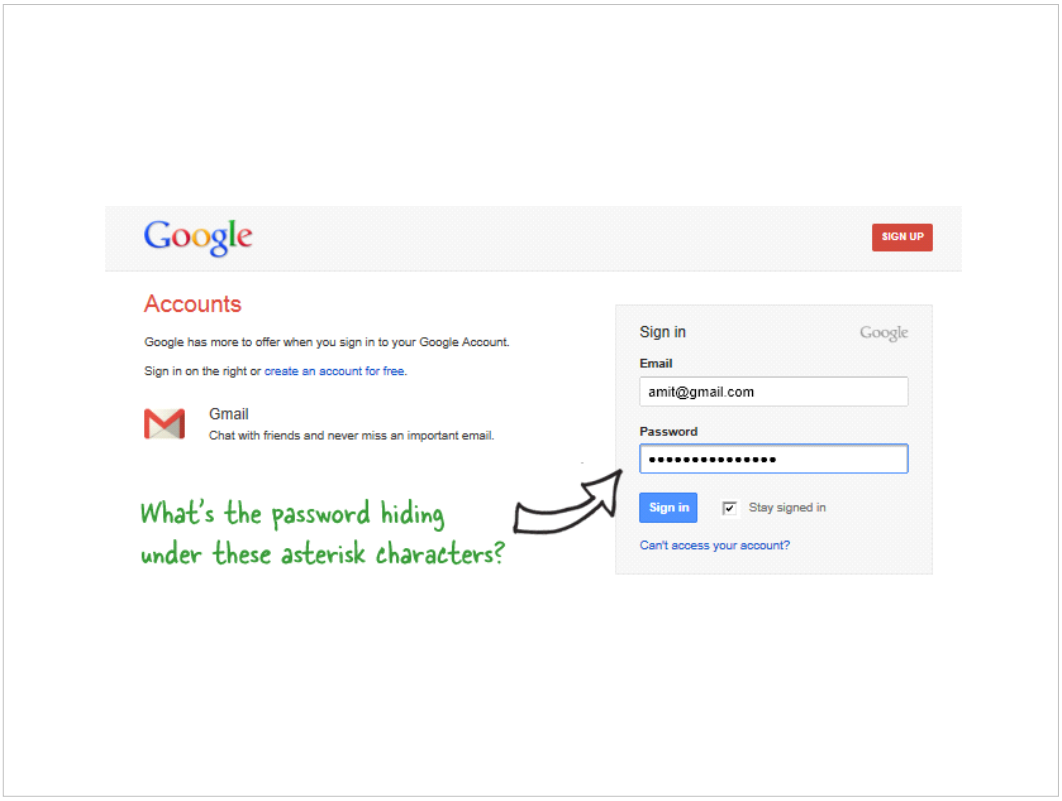
- Bodyguard FLARE home security camera (link. Also, among the funniest videos I've seen)
- A depolarized monitor matched to polarizing glasses (link)



Today we talk about what happens when there are automated systems ready to interpret what is available from the analog hole. In recent work we've called this "the analog keyhole problem."

You can think of the analog keyhole problem as industrialized shoulder surfing.

What is regular shoulder-surfing?



shoulder surfing is the reason that characters are not shown as you type in your password.

How effective is that? Let's say for simplicity a service requires passwords to be between 6 and 8 characters long, and can use any upper or lowercase letter, any number, and the symbols % and @. How many guesses to exhaust the set of possible passwords?

Now suppose you see 7 character dots as a person puts in their password. How many guesses now? (consider 3T guess/s). Rate limiting.

DARKLY is for only a limited threat related to the AKP

- Suppose you own devices with perceptual capabilities and want to be sure that the apps that use those capabilities don't misbehave
- Darkly (@ 31:49 - 43:00)
- Trust includes
  - device operating system
  - the hardware of its perceptual sensors
- Trust does not include a third party application running on your device

One version of the AKP involves ensuring that applications that need access to sensors do not have unlimited access.

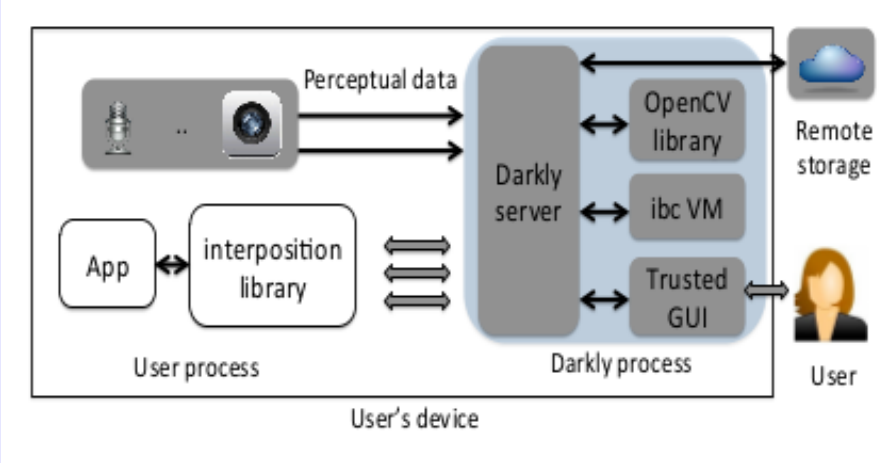
If an app needs to look at you to see what gesture you're doing to indicate the video you want to watch, what's to stop it from also using OCR to check for credit card numbers, or passwords?

Here nonthreats include the device OS and its hardware. The worry is about software installed on an otherwise trusted device.

DARKLY focuses on sensors for video processing (it is specific to OpenCV).

Opaque References

- “the application will never have access to the raw pixels”



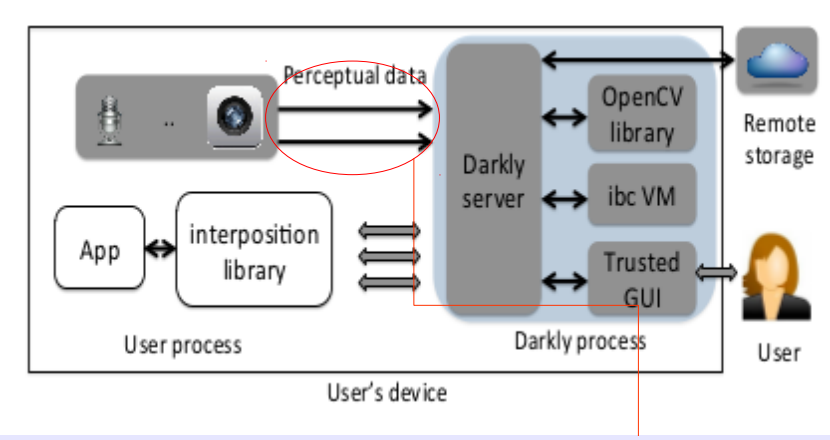
Here’s the architecture of DARKLY. It sits between the vision library and the app, and prevents the app from getting more vision data than it needs. No raw pixels!

The triple arrows are standard OS user isolation, which we’ll clarify in a second. Notice for now that part of DARKLY runs on the app side, and part on the OS side (inputs, OS, sensors are trusted).

To block direct access to raw images, DARKLY replaces OpenCV’s pointers to pixels with opaque references that cannot be dereferenced by applications. Applications can still pass them as arguments into OpenCV functions, which dereference them internally and access the data.

Opaque References

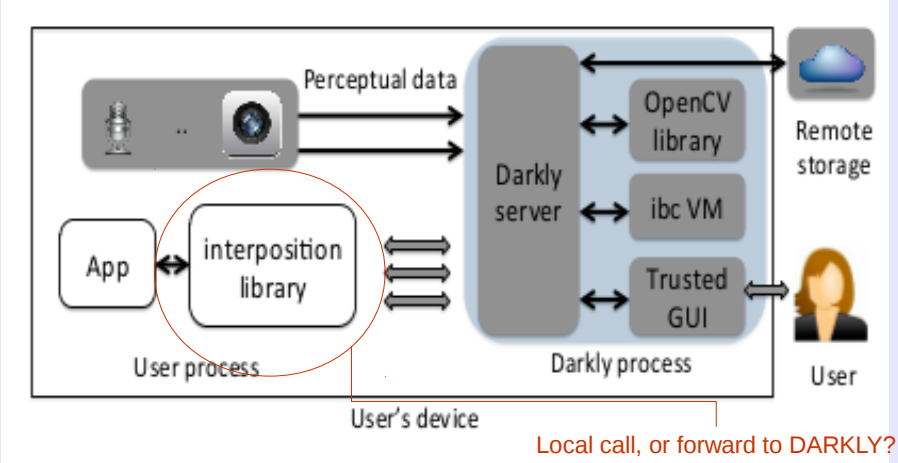
- "the application will never have access to the raw pixels"
  - opaque references



D ARKLY exploits the fact that the lower part of the address space is typically reserved for the OS code, and therefore all valid pointers must be greater than a certain value. For example, in standard 32-bit Linux binaries, all valid stack and heap addresses are higher than 0x804800. The values of all opaque references are below this address. It only does this for the pixel data inside the data structures; it leaves metadata alone.

Opaque References

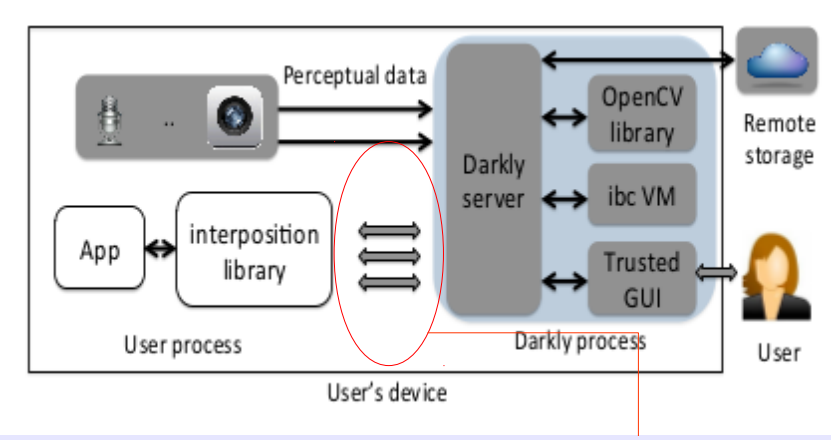
- "the application will never have access to the raw pixels"
  - opaque references



For each call made by an application to an OpenCV function, the interposition library must decide whether to execute it within the application or forward it to the trusted D ARKLY server running as a separate "user" on the same device (only this server has access to camera inputs).

Opaque References

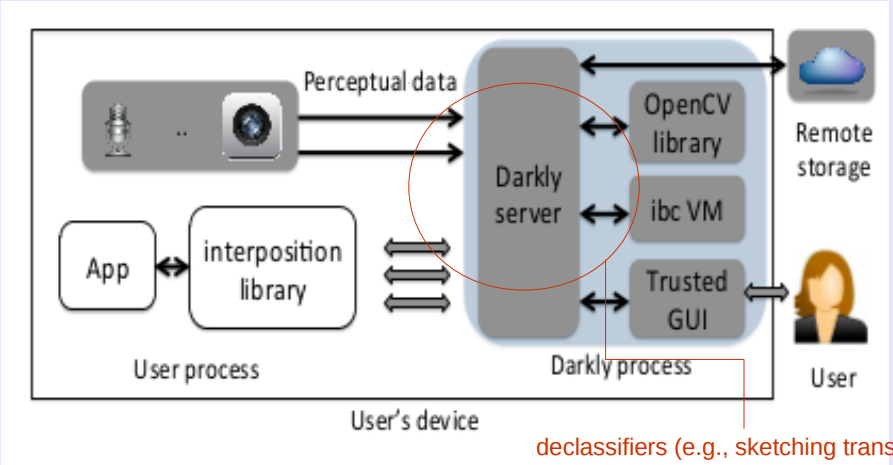
- “the application will never have access to the raw pixels”



If there is at least one argument with an opaque reference, executing the function requires access to the image. The interposition library marshals the local arguments and opaque references, and forwards the call to D ARKLY for execution. If none of the arguments contain an opaque reference, the function does not access the image and the interposition library simply calls the function in the local OpenCV library.

Declassifiers for privacy transformation

- "the application will never have access to the raw pixels"
  - opaque references



Sometimes an application can't work only by composing calls to OpenCV, and so needs some kind of access to the visual data itself. The answer DARKLY provides here is to degrade the visual data so that it is as minimal as possible for the use needed by the application.

That's an ad-hoc answer, notice.

Here you can see that this is a credit card, but you can't recover the number.

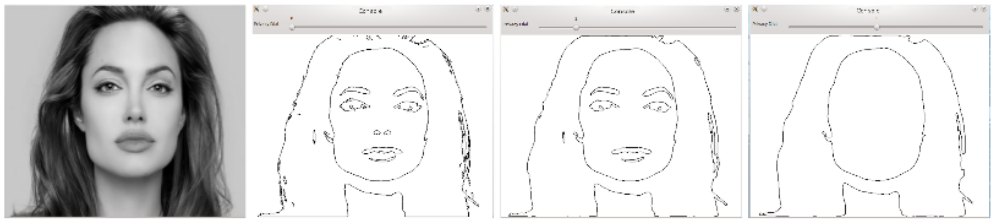


Figure 2. Output of the sketching transform on a female face image at different privacy levels.

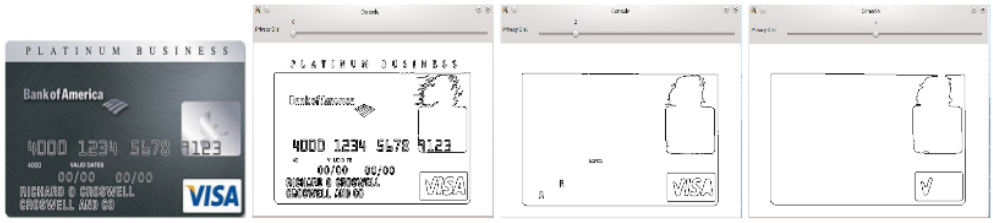


Figure 3. Output of the sketching transform on a credit card image at different privacy levels.

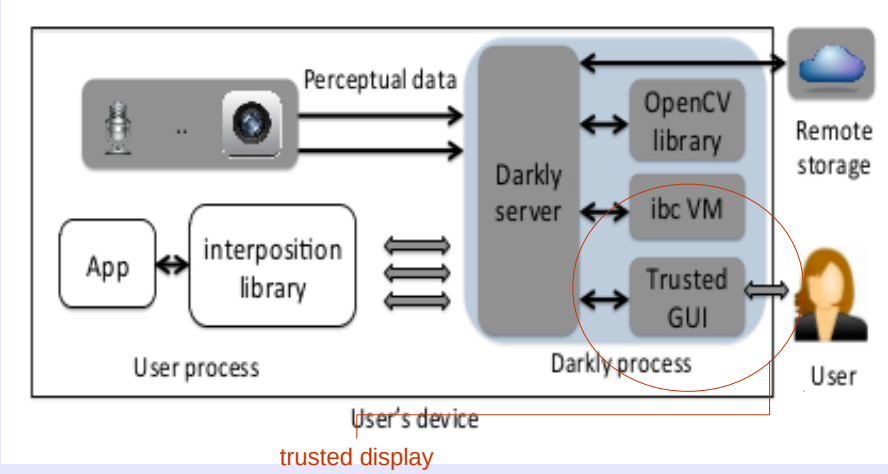
Sometimes an application can’t work only by composing calls to OpenCV, and so needs some kind of access to the visual data itself. The answer DARKLY provides here is to degrade the visual data so that it is as minimal as possible for the use needed by the application.

That’s an ad-hoc answer, notice.

Here you can see that this is a credit card, but you can’t recover the number.

Trusted GUI and Storage

- “the application will never have access to the raw pixels”

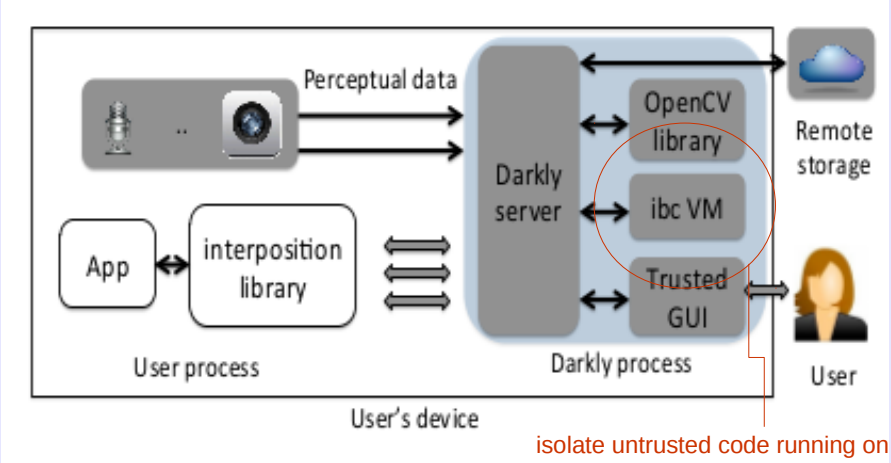


There is a second layer of “field verifiability” as a failsafe. The user is asked about the level of the privacy transform, and can adjust it. This involves use of a trusted display.

The trusted display actually serves a dual purpose: (1) an application can use it to show images to which it does not have direct access, and (2) it shows to the user the privacy-transformed features and objects released to the application by declassifiers. Why might this be important?

ibc for untrusted arbitrary computation

- “the application will never have access to the raw pixels” **(mostly)**



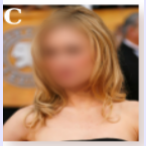
ibc is a DSL they’ve made to handle even the case in which arbitrary untrusted computation has to be performed on raw pixels (their example is eigenface algorithm for face recognition). ibc programs cannot access DARKLY ’s or OpenCV’s internal state, and can only read or write through a few D ARKLY functions. That’s a portability cost, but it at least provides an avenue for such untrusted computation on raw pixels.

DARKLY is domain-specific

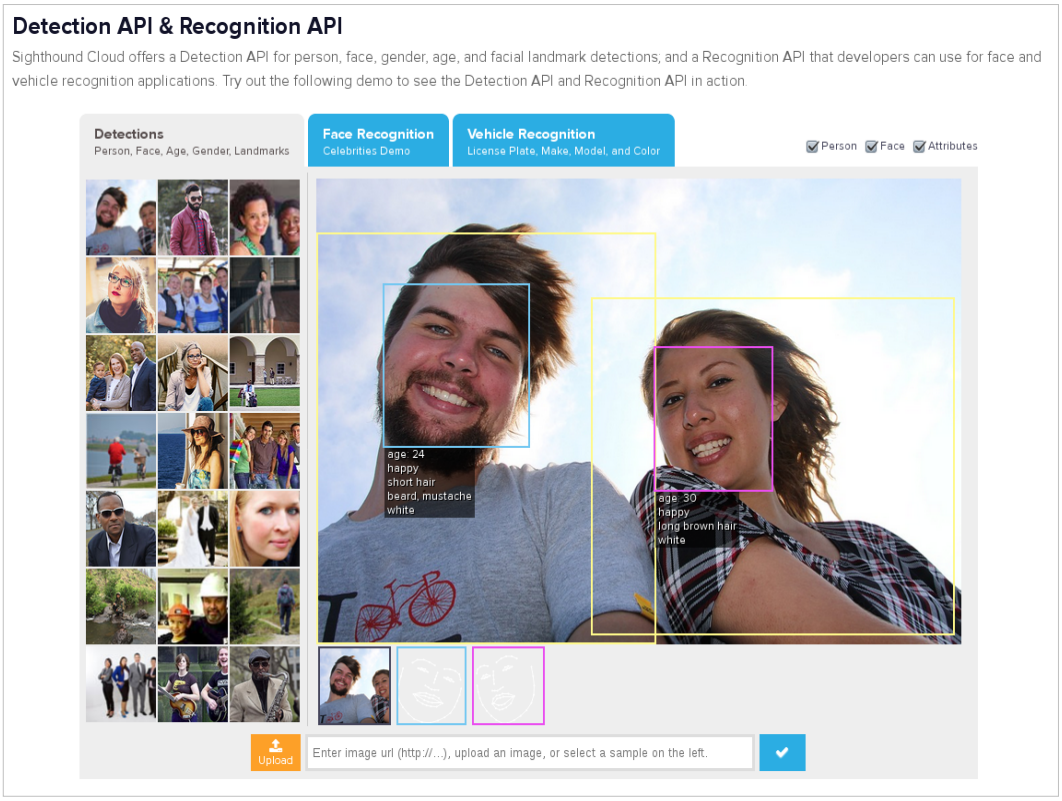
- Architecture is general in principle, but in practice lots of OpenCV specific tinkering required
  - “DARKLY exploits the fact that most OpenCV data structures for images and video include a separate pointer to the actual pixel data. For example, IplImage’s data pointer is stored in the imageData field; CvMat’s data pointer is in the data field. For these objects, DARKLY creates a copy of the data structure, fills the meta-data, but puts the opaque reference in place of the data pointer. Existing applications can thus run without any modifications as long as they do not dereference the pointer to the pixels”

Its hard to make privacy transforms *principled*

- Not always clear what a system needs to perform its work, and manual intervention is problematic
  - "The sketch of an image is intended to convey its high-level features while hiding more specific privacy-sensitive details. A loose analogy is publicly releasing statistical aggregates of a dataset while withholding individual records."
  - May reduce performance in unexpected ways
  - May reduce privacy in unexpected ways
    - Not always intuitive what privacy protections are guaranteed by different transformations of visual input: sketching transform
    - Example: Gaussian blur



What makes a transform of visual data privacy preserving, or not? Is a sketching transform like DARKLY uses a good one in all cases? How trustworthy are our intuitions about this? (Hint: not very trustworthy---see Gaussian blur).



DARKLY supposes a trusted platform---hardware, and OS. How do we get there with the proliferation of perceptually-capable devices? DARKLY thinks about devices that the user controls, but what about devices that the user wants to be able to judge trustworthy that are not under her control? There are going to be a lot of those, given the widespread availability of sophisticated vision functionality as a service.

CLOUD VISION API FEATURES

Derive insight from images with our powerful Cloud Vision API

Label Detection

Detect broad sets of categories within an image, ranging from modes of transportation to animals.

Explicit Content Detection

Detect explicit content like adult content or violent content within an image.

Logo Detection

Detect popular product logos within an image.

Landmark Detection

Detect popular natural and man-made structures within an image.

Optical Character Recognition

Detect and extract text within an image, with support for a broad range of languages, along with support for automatic language identification.

Face Detection

Detect multiple faces within an image, along with the associated key facial attributes like emotional state or wearing headwear. **Facial Recognition is not supported.**

Image Attributes

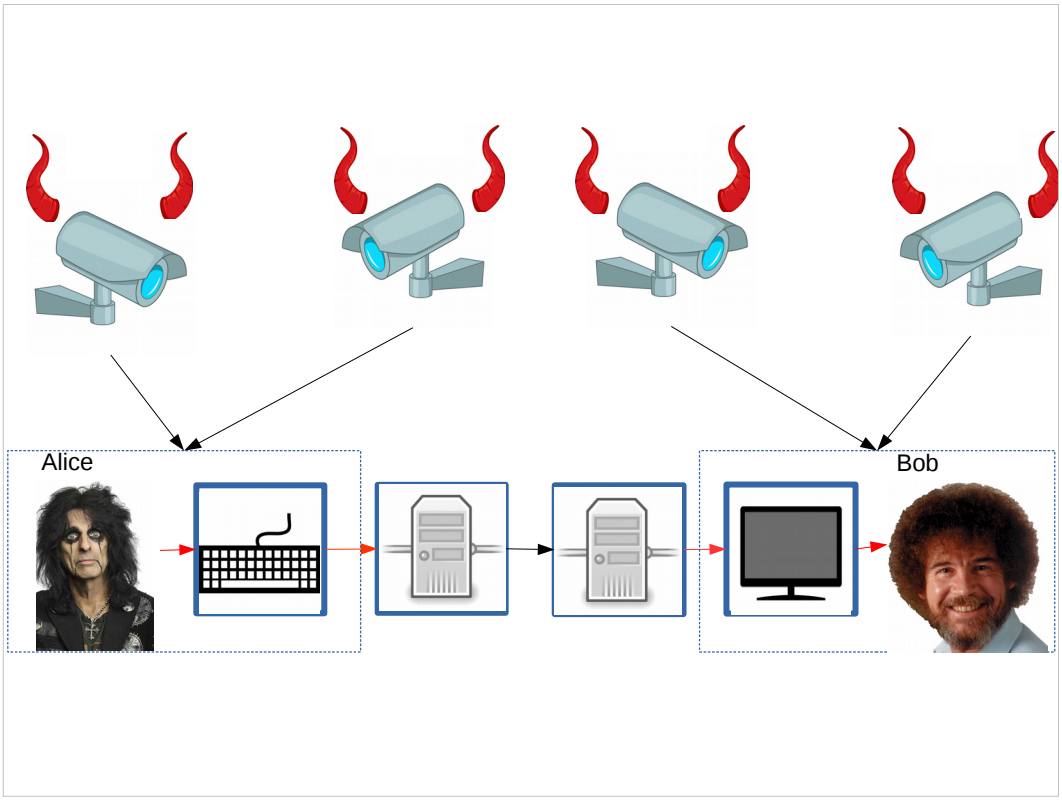
Detect general attributes of the image, such as dominant colors and appropriate crop hints.

Web Detection

Search the Internet for similar images.

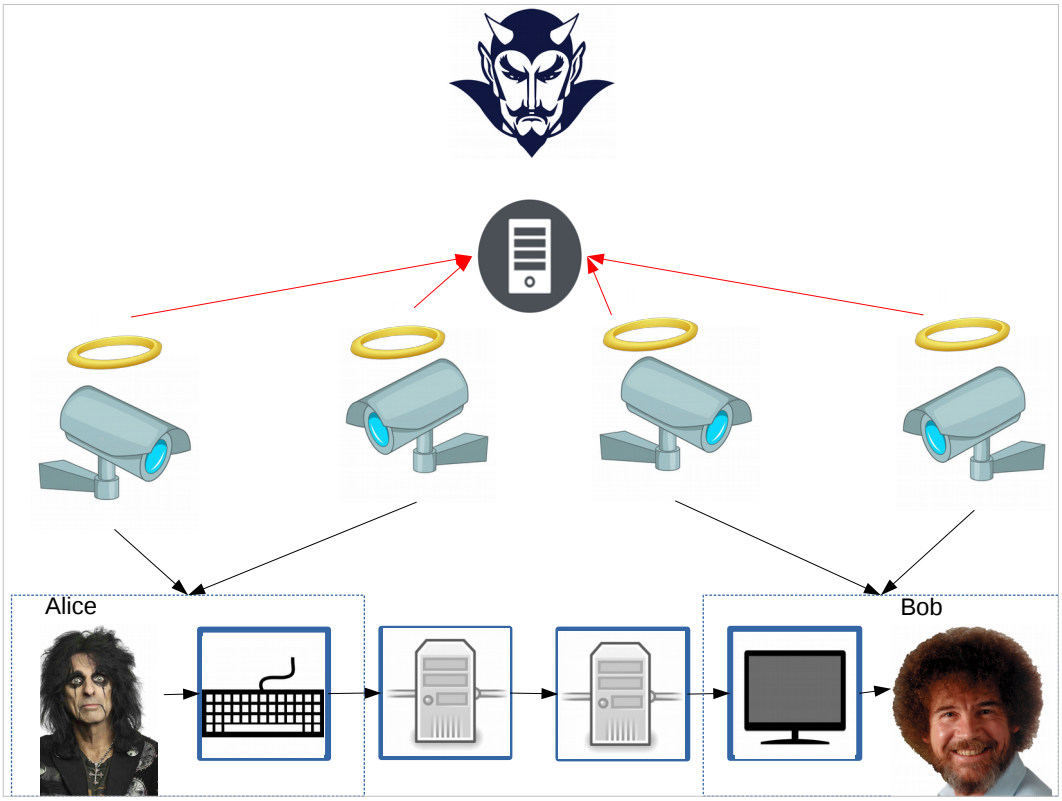
Integrated REST API

Access via REST API to request one or more annotation types per image. Images can be uploaded in the request or integrated with Google Cloud Storage.



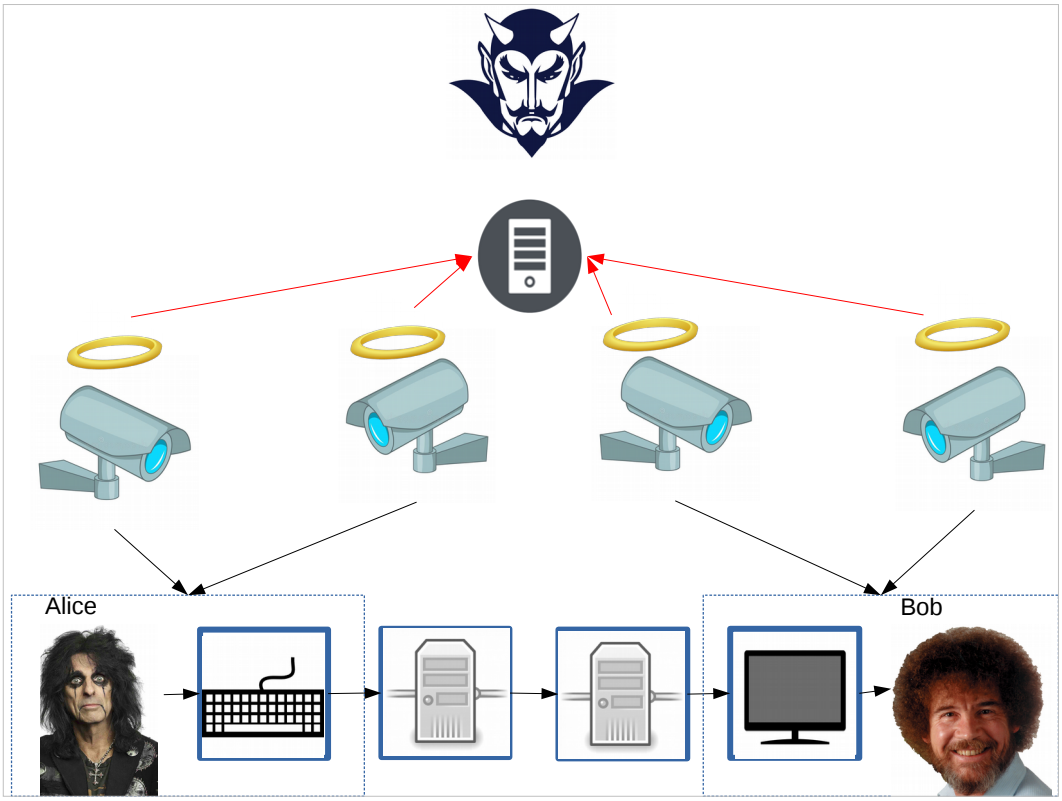
That’s hard to handle, but a clear first step is to take an infrastructural approach: provide privacy infrastructure that sits between these perceptually-capable devices and their subjects, and helps mediate their interactions.

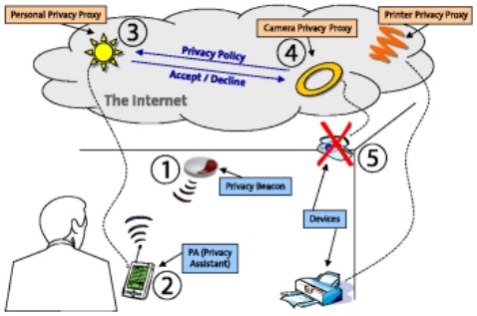
Here’s an example of that kind of architecture that I’ve worked on. (Explain this, the harder case).



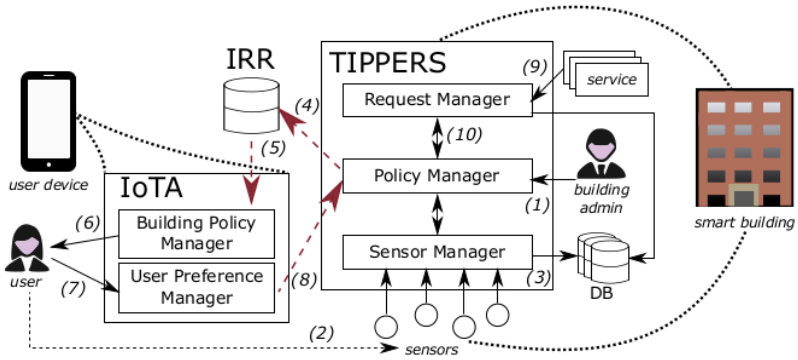
The easier case which we're interested in, inspired by DARKLY.

image credit: mcssolutions.com

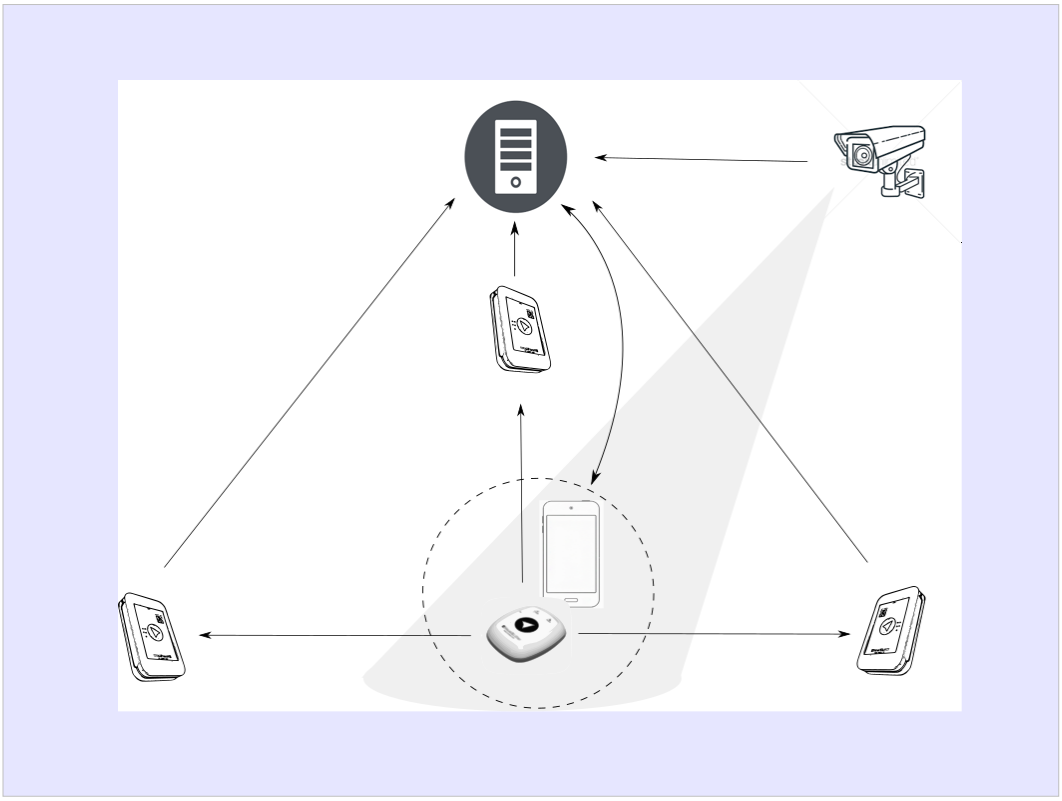

Alice
Bob



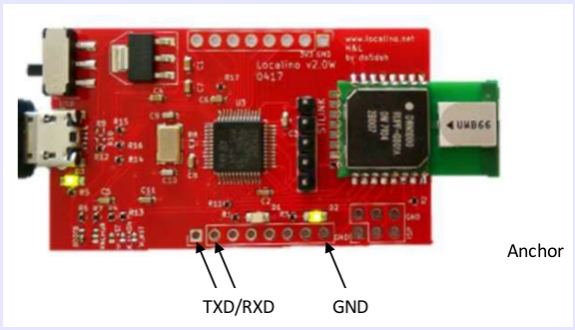
Langheinrich, 'A Privacy Awareness System for Ubiquitous Computing Environments'



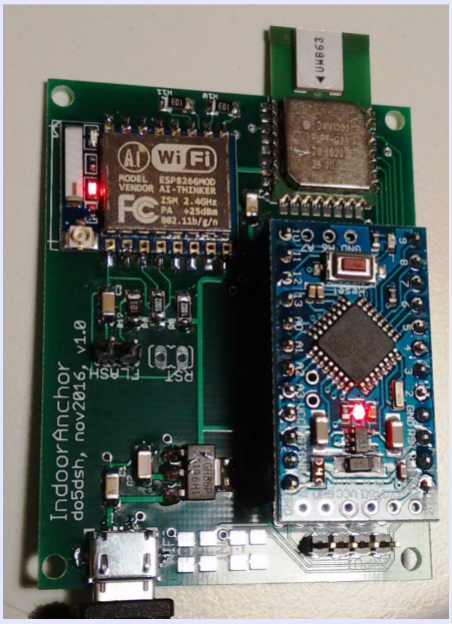
Pappachan et al., 'Towards Privacy-Aware Smart Buildings: Capturing, Communicating, and Enforcing Privacy Policies and Preferences'



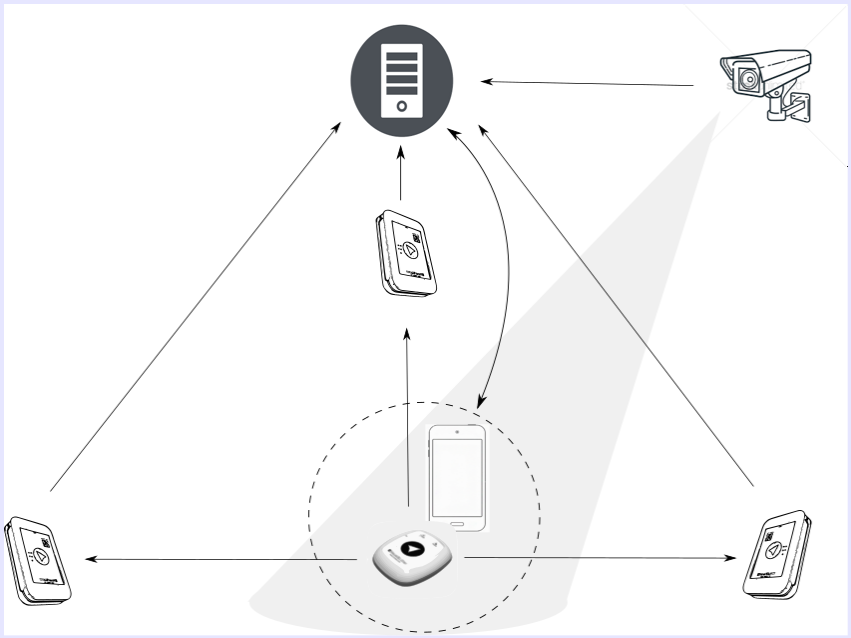
Supporting physical protection of privacy at displaytime with a notification infrastructure for video systems.

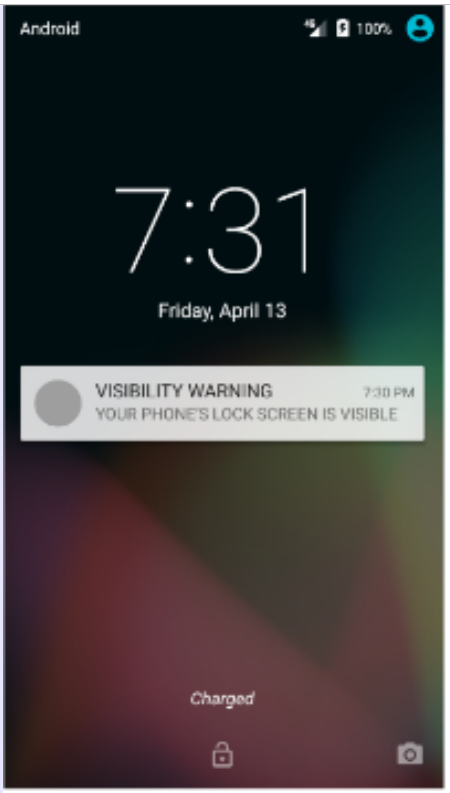


Anchor

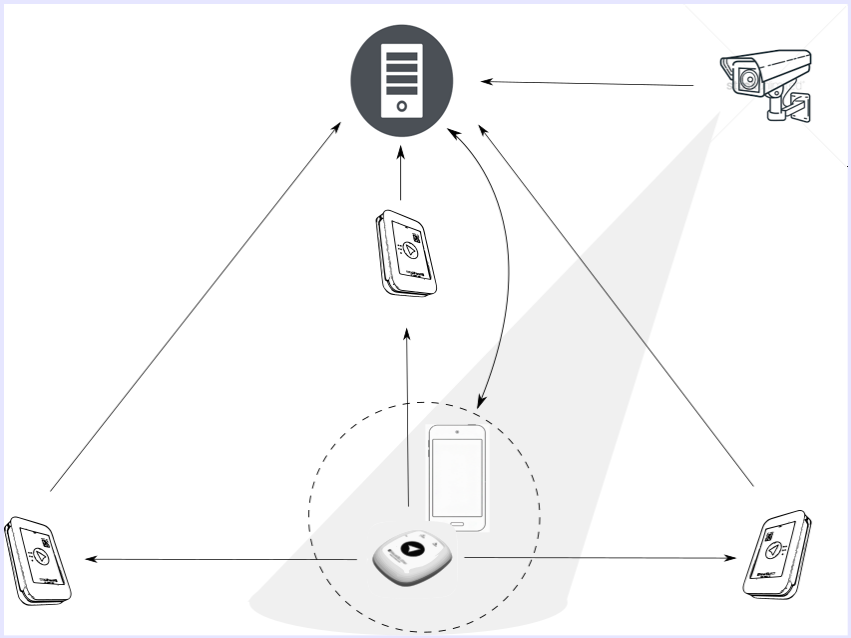


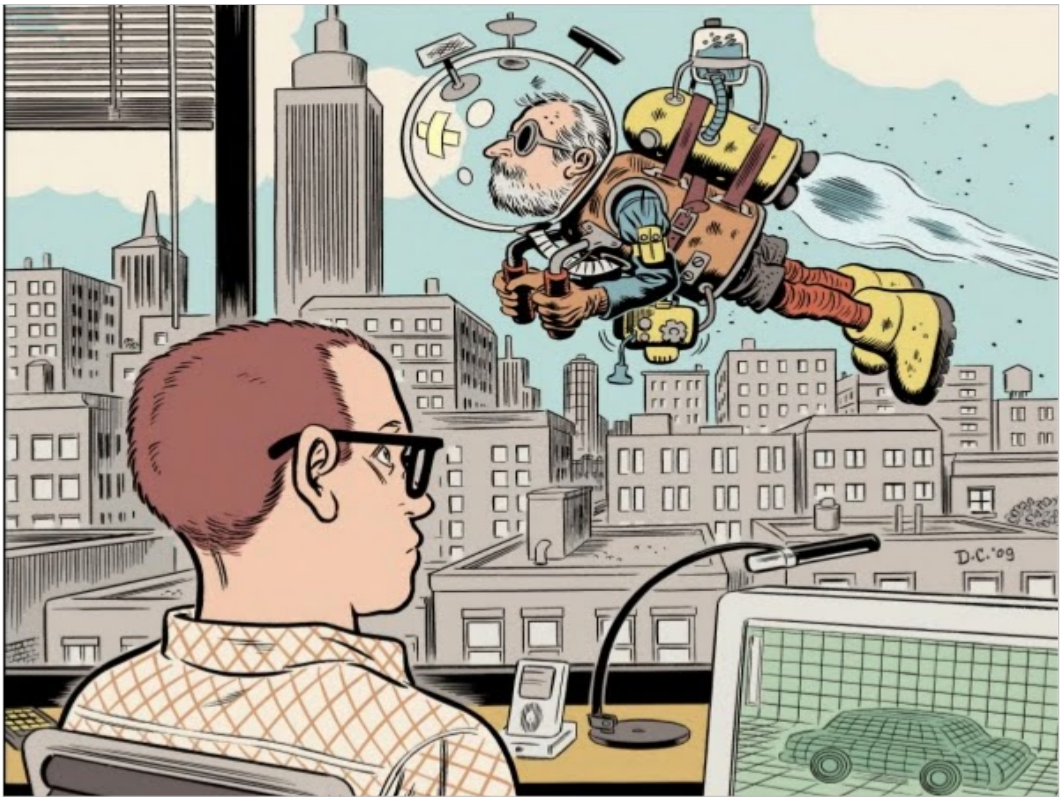
Our transceivers.





Several approaches to notification are compared.





Let’s look at OpenCV in the VM (tutorial).

Let’s also look at Haar Cascades:
https://docs.opencv.org/3.4.3/d7/d8b/tutorial_py_face_detection.html

Let’s look at toy RSA, which shows the kind of thing you’ll be doing for the DH part of the assignment (obviously it’s not directly applicable):
https://github.com/gdanezis/petlib/blob/master/examples/toyrsa.py